IEICE TRANS. INF. & SYST., VOL.E102–D, NO.3 MARCH 2019

PAPER Special Section on Foundations of Computer Science — Algorithm, Theory of Computation, and their Applications —

Evasive Malicious Website Detection by Leveraging Redirection Subgraph Similarities*

Toshiki SHIBAHARA[†,††a)], **Yuta TAKATA**[†], ***Nonmembers***, **Mitsuaki AKIYAMA**[†], **Takeshi YAGI**[†], **Kunio HATO**[†], ***Members***, ***and*** **Masayuki MURATA**[††], ***Fellow***

SUMMARY Many users are exposed to threats of drive-by download attacks through the Web. Attackers compromise vulnerable websites discovered by search engines and redirect clients to malicious websites created with exploit kits. Security researchers and vendors have tried to prevent the attacks by detecting malicious data, i.e., malicious URLs, web content, and redirections. However, attackers conceal parts of malicious data with evasion techniques to circumvent detection systems. In this paper, we propose a system for detecting malicious websites without collecting all malicious data. Even if we cannot observe parts of malicious data, we can always observe compromised websites. Since vulnerable websites are discovered by search engines, compromised websites have similar traits. Therefore, we built a classifier by leveraging not only malicious but also compromised websites. More precisely, we convert all websites observed at the time of access into a redirection graph and classify it by integrating similarities between its subgraphs and redirection subgraphs shared across malicious, benign, and compromised websites. As a result of evaluating our system with crawling data of 455,860 websites, we found that the system achieved a 91.7% true positive rate for malicious websites containing exploit URLs at a low false positive rate of 0.1%. Moreover, it detected 143 more evasive malicious websites than the conventional content-based system.

key words: *drive-by download attack, browser fingerprinting, graph mining, clustering*

1. Introduction

Attackers have distributed malware through the Web by drive-by download attacks. When a client accesses a landing URL that is a starting point of attacks, the client is redirected to an exploit URL via multiple redirection URLs. At the exploit URL, vulnerabilities in browsers and/or their plugins are exploited, and the client is finally infected with malware [2]. This infected client suffers from damage, such as sensitive data leakage and illegal money transfer, and/or is integrated into distributed denial-of-service attacks. To expose more users to threats of drive-by downloads, attackers compromise benign websites and inject redirection code to malicious websites such as redirection and exploit URLs. Attackers compromise benign websites and create malicious websites automatically. Since websites built using the old version of content management systems (CMSs) are vulnerable, attackers automatically discover them with search engines by using specific search queries, typically called "search engine dorking" [3], [4], and compromise them to turn them into landing URLs. Malicious websites are automatically created with exploit kits [5].

To prevent drive-by download attacks, security researchers and vendors analyze malicious data, e.g., malicious URLs, web content, and redirections. They can create signatures of anti-virus software and build classifiers on the basis of malicious web content [6]–[8], redirection chains [9]–[11], and exploit kits [12]. All the above systems require malicious data to be collected by accessing malicious websites with a honeyclient, which is a decoy browser [13], [14]. Unfortunately, collecting all malicious data from malicious websites is not easy because attackers conceal the data with evasion techniques. To increase the exploitation success rate, attackers check clients by browser fingerprinting and change the destination URL depending on the fingerprint, e.g., IP address and client environments (the family/version of a browser on a real operating system (OS)/virtual machine/emulator) [15], [16]. In addition, if a client environment differs from the environment of attackers' targets, the attackers thwart the client's accesses by changing the server responses of malicious websites, which is called "cloaking" [17]. In other words, collecting all malicious data requires *correct* access by the clients of attackers' targets. Although multiple accesses to malicious websites by various clients improves the coverage of collected malicious data, preparing or emulating all the clients (i.e., OSs, browsers, and plugins) is not a realistic solution due to the requirement of a large amount of computational resources [16].

In this paper, we propose a system for detecting malicious websites without collecting all malicious data. Even if we cannot observe parts of malicious data, e.g., exploit code and malware, we can always observe compromised websites, into which attackers inject redirection code to malicious data. Since vulnerable websites are automatically discovered with search engines by attackers, compromised websites have similar traits. Therefore, we built a classifier by leveraging not only malicious but also compromised websites. More precisely, we convert all websites observed at the time of access into a redirection graph, whose vertices are URLs and edges are redirections between two URLs, and classify it with a graph mining approach. To perform classification, we integrate similarities between the redi-

Manuscript received March 22, 2018.
Manuscript revised July 27, 2018.
Manuscript publicized October 30, 2018.

[†]The authors are with NTT Secure Platform Laboratories, Tokyo, 180–8585 Japan.
[††]The authors are with Osaka University, Suita-shi, 565–0871 Japan.

*An earlier version of this paper appeared in [1].

a) E-mail: toshiki.shibahara.de@hco.ntt.co.jp

DOI: 10.1587/transinf.2018FCP0007

Copyright © 2019 The Institute of Electronics, Information and Communication Engineers

rection graph's subgraphs and redirection subgraphs shared across malicious, benign, and compromised websites. As a result of evaluating our system with crawling data of 455,860 websites, we found that it achieved a 91.7% true positive rate (TPR) for malicious websites containing exploit URLs at a low false positive rate (FPR) of 0.1%. Moreover, it detected 143 more malicious websites that use evasion techniques than conventional systems. These detected evasive websites were, for example, built by compromising a vulnerable CMS. These results show that our system successfully captures redirection subgraphs of not only malicious but also compromised websites.

Our contributions are summarized as follows.

- We propose a system that detects malicious websites by leveraging all websites observed at the time of access even if all malicious data cannot be collected due to evasion techniques.
- We show that leveraging the redirection subgraphs of benign, compromised, and malicious websites enhances the classification performance; the benign subgraphs contribute to reducing false positives such as subgraphs of web advertisements and the compromised and malicious subgraphs contribute to improving true positives such as subgraphs of compromised CMSs and exploit kits.

2. Motivating Example

We use simplified websites to demonstrate the effectiveness of our approach. Figure 1 shows a redirection graph. When a client accesses the URL of a compromised website, i.e., `http://a.example/`, the server responds with web content such as Fig. 2, and the client additionally requests the web content of the URLs specified in HTML tags. The `iframe` tag at line 13 in Fig. 2 is injected by an attacker, and the client is redirected to the next URL, `http://redirect.example/`, without being aware of it because this `iframe` tag is written in an invisible state and is outside the display. When the client accesses the URL specified by the `iframe` tag, it loads web content that contains the JavaScript code shown in Fig. 3. Lines 2–6 in Fig. 3 are evasion code that checks whether the client is a browser emulator or an actual browser. A browser emulator is usually designed to never raise exceptions regarding ActiveXObject [13]. However, since the code in Fig. 3 intentionally throws an ActiveXObject error, only browsers with correct exception handlers can execute browser fingerprinting code at line 5. The code at line 5 stores the UserAgent strings of the client in a variable `ua`. The variable `ua`, i.e., `navigator.userAgent` strings, is used for the following conditional branches at lines 7 and 10, and the redirection code at line 8 or 11 is executed if the variable contains "msie 6" or "msie 8" strings, respectively. In other words, Internet Explorer (IE) 6 is redirected to `http://exploit.example/IE6/`, and IE8 is redirected to `http://exploit.example/IE8/`. However, when clients other than IE6 and IE8 are used, no redirection occurs. Browser emulators also cannot execute browser fingerprinting code due to exception handling, so no redirection occurs. Therefore, this example illustrates a website where only IE6 and IE8 can access exploit URLs and collect exploit code.

Conventional systems cannot detect this example for several reasons. A high-interaction honeyclient that uses an actual browser fails to collect exploit code and malware when the browser is not IE6 or IE8 due to browser fingerprinting. Similarly, a low-interaction honeyclient, i.e., a browser emulator, also fails to execute redirection code due to evasion code even if it emulates IE6 or IE8. Consequently, systems detecting malicious websites on the basis of URLs, redirections, and web content do not work effectively when these honeyclients cannot collect malicious data.

Our system can detect malicious websites that use evasion techniques by utilizing the redirection graphs of all

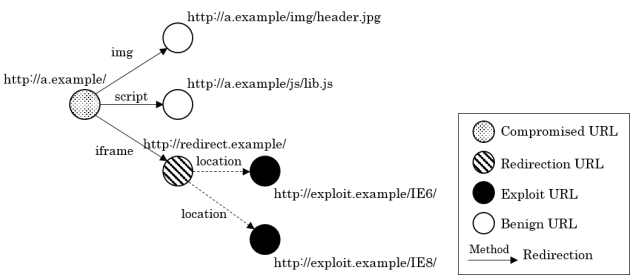

Fig. 1 Redirection graph of a motivating example.

```
1  <!DOCTYPE html>
2  <html>
3   <head>
4   <meta charset="UTF-8" />
5   <title>Sample Website</title>
6   <script src="js/lib.js" type="text/javascript"></script
        >
7   </head>
8   <body>
9   <img src="img/header.jpg" width="800px" alt="header" />
10  ... [snipped] ...
11  </body>
12 </html>
13 <iframe src="http://redirect.example/" style="position:
       absolute;width:0px;left:-99px;display:none;"></
       iframe>
```

Fig. 2 Iframe injection at `http://a.example/`.

```
1  var ua = "";
2  try{
3   new ActiveXObject("dummy");
4  }catch( e ){
5   ua = window.navigator.userAgent.toLowerCase();
6  }
7  if( ua.indexOf("msie 6") != -1 ){
8   location.href = "http://exploit.example/IE6/";
9  }
10 else if( ua.indexOf("msie 8") != -1 ){
11  location.href = "http://exploit.example/IE8/";
12 }
```

Fig. 3 Evasion and browser fingerprinting at `http://redirect.example/`.

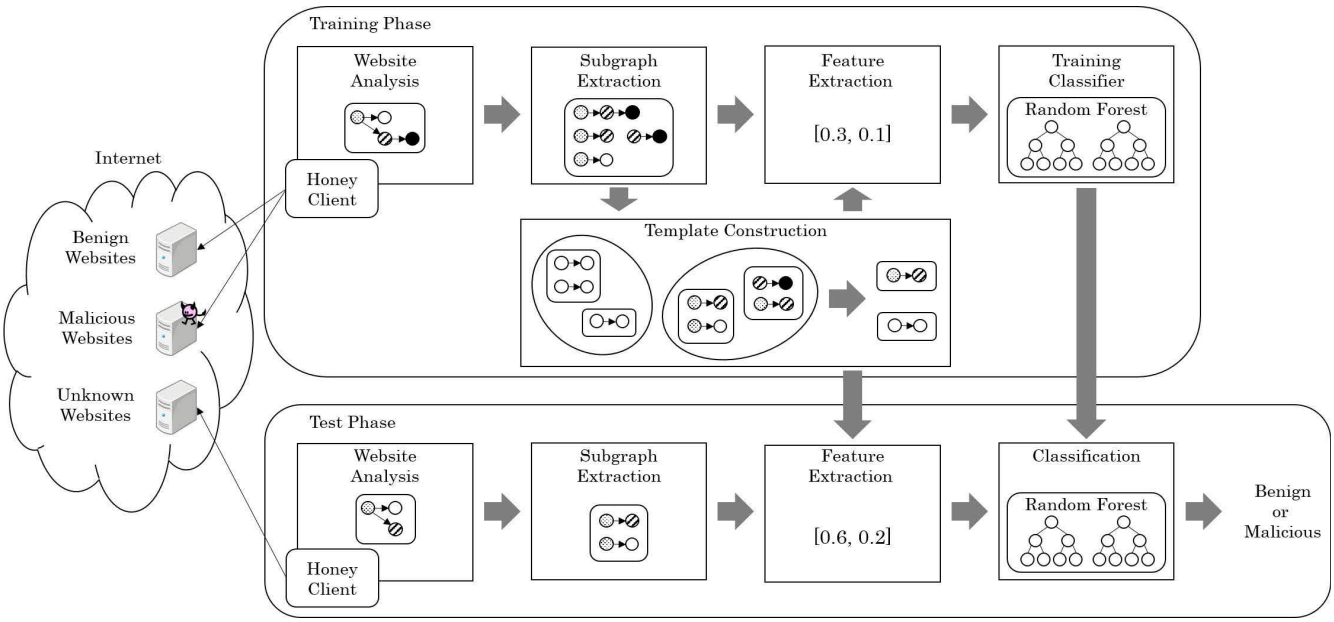

Fig. 4 System framework.

websites observed at the time of access without being limited to those of malicious websites created with exploit kits. In the above example, we can certainly observe the redirection of the invisible `iframe` tag and redirections to benign URLs to which the compromised website originally refers, i.e., `http://a.example/js/lib.js` specified by the `script` tag and `http://a.example/img/header.jpg` specified by the `img` tag in Fig. 1. In other words, we can detect malicious websites by building our classifier with features representing redirection subgraphs of easily compromised websites even if we fail to observe parts of malicious redirections and web content due to evasion techniques. As shown in Sect. 5, our system detected 143 more evasive malicious websites than the conventional content-based system.

3. Proposed System

We describe the design and implementation of our proposed system, which detects malicious websites on the basis of the redirection graphs of all websites.

3.1 System Design

Websites consist of multiple URLs and redirections between them. Their structure is represented as redirection graphs whose vertices are URLs and edges are redirections. To take advantage of their structure, we utilize a graph mining approach. One common approach is to perform classification by leveraging similarities of graphs. More precisely, subgraphs are extracted from each graph, and the similarities of many pairs of graphs are calculated on the basis of the number of subgraphs shared by the graphs. This approach leads to high classification accuracy but also has a drawback: a high computational cost. To achieve both high classification accuracy and low computational cost, we reduce the computational cost of subgraph extraction and the number of similarity calculations. In Sect. 3.2, we describe how to reduce the computational cost of subgraph extraction. Here, we discuss the number of similarity calculations. The classification is performed on the basis of similarities between test and training data. A large number of training data improves classification accuracy but results in a large number of similarity calculations, i.e., $O(NM)$, where N and M represent the number of training and test data, respectively. Our system constructs a comparatively small number of templates, which are subgraphs shared across training data, and performs classification on the basis of similarities between test data and templates. The number of similarity calculations is reduced to $O(M)$. Note that a graph mining approach also has another drawback: a large memory requirement. Because a large memory has become easier to obtain, we focus on only computational cost in this paper.

Figure 4 illustrates the framework of our system. In the training phase, we collect labeled redirection graphs (malicious or benign) with the honeyclient [14]. The redirection graphs are decomposed into their subgraphs. Then, templates are constructed from redirection subgraphs shared across them. Their feature vectors are extracted on the basis of the similarities between their subgraphs and templates. The classifier of random forest is trained with their feature vectors. In the test phase, we collect unlabeled redirection graphs with the honeyclient. Their feature vectors are extracted in the same manner as in the training phase and classified using the trained classifier. We explain details of our system in the next subsection.

Table 1 Features for calculating maliciousness of web content.

No.	Type	Feature
1	Redirection	HTTP 3xx redirections to different domain
2		Redirection to Flash file
3		Redirection to PDF file
4		Redirection to Java Applet File
5		Redirection without referer
6	HTML	Suspicious DOM position
7		Invisible content
8		Exploitable classid
9	JavaScript	# of Element object functions
10		# of String object functions
11		# of Node object functions
12		# of ActiveXObject functions
13		# of Document object functions
14		# of Navigator object functions
15		# of object-encoding functions
16		# of Time object functions
17		# of eval functions
18		# of fingerprinting functions
19		# of obfuscated content
20		# of content containing shellcode
21		# of long parameters
22		Entropy

3.2 Implementation

Subgraph Extraction. We collect web content at each URL and the methods used for redirections by analyzing websites with a honeyclient. The websites are represented as redirection graphs, i.e., directed graphs whose vertices are URLs and edges are redirections. The most important structure of redirection graphs for detecting malicious websites is the path from landing to exploit URLs. To reduce the computational cost, we extract only subgraphs that have an important structure, i.e., path-shaped subgraphs. Excluding subgraphs that have a branch structure reduces the computational cost. Let $G = (V, E)$ denote a redirection graph, where V is a set of vertices, and $E \subseteq (V \times V)$ is a set of edges. Edge (v_i, v_j) represents the redirections from vertex v_i to vertex v_j. A set of paths, P, is defined as $P = \{p_{i,j} | v_j \in c(v_i), v_i \in V\}$, where $p_{i,j}$ is a path from v_i to v_j, and $c(v_i)$ is a set of vertices reachable from v_i through edges. We use the information of vertices and edges as a feature of a subgraph. The feature of a subgraph sg is represented as (m, r), where m is a vector containing the information of vertices and r is a vector containing the information of redirections. A redirection graph is decomposed into a set of features of subgraphs extracted from all path-shaped subgraphs.

For the information of vertices, we calculate the maliciousness of web content because exact matching of URLs and their web content can be easily evaded by changing the characters of URLs and small pieces of their web content. The maliciousness is calculated with machine learning using the 22 widely used features in Table 1. These features are extracted with the honeyclient [14]. The features are divided into three types: redirection, HTML, and JavaScript. We extract five redirection features: HTTP 3xx redirections to different domains (No. 1), redirections to files used for exploit (Nos. 2–4), and redirection without referer (No. 5). We extract three HTML features: the suspicious Document Object Model (DOM) position (No. 6), invisible content (No. 7), and exploitable classid (No. 8). We extract 13 JavaScript features: the number of functions (Nos. 9–18), the number of suspicious content (Nos. 19–20), the number of long parameters (No. 21), and entropy (No. 22).

Table 2 Methods of redirections.

Type	Examples of methods		
HTTP 3xx	HTTP 301	HTTP 302	
HTML tag	`iframe`	`script`	`link`
JavaScript	`document.write`	`innerHTML`	

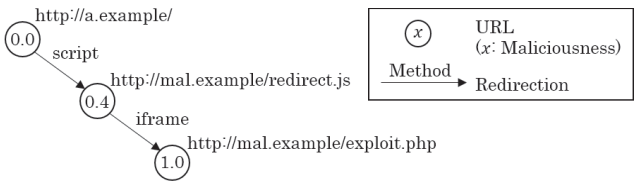

Fig. 5 Example of a subgraph.

Fingerprinting functions are identified by arguments including versions of plugins. Obfuscated content is identified by Latin-1 code or a comma delimited string whose length is 128 or more. Content containing shellcode is identified by a string including 128 or more Unicode/non-printable ASCII characters. We define long parameters as JavaScript functions' arguments whose lengths are 350 or more. To calculate maliciousness, we apply random forest [18] as a classifier. We train the classifier by using the training data in Sect. 4. For the information of edges, we use redirection methods and destination domains (external or internal). Table 2 shows the three types of redirection methods: HTTP 3xx, HTML tag, and JavaScript. HTML tag redirections are triggered by tags for acquiring external sources such as `iframe`, `script`, and `link`. JavaScript redirections are triggered by DOM modification functions such as `document.write` and `innerHTML`.

Figure 5 shows an example of a subgraph consisting of three vertices and two edges. The maliciousness of the first, second, and third vertices is 0.0, 0.4, and 1.0, respectively. The redirection methods of the first and second edges are `script` and `iframe`. The destination domains of the first and second edge are *external* and *internal*. The feature of a subgraph of this example sg is represented as (m, r), where $m = [0.0, 0.4, 1.0]$, $r = [$`script`$-$external, `iframe`$-$internal$]$. The information of edges is represented by the hyphenation of the redirection method and the destination domain. Hereafter, we attach new indexes to features of subgraphs and represent a subgraph feature set extracted from a redirection graph as $SG = \{sg_i\}$ for convenience.

Template Construction. We split redirection graphs into clusters composed of similar redirection graphs and construct templates from the clusters. The similarity utilized for clustering is defined similarly to the Dice index.

The Dice index, D, between set A and set B is defined as $D = 2\times|A\cap B|/(|A|+|B|)$. If the intersection of two subgraph feature sets is defined on the basis of both the redirection information and the maliciousness, the amount of difference in the maliciousness does not properly affect the similarity because the maliciousness is a continuous value. For this reason, we define the intersection of two subgraph feature sets on the basis of only the redirection information and use the maliciousness as weighting coefficients.

The similarity function $S(SG_i, SG_j)$, given subgraph feature sets SG_i and SG_j, is defined as

$$S(SG_i, SG_j) = \frac{2 \times \sum_{(m_k,m_l)\in\Lambda} s(m_k, m_l)}{|SG_i| + |SG_j|},$$
$$s(m_k, m_l) = \frac{1}{1 + \alpha \times |m_k - m_l|^2},$$
$$\Lambda = \{(m_k, m_l)|(m_k, r_k) \in SG_i, (m_l, r_k) \in SG_j\},$$

where α is a weight coefficient. Here, $s(m_k, m_l)$ denotes the similarity function given the maliciousness m_k and m_l. If there are multiple subgraph-feature-pair possibilities, we adopt the pair that has higher similarity than the others. The optimal pair can be quickly found by using the Hungarian algorithm [19].

This similarity is utilized for clustering. If the maximum similarity between redirection graphs belonging to one cluster and redirection graphs belonging to another is higher than threshold β, the two clusters are merged. This process is conducted from when each cluster is composed of one redirection graph to when no cluster can be merged.

Clustering is computationally expensive because similarities between all pairs of redirection graphs need to be calculated, i.e., $O(n^2)$, where n denotes the number of redirection graphs. We leverage locality sensitive hashing (LSH) [20] to avoid calculating the similarities of redirection graphs that have low similarities. Reducing the computational cost of clustering enables us to reduce computational resources or increase the number of candidate hyperparameters used for optimization. We encode a redirection graph into a vector by using bag-of-words representation to apply LSH. The vector contains the number of redirection methods or JavaScript functions/objects. The redirection methods include HTTP 302, `iframe` tag, and `script` tag. JavaScript functions/objects include `documentw.write`, `innerHTML`, `setInterval`, and `ActiveXObject`. The hash function is formulated as $h(x) = \lfloor \frac{a^T x+b}{c} \rfloor$, where a is a vector and b is a real number. Here, $\lfloor x \rfloor$ denotes the largest integer, which is equal to or less than x. Each element of a is chosen from the normal distribution whose mean is 0 and variance is σ^2. The real number, b, is chosen from the uniform distribution whose range is $[0, c]$. The parameters $\sigma^2 = 10, c = 1$ are selected so that the number of websites that have the same hash values is not too small.

We construct templates from the clusters composed of γ or more redirection graphs. The template, T, is a set of features of subgraphs whose redirection information is shared across all redirection graphs in the cluster $C = \{SG_i\}$. Since maliciousness is a continuous value, we use the average maliciousness as the maliciousness of the template. Template T is formulated as

$$T = \{(m_i, r_i)|\forall SG \in C, \exists(m_i, r_i) \in SG\},$$
$$m_i = \frac{1}{|C|} \sum_{m_j\in M_i} m_j,$$
$$M_i = \{m_j|(m_j, r_i) \in SG_j, SG_j \in C\}.$$

Feature Extraction. A high similarity between the features of subgraphs of a redirection graph and a template indicates that the redirection graph contains the template as its subgraph. In other words, we can encode the redirection graphs of websites into numerical values by calculating similarities between features of subgraphs of redirection graphs and templates. We extract a feature vector x containing similarities between a subgraph feature set SG and templates T_i: $x = [S(SG, T_1), \ldots, S(SG, T_N)]$, where N is the number of templates.

Classification. These feature vectors are classified by using supervised machine learning. We use random forest [18], which can classify a large amount of data accurately and quickly. We use a `randomForest` package in R [21]. Note that the classification algorithm is not limited to random forest; other algorithms of supervised machine learning can be applied.

4. Experimental Setup

Our proposed system is designed to detect not only malicious redirection graphs containing exploit URLs but also evasive malicious redirection graphs, which do *NOT* contain all malicious data. We evaluate the detection performance of our system using these redirection graphs. Here, we describe the experimental setup for evaluation.

4.1 Conventional Systems for Comparison

We evaluate the effectiveness of our system by comparing it with conventional systems. Some conventional systems detect malicious websites by matching redirection or exploit URLs [9], [12]. These systems suffer from false negatives when targeted URLs are concealed by evasion techniques. Other conventional systems using statistical features [6], [7], [10] are assumed to be more robust against evasion techniques because their targets are not limited to specific URLs. For this reason, we compare our system with conventional systems that use statistical features. Note that we cannot compare our system with conventional systems that leverage large-scale user traffic [22], [23] because our system is supposed to use web content and redirections collected with a honeyclient.

We compare our system with the content-based system, the redirection-based system, and the combination of these systems. The content-based system extracts widely used features listed in Table 1 such as the conventional system [6] and classifies them by using random forest. If one

Table 3 Features of the redirection-based system.

No.	Feature
1	# of different domains
2	Path length
3	# of HTTP 3xx redirections
4	# of different domain HTTP 3xx redirections
5	# of consecutive HTTP 3xx redirections
6	# of consecutive different domain HTTP 3xx redirections
7	# of consecutive short redirections
8	Median of redirection duration
9	Average of redirection duration
10	Minimum of redirection duration
11	Maximum of redirection duration

or more pieces of web content in a redirection graph are detected, the redirection graph is classified as malicious. If no piece of web content is detected, the redirection graph is classified as benign.

The redirection-based system extracts features from paths between landing URLs and final destinations of redirections and classifies them by using random forest. Table 3 shows a list of features that have been proposed for the conventional system [10]. More precisely, the features are the number of different domains (No. 1), path length (No. 2), HTTP 3xx redirections (Nos. 3–6), and redirection duration (Nos. 7–11). Short redirections are defined as redirections that occur in no more than one second. If one or more paths in a redirection graph are detected, the redirection graph is classified as malicious. If no path is detected, the redirection graph is classified as benign.

The combination of the content-based and redirection-based systems (combination system for short) classifies a redirection graph as malicious if it is detected by the content-based or redirection-based system. A redirection graph is classified as benign if it is detected by neither system.

4.2 Ground Truth

We collect the redirection graphs used for the evaluation by accessing websites listed on public blacklists [24], [25] or a list of popular websites [26] by using the low-interaction honeyclient [14]. Some websites listed on public blacklists no longer contain malicious web content, and websites listed on the popular-website list can be compromised and forced to engage in attacks. Since we need ground truth of redirection graphs, we access websites by using low-interaction and high-interaction honeyclients [27] almost at the same time. The high-interaction honeyclient detect exploit URLs on the basis of system behavior such as unintended process creation and file/registry accesses. We label redirection graphs detected by the high-interaction honeyclient as malicious and redirection graphs listed on the popular-website list and not detected as benign. We do not use the redirection graphs listed on public blacklists but not detected because they might not be detected due to the discrepancy between the targeted environment of exploit and the environment of the high-interaction honeyclient. Since redirection graphs are all websites observed at the time of access, malicious redirection graphs contained compromised and malicious websites. Note that the malicious redirection graphs do not necessarily contain exploit URLs due to evasion techniques even if the high-interaction honeyclient access exploit URLs. We use malicious redirection graphs that do not contain exploit URLs as evasive malicious redirection graphs.

Table 4 Dataset.

	Label	Number	Period
Training	Malicious	2,170	Jan.–Apr. 2016
	Benign	199,982	Aug. 2016
1st Test	Malicious	365	May–Sep. 2016
	Benign	249,958	Aug. 2016
2nd Test	Evasive	3,385	Jan.–Sep. 2016

Conventional systems require labeled web content or paths. For the content-based system, we label web content as malicious if it is collected from destinations of malicious redirections and domains of corresponding URLs are different from those of landing URLs. We label the web content of benign redirection graphs as benign. For the redirection-based system, we label paths including exploit URLs as malicious and paths of benign websites as benign.

4.3 Dataset

We use one training dataset and two test datasets for our evaluation as shown in Table 4. The training dataset consists of 2,170 malicious redirection graphs collected from Jan.–Apr. 2016 and 199,982 benign redirection graphs collected in Aug. 2016. We confirm that each malicious redirection graph in the training dataset contains at least one malicious redirection. Note that we can collect malicious redirection graphs because their targeted environment and the environment of the low-interaction honeyclient are identical or evasion techniques are not used. The benign training dataset should be collected during the same period as the malicious dataset. However, we have not collected benign redirection graphs during that period; hence, we use benign redirection graphs collected in Aug. 2016 as a training dataset. Note that the collection period is not assumed to affect the evaluation results because the redirection graphs of benign websites are not subject to change.

We use the first test dataset to evaluate the detection performance with malicious redirection graphs containing exploit URLs from a large number of benign redirection graphs. It consists of 365 malicious redirection graphs collected from May–Sep. 2016 and 249,958 benign redirection graphs collected in Aug. 2016. We use the second test dataset to evaluate the detection performance with evasive malicious redirection graphs. It consists of 3,385 evasive malicious redirection graphs collected from Jan.–Sep. 2016. This dataset does not overlap with the training data collected in the period Jan.–Apr. 2016.

4.4 Hyperparameter Optimization

The hyperparameters are a weight coefficient, α, of similarity, threshold β for clustering, and threshold γ for template construction. In addition to them, we optimize the training dataset. The prevalence of new exploit kits or updates to exploit kits changes the redirection subgraphs of malicious websites. Therefore, malicious redirection graphs detected in the past do not always contribute to improving the classification performance. We optimize the percentage θ of malicious redirection graphs that we use for training. The number of malicious redirection graphs is limited, but we can collect a large number of benign redirection graphs. We optimize the ratio η of malicious to benign redirection graphs.

We further split the training dataset into a prior-training dataset and a validation dataset. The prior-training dataset includes 90% of the training dataset selected from the oldest data and is used for training a classifier The validation dataset includes 10% of the training dataset selected from the newest data and is used for evaluating the classification performance. We select the hyperparameters that had the highest classification performance. To evaluate the performance, we use the f-measure defined as:

$$\text{f-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}},$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

where TP, FN, TN, and FP denote the number of true positives, false negatives, true negatives, and false positives, respectively. The best hyperparameters were $\alpha = 1, \beta = 0.4$, $\gamma = 2$, $\theta = 50$, and $\eta = 1 : 10$. The hyperparameters of random forest, i.e., the number of decision trees and the number of features for each decision tree, are optimized by using the `tuneRF` function of the `randomForest` package in R [21] when a classifier is trained.

If a small difference in the hyperparameters of our system results in a large difference in its classification performance, it makes our system difficult to deploy because the hyperparameters need to be carefully optimized. To determine whether they are difficult or not to optimize, we investigate classification performance with different system-specific hyperparameters, i.e., α, β, and γ. We select α from 1, 10, and 100, β from 0.4, 0.6, and 0.8, and γ from 2, 5, and 10. The classifier is trained by using the prior-training dataset and calculate the f-measure on the validation dataset. Figure 6 shows the f-measure of every hyperparameter. The hyperparameters are arranged in the ascending order of f-measures. The hyperparameters and their orders are listed in Table 5. Since our system achieved high f-measures with several hyperparameters, we can find with simple optimization methods such as a grid search.

Our system leverages LSH to reduce the computational cost of clustering. We evaluate the effectiveness of LSH

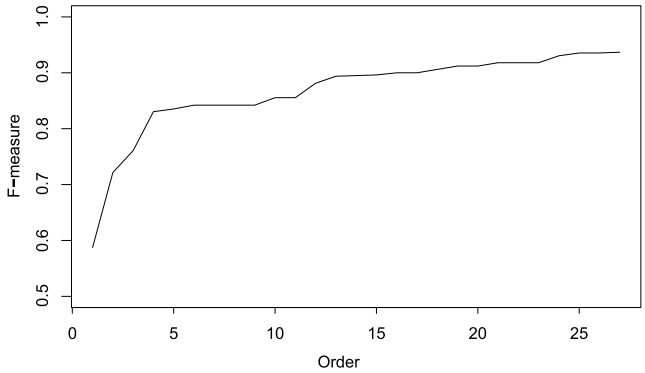

Fig. 6 F-measure of different hyperparameters.

Table 5 Hyperparametes and their orders.

Order	α	β	γ	Order	α	β	γ
1	1	0.4	10	15	1	0.8	2
2	10	0.8	10	16	10	0.6	2
3	100	0.8	10	17	10	0.6	5
4	1	0.8	10	18	1	0.6	5
5	10	0.8	5	19	1	0.4	5
6	100	0.4	5	20	10	0.4	2
7	100	0.6	5	21	10	0.4	5
8	100	0.6	10	22	10	0.8	2
9	100	0.8	5	23	100	0.8	2
10	10	0.6	10	24	1	0.6	2
11	100	0.4	10	25	100	0.4	2
12	10	0.4	10	26	100	0.6	2
13	1	0.8	5	27	1	0.4	2
14	1	0.6	10				

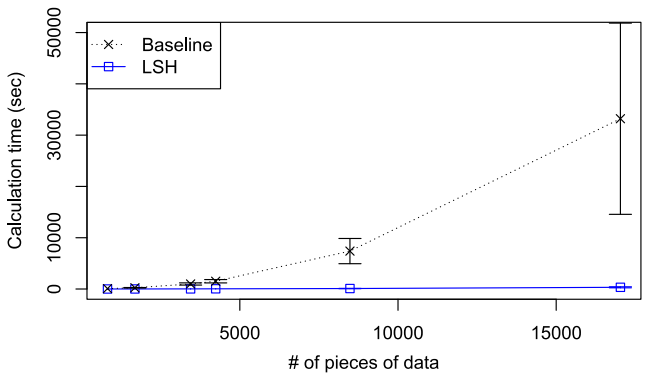

Fig. 7 Calculation time of clustering. Error bars represent standard deviation.

by comparing the calculation times with clustering without LSH (baseline). Figure 7 shows the average and standard deviation of calculation time. If the number of data is the same, the calculation time differed depending on the hyperparameters. This result shows that LSH drastically speeds up clustering. Note that we selected hyperparameters of LSH so that similar redirection graphs have the same hash value with high probability. Such hyperparameters cannot achieve optimal clustering speed but suppress degradation of classification performance caused by inaccurate clustering results.

5. Experimental Results

We now report the experimental results. In Sect. 5.1, we show the results of the first test dataset (see Sect. 4.3 for more detail) including malicious redirection graphs containing exploit URLs. We compare our system with conventional systems in terms of classification performance, degradation in classification performance over time, and calculation time. We further investigate constructed templates, false positives, and false negatives to obtain a better understanding of our system. Lastly, we present a case study of server-side cloaking to show the effectiveness of our system. In Sect. 5.2, we show the results of the second dataset including evasive malicious redirection graphs. To avoid duplicate evaluation, we focus on the detection capability of evasive malicious redirection graphs in this subsection. Specifically, we show classification performance, false negatives, and a case study of an evasive malicious website. The prototype version of our system is installed on an Ubuntu server with four 12-core CPUs and 128-GB RAM.

5.1 Detecting Malicious Websites with Exploit URLs

We report the evaluation results of the first test dataset including malicious redirection graphs containing exploit URLs in terms of classification performance, degradation in classification performance over time, and calculation time. We also show analysis results of constructed templates, false positives, false negatives, and a redirection graph of server-side cloaking.

Classification Performance. We evaluate our system by using widely used metrics: TPR (also known as recall), FPR, precision, f-measure, area under the receiver operating characteristic curve (AUC), and TPR at a fixed FPR of 0.1%. As shown in Table 6, our system outperformed all conventional systems for all metrics. These results show that leveraging the redirection graphs of all websites contributes to improving the classification performance.

Classification Performance Degradation Over Time. The prevalence of new exploit kits and updates to exploit kits degrades the classification performance. Therefore, we evaluate the performance degradation over time. Figure 9 shows the TPR of malicious redirection graphs collected for the first, second, and third or following month of the test dataset. The TPRs of our system and the redirection-based system become smaller for the second month of the test dataset than for the first month. The TPRs of the content-based and combination systems become smaller for the third or following months of the test dataset than for the first and second months. The degradation of the combination system was similar to that of the content-based system because the content-based system had a dominant role in the classification of the combination system. The degradation of our system was steeper than those of conventional systems because our system utilizes more types of websites. That is, our system is focused on malicious, benign, and compromised websites, and a change in structure of any one of them degrades classification performance. Specifically, classification performance of our system degraded because features of compromised websites changed between the first and second months of the test dataset. The training data includes a large number of compromised websites constructed by using WordPress, which is a CMS. Similarly, the number of compromised websites constructed with WordPress in the test dataset is high in the first month but low in the second month. The number of such compromised websites in the third or following months of the test dataset was higher than that in the second month. This is why the proposed system had a higher TPR than the conventional systems in the third or following months of the test dataset.

Calculation Time. To detect malicious websites, we must analyze a large number of websites. To evaluate the capability of large-scale analysis, we evaluate the calculation time of feature extraction and classification. Table 8 shows that our system required a 35 times longer calculation time than the content-based system. However, our system finished the feature extraction and classification of one redirection graph in less than 0.13 seconds. The calculation time of feature extraction and classification was shorter than that of the download and execution of web content. Therefore, our system can classify a large number of redirection graphs collected by using a honeyclient.

Template Analysis. We analyze the templates of our system to elucidate the improvement in classification performance. Table 7 shows the number of templates and order of the *importance* calculated with the `randomForest` package for each label of template. The labels of templates are those of redirection graphs from which templates are constructed: malicious, benign, and malicious and benign (compromised for short). The *importance* represents the level of contri-

Table 6 Classification performance with malicious redirection graphs containing exploit URLs.

System	Proposed	Content	Redirect	Comb.
TPR (recall)	**0.9057**	0.8465	0.2564	0.8876
FPR	**0.0007**	0.0022	0.0028	0.0041
Precision	**0.6631**	0.0385	0.1388	0.2401
F-measure	**0.7657**	0.5300	0.1801	0.3780
AUC	**0.9938**	0.9664	0.6408	0.9774
TPR (FPR=0.1%)	**0.9171**	0.7726	0.2256	0.7726

Table 7 Statistics of templates

		Malicious	Benign	Compromised
# of templates		167	246	54
Order of importance	Highest	1	10	7
	Lowest	454	467	458
	Average	111.7	293.1	343.0

Table 8 Calculation time for one website (sec).

System	Proposed	Content	Redirection	Comb.
Feature extraction	0.1251	**0.0033**	0.0073	0.0106
Classification	0.0005	**0.0003**	0.0002	0.0004

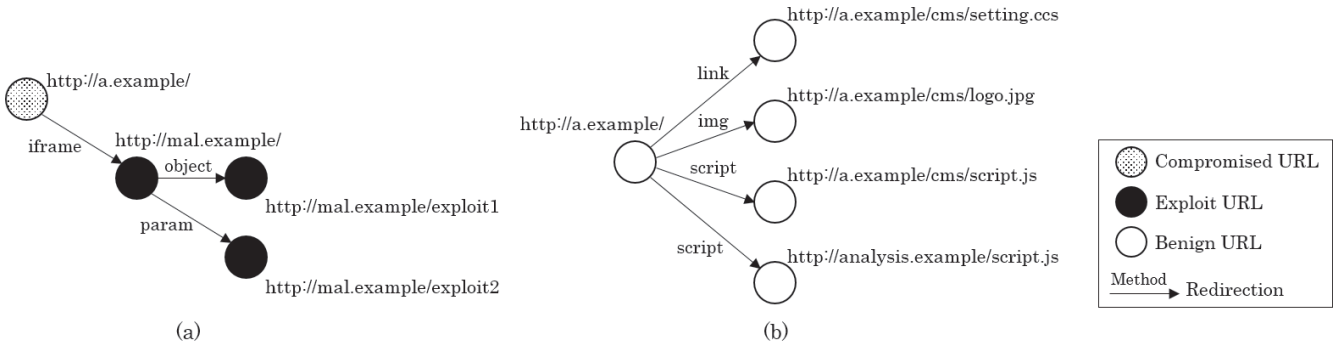

Fig. 8 Redirection subgraphs of templates.

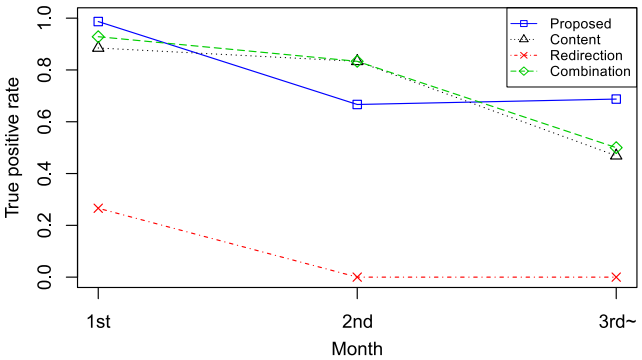

Fig. 9 True positive rate degradation over time.

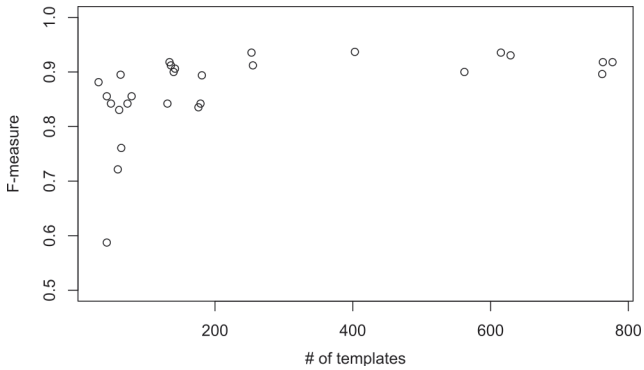

Fig. 10 F-measure and the number of templates on different hyperparameters.

bution of each template to classification. More than half the templates were constructed from only benign websites. This is because benign redirection graphs outnumbered malicious ones. However, malicious templates tended to have a higher *importance* than benign and compromised templates. Some benign and compromised templates also had the highest *importance*.

One malicious template that had a high *importance* contained redirection subgraphs relevant to the Angler Exploit Kit as shown in Fig. 8 (a). The `iframe` tag redirecting to the exploit URL of a different domain was injected at the landing URL. The malicious web content at `http://mal.example/` exploited a use-after-free vulnerability (CVE-2014-4130) and also contained redirection to other malicious web content such as Flash (CVE-2015-0313). Another malicious template that had a high *importance* contained redirection subgraphs relevant to an exploit kit and compromised website. Since it contained all websites related to drive-by download attacks, it was a large template composed of many redirection subgraphs.

One compromised template that had a high *importance* contained redirection subgraphs relevant to a CMS as shown in Fig. 8 (b). Websites created with a CMS tend to be targeted and compromised by attackers. For this reason, redirection subgraphs relevant to the same CMS were included in malicious and benign redirection graphs. The landing URL includes some HTML tags redirecting to Cascading Style Sheets, image, and JavaScript code. These redirections are included in the template of the CMS. The landing URL also includes a `script` tag redirecting to an analysis service because many websites' administrators use it.

One benign template that had a high *importance* contained redirection subgraphs relevant to an advertisement. If websites use the same advertisement service, they have the same redirection subgraph for obtaining advertisement content. This is why a template relevant to the advertisement was constructed. We omit illustrating the template because it contains too many URLs to be depicted in a limited amount of space.

By using these templates, our system can classify redirection graphs on the basis of the structural similarities to exploit kits, CMSs, and advertisements. If the content-based system wrongly classified advertisement content as malicious, our system classified its redirection graphs as benign by referring to other web content and redirections relevant to the advertisement. In addition, if the content-based system could not detect malicious web content, our system detected its redirection graphs by taking the compromised CMSs into account.

As shown in the aforementioned examples, effective templates are essential to achieve high classification performance. We investigate classification performance on the basis of different templates. Specifically, we investigate the number of templates, distribution of template sizes, and f-measure with different hyperparameters. We use the number of redirection subgraphs consisting of a template as the size of a template. When the number of templates was small, our system had a low f-measure as shown in Fig. 10. This is because coverage of redirection subgraphs was not sufficient

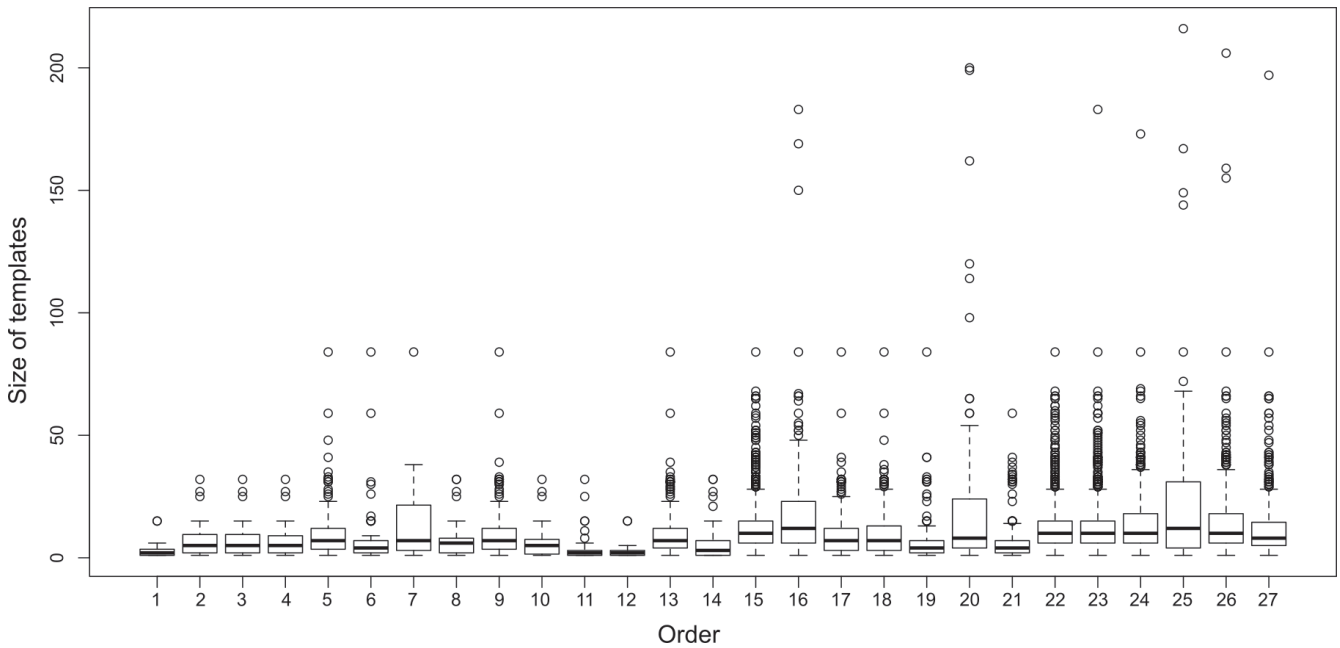

Fig. 11 Distribution of size of templates on different hyperparameters.

to classify redirection graphs accurately. Figure 11 shows the distribution of template sizes with different hyperparameters. The hyperparameters are arranged in the ascending order of f-measures. When the f-measure was high, large templates tended to be constructed and variance of the distribution tended to be high. When the f-measure was low, most templates were small. This is because small templates were similar to many redirection graphs and unable to extract discriminative features. To achieve a high classification performance, various differently sized templates need to be constructed.

False Positives and False Negatives. We identified two main cases of false positives with manual inspection. The first was websites created with a CMS for electronic commerce sites. They contained multistage redirections to JavaScript code. Similarly, malicious redirection graphs have multistage redirections to malicious JavaScript code. This structural similarity caused the false positives. The second case was websites created with a CMS and slightly modified by their administrators. Some redirections for advertisement or analysis services were injected to landing URLs. The redirection graphs of these websites were similar to malicious redirection graphs created by compromising CMSs. Note that our system accurately classified redirection graphs of plain or customized CMS websites as benign.

We identified one main case of false negatives. On the websites of false negatives, benign JavaScript code that was the destination of redirections from landing URLs was compromised. However, in most malicious redirection graphs, the web content of landing URLs was compromised. The difference in redirection graphs caused false negatives.

Case Study of Server-side Cloaking. We describe a website for which redirection subgraphs of compromised websites need to be captured for detection. The redirection

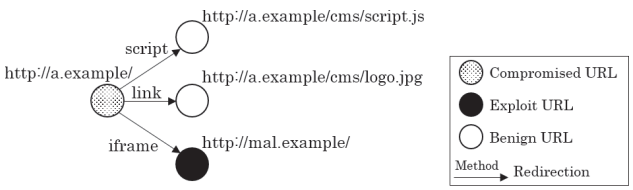

Fig. 12 Redirection graph of a website launching server-side cloaking.

from the landing URL to the exploit URL was triggered by an injected `iframe` tag as shown in Fig. 12. The status code of the HTTP response from the exploit URL was 200, but its body was empty. This website was speculated to be an attempt of server-side cloaking, which detects security vendors or researchers on web servers and conceals malicious web content from them. The exploit URL was created with Rig Exploit Kit, and most attempts to obtain malicious web content were unsuccessful. The content-based system could definitely not detect this website due to the lack of malicious web content.

Our system detected the website by utilizing the redirection graph of all of the websites. The compromised website was created with a CMS that is sometimes compromised by attackers. Moreover, it had a redirection to a different domain with a `iframe` tag. The same redirection is frequently used on malicious redirection graphs. Our system detected this website by capturing both traits of the compromised website and the malicious redirection.

5.2 Detecting Evasive Malicious Websites

We report the evaluation results of the second dataset including evasive malicious redirection graphs. To avoid duplicate evaluation, we focus on the detection capability and show

Table 9 Number of TP and TPR at fixed FPR of 0.1%.

System	Proposed	Content	Redirection	Comb.
# of TP	**821**	678	71	669
TPR	**0.243**	0.200	0.021	0.206

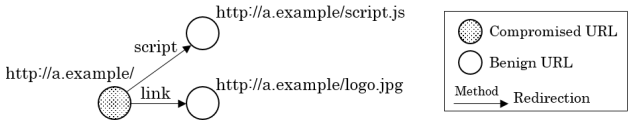

Fig. 13 Redirection graph of a false negative.

Fig. 14 Redirection graph of an evasive malicious website.

classification performance, false negatives, and a case study of an evasive malicious website.

Classification Performance. Table 9 shows the number of TP and TPR at a fixed FPR of 0.1%. FPR was fixed using the test dataset of malicious redirection graphs containing exploit URLs. Our system detected 143 more evasive malicious redirection graphs than the content-based system. On the evasive malicious websites detected by our system, the evasion code prevented the low-interaction honeyclient from accessing exploit URLs at redirection URLs. The redirection graph and evasion code were more precisely illustrated as a case study. The content-based system could not detect some malicious web content at redirection URLs. This is why our system detected more evasive malicious redirection graphs.

False Negatives. We identified one main case of false negatives as shown in Fig. 13. On these websites, the evasion code was used at landing URLs, i.e., `http://a.example/` in Fig. 13. Therefore, the low-interaction honeyclient was not redirected to malicious URLs but only to benign URLs, i.e., `http://a.example/script.js` and `http://a.example/logo.jpg` in Fig. 13. The redirection graphs of these websites were the same as those of benign websites. On benign websites, a client is typically redirected to some benign URLs to obtain scripts or images when it accesses a landing URL. As a result, benign redirection graphs are shallow and have many branches. For this reason, our system could not detect websites where the evasion code was used at landing URLs. Note that we did not find any websites containing malicious redirections after manually inspecting 100 false negatives.

Case Study of Evasive Malicious Website. We describe the evasive malicious website shown in Fig. 14. This website (redirection URL) was pointed to by the `iframe` tag injected at the landing URL. Figure 15 shows the evasion code created with the Angler Exploit Kit. A different value was assigned to `flag` depending on the error when the

```
1 var flag;
2 try {
3  var a = new ActiveXObject("Anti-Virus");
4  flag = true;
5 }catch( e ){
6  flag = false;
7 }
```

Fig. 15 Evasion code at `http://mal.example/redirect.js`.

ActiveXObject of "Anti-virus" was loaded at line 3. If the error occurred, `true` was assigned, and vice versa at lines 4 and 6. Subsequent malicious code was executed only if it was `true`. Since the high-interaction honeyclient had no anti-virus software installed, it raised the exception. However, the low-interaction honeyclient did not raise any exception. By leveraging this difference, the evasive malicious website prevented the low-interaction honeyclient from accessing the exploit URL.

The compromised website was also created with a CMS sometimes compromised by attackers. Similar to the website launching server-side cloaking, our system could detect it by utilizing the redirection graph of all of the websites.

6. Limitations

Our system requires redirection subgraphs widely shared across malicious websites to distinguish malicious redirection graphs from benign ones. Therefore, our system did not detect malicious redirection graphs that had subgraphs different from those of typical malicious websites (as discussed in Sect. 5.1) or malicious redirection graphs that contain evasion code used at landing URLs (as discussed in Sect. 5.2). This is a general limitation from which all machine-learning-based systems suffer. To detect uncommon malicious redirection graphs, the number of malicious training data must be increased. To detect malicious redirection graphs that contain evasion code used at landing URLs, systems that detect injected code on compromised websites [3], [4] can be utilized complementarily. We can analyze a relatively small number of websites detected by these systems with various clients and detect malicious websites using collected malicious redirection graphs by using our system.

Another limitation is the degradation in classification performance over time. Conventional systems also have this limitation, but the degradation of our system was steeper than those of the conventional systems as discussed in Sect. 5.1. Our system is based on redirection subgraphs of many websites such as benign, compromised, and malicious websites. Change in the structure of any one of them degrades the classification performance of our system. The advantage of our system is to achieve high classification performance if training and test data are similar. The evaluation results showed our system achieved the highest classification performance on the first month of test data. To maintain high classification performance, the classifier should be retrained every month by using data labeled by high-

interaction honeyclients. High-interaction honeyclients are not suitable for large-scale analysis due to their slow processing speed but are useful for labeling a limited number of websites.

7. Related Work

7.1 Malicious Website Detection

Many systems have been proposed from different perspectives for detecting malicious websites launching drive-by download attacks. Here, we describe conventional systems focused on large-scale user traffic, system behavior, and web content and redirections.

Large-scale User Traffic. One approach for detecting malicious websites is aggregating large-scale user traffic [22], [23]. Attackers redirect clients to the same redirection URL from landing URLs and then redirect them to the exploit URLs targeting their environment. Geographical diversity and uniform client environments can be used as traits of malicious websites. However, these systems require logs provided by anti-virus vendors or large ISPs, and these logs are generally difficult to obtain. From the perspective of deployment, we designed our system to use data collected with a honeyclient.

System Behavior. Decoy client systems using actual browsers have been proposed to detect exploit accurately by monitoring unintended process creation and file/registry accesses [27]–[29]. These systems are known as high-interaction honeyclients. They have the limitations of a slow processing speed due to the use of actual browsers and the limited coverage of collected malicious data due to browser fingerprinting as discussed in Sects. 1 and 2. For this reason, they are not suitable for large-scale analysis.

Web Content and Redirections. Systems in this category are designed for large-scale analysis. They collect web content or redirections by using browser emulators and classify them by using machine learning. Browser emulators developed for light-weight collection are known as low-interaction honeyclients. Malicious web content has distinctive features to exploit known CVE-ID (Common Vulnerabilities and Exposures identification) or trigger malicious redirections. Some systems are focused on JavaScript code and HTML tags [6]–[8]. Other systems are focused on multistage redirections such as the difference in domains and duration of redirections [10], URLs shared across malicious websites [9], sequences of URLs [30], and URLs/HTTP headers and redirections between them [12]. These systems are focused on malicious URLs, web content, or redirections, but our system is focused on the redirection graphs of all websites, i.e., malicious/benign web content and malicious/benign redirections.

7.2 Compromised Website Detection

Detecting compromised websites is another approach to preventing damage caused by drive-by download attacks. Soska and Christin [3] detect websites that will become malicious in the future by focusing on web content that is not generated by users, such as vulnerable CMSs. Li et al. [4] detect compromised JavaScript code that triggers malicious redirections by comparing it with its clean counterpart. These systems require a clean version of compromised websites or JavaScript code. Our system detects compromised websites by leveraging only already compromised websites and totally benign websites.

7.3 Classification of Graphs

Many researchers have studied different approaches for classifying graph-structured data such as protein or medicine.

Graph Kernels. One approach is to design a graph kernel that is a function to calculate similarities between graphs and classify graphs with machine learning algorithms based on kernel methods such as support vector machine [31]. The random walk kernel [32] is based on the co-occurrence of sequences of labels on vertices or edges in random walks. The shortest-path kernel [33] is based on the co-occurrence of the length of shortest-paths between pairs of vertices. The graphlet kernel [34] is based on the co-occurrence of subgraphs that have k vertices. The Weishfeiler-Lehman graph kernel [35] is based on the co-occurrence of multilabels that are created by iteratively integrating a vertex's and its neighbors' labels. The deep graph kernel [36] is designed to extend the above graph kernels by leveraging latent representations of sub-structures. The deep graph kernel is defined by considering similarities between sub-structures as well as co-occurrence of sub-structures. These methods effectively calculate similarities between graphs on the basis of their sub-structures, but counting the occurrence of a large number of sub-structures is computationally expensive. Our system reduces computational cost by calculating similarities between a redirection graph and a small number of templates.

Convolution on Graphs. Since deep neural networks (DNNs) have achieved outstanding classification performance in image recognition and natural language processing, some studies have applied DNNs to classification of graphs. Duvenaud et al. [37] have proposed a convolutional neural network (CNN) that iteratively convolutes vectors representing a vertex and its neighbors and calculates the summation of convoluted vectors as a representation of a whole graph. Li et al. [38] conduct similar convolutions with the gated recurrent units. Dai et al. [39] designed an architecture of neural networks on the basis of graphical model inference procedures. Niepert et al. [40] apply conventional CNNs by arranging vertices in the order of certain criteria such as centrality. These methods have difficulty extracting features of differently sized redirection subgraphs such as exploit kits, compromised websites, and advertisements because the number of iterations of convolutions must be determined before training. Our system extracts features of differently sized redirection subgraphs by leveraging differently sized templates.

8. Conclusion

We have proposed a system for detecting malicious websites on the basis of the redirection graphs of all websites even when some malicious web content is concealed. We have extracted redirection subgraphs shared across malicious, benign, and compromised websites as templates and classified websites using feature vectors containing similarities between their features of subgraphs and the templates. We have found that templates contained redirection subgraphs of exploit kits, compromised websites, and advertiser websites. These templates enable our system to identify malicious websites by capturing redirection subgraphs of compromised websites as well as those of malicious websites. As a result of evaluating our system with crawling data of 455,860 websites, we have found that it achieved a 91.7% true positive rate for malicious websites containing exploit URLs at a low false positive rate of 0.1%. Moreover, our system have detected 143 more evasive malicious websites than the conventional content-based system.

References

[1] T. Shibahara, Y. Takata, M. Akiyama, T. Yagi, and T. Yada, "Detecting malicious websites by integrating malicious, benign, and compromised redirection subgraph similarities," Proceedings of the 41st Annual IEEE Computer Software and Applications Conference, pp.655–664, 2017.

[2] N. Provos, P. Mavrommatis, M.A. Rajab, and F. Monrose, "All your iframes point to us," Proceedings of the 17th USENIX Security Symposium, pp.1–16, 2008.

[3] K. Soska and N. Christin, "Automatically detecting vulnerable websites before they turn malicious," Proceedings of the 23rd USENIX Security Symposium, pp.625–640, 2014.

[4] Z. Li, S. Alrwais, X. Wang, and E. Alowaisheq, "Hunting the red fox online: Understanding and detection of mass redirect-script injections," Proceedings of the 35th IEEE Symposium on Security and Privacy, pp.3–18, 2014.

[5] C. Grier, L. Ballard, J. Caballero, N. Chachra, C.J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M.Z. Rafique, M.A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, G.M. Voelker, "Manufacturing compromise: the emergence of exploit-as-a-service," Proceedings of the 19th ACM Conference on Computer and Communications Security, pp.821–832, 2012.

[6] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: a fast filter for the large-scale detection of malicious web pages," Proceedings of the 20th International Conference on World Wide Web, pp.197–206, 2011.

[7] C. Curtsinger, B. Livshits, B.G. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection.," Proceedings of the 20th USENIX Security Symposium, pp.33–48, 2011.

[8] B. Stock, B. Livshits, and B. Zorn, "Kizzle: A signature compiler for detecting exploit kits," Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp.455–466, 2016.

[9] J. Zhang, C. Seifert, J.W. Stokes, and W. Lee, "Arrow: Generating signatures to detect drive-by downloads," Proceedings of the 20th International Conference on World Wide Web, pp.187–196, 2011.

[10] H. Mekky, R. Torres, Z.-L. Zhang, S. Saha, and A. Nucci, "Detecting malicious http redirections using trees of user browsing activity," Proceedings of the 33rd Annual IEEE International Conference on Computer Communications, pp.1159–1167, 2014.

[11] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, "Webwitness: investigating, categorizing, and mitigating malware download paths," Proceedings of the 24th USENIX Conference on Security Symposium, pp.1025–1040, 2015.

[12] T. Taylor, X. Hu, T. Wang, J. Jang, M.P. Stoecklin, F. Monrose, and R. Sailer, "Detecting malicious exploit kits using tree-based similarity searches," Proceedings of the 6th ACM Conference on Data and Application Security and Privacy, pp.255–266, 2016.

[13] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," Proceedings of the 19th International Conference on World Wide Web, pp.281–290, 2010.

[14] Y. Takata, M. Akiyama, T. Yagi, T. Yada, and S. Goto, "Website forensic investigtion to identify evidence and impact of compromise," Proceedings of the 12th International Conference on Security and Privacy in Communication Networks, vol.198, pp.431–453, 2017.

[15] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware.," Proceedings of the 22nd USENIX Security Symposium, pp.637–652, 2013.

[16] K. Kim, I.L. Kim, C.H. Kim, Y. Kwon, Y. Zheng, X. Zhang, and D. Xu, "J-force: Forced execution on javascript," Proceedings of the 26th International Conference on World Wide Web, pp.897–906, 2017.

[17] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: Decloaking internet malware," Proceedings of the 33rd IEEE Symposium on Security and Privacy, pp.443–457, 2012.

[18] L. Breiman, "Random forests," Machine learning, vol.45, no.1, pp.5–32, 2001.

[19] J. Munkres, "Algorithms for the assignment and transportation problems," Journal of the Society for Industrial and Applied Mathematics, vol.5, no.1, pp.32–38, 1957.

[20] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," Proceedings of the 20th Annual Symposium on Computational Geometry, pp.253–262, 2004.

[21] R Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[22] G. Stringhini, C. Kruegel, and G. Vigna, "Shady paths: Leveraging surfing crowds to detect malicious web pages," Proceedings of the 2013 ACM Conference on Computer and Communications Security, pp.133–144, 2013.

[23] L. Invernizzi, S. Miskovic, R. Torres, C. Kruegel, S. Saha, G. Vigna, S.-J. Lee, and M. Mellia, "Nazca: Detecting malware distribution in large-scale networks," Proceedings of the 21st Network and Distributed System Security Symposium, 2014.

[24] Malwarebytes, "hphosts." http://www.hosts-file.net/

[25] "Malware domain list." http://www.malwaredomainlist.com/

[26] Alexa Internet, Inc., "The top 500 sites on the web." http://www.alexa.com/topsites

[27] 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 on Communications, vol.E93-B, no.5, pp.1131–1139, 2010.

[28] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, "Blade: an attack-agnostic approach for preventing drive-by malware infections," Proceedings of the 17th ACM Conference on Computer and Communications Security, pp.440–450, 2010.

[29] Honeynet Project, "Capture-hpc." https://projects.honeynet.org/capture-hpc

[30] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: understanding and detecting malicious web advertising," Proceedings of the 19th ACM Conference on Computer and Communications Security, pp.674–686, 2012.

[31] J.A.K. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," Neural processing letters, vol.9, no.3, pp.293–300, 1999.
[32] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," Learning Theory and Kernel Machines, vol.2777, pp.129–143, 2003.
[33] K.M. Borgwardt and H.P. Kriegel, "Shortest-path kernels on graphs," Proceedings of the 5th IEEE International Conference on Data Mining, pp.74–81, 2005.
[34] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, pp.488–495, 2009.
[35] N. Shervashidze, P. Schweitzer, E.J.V. Leeuwen, K. Mehlhorn, and K.M. Borgwardt, "Weisfeiler-lehman graph kernels," Journal of Machine Learning Research, vol.12, pp.2539–2561, 2011.
[36] P. Yanardag and S. Vishwanathan, "Deep graph kernels," Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.1365–1374, 2015.
[37] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R.P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in Advances in Neural Information Processing Systems 28, pp.2224–2232, 2015.
[38] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," arXiv preprint arXiv:1511.05493, 2015.
[39] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," Proceedings of the 33rd International Conference on Machine Learning, pp.2702–2711, 2016.
[40] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," Proceedings of the 33rd International conference on machine learning, pp.2014–2023, 2016.

Toshiki Shibahara is currently a researcher at NTT Secure Platform Laboratories, Tokyo, Japan. He is also a Ph.D. student at Osaka University. He received his B.E. degree in engineering and M.E degree in information science and technology from The University of Tokyo, Japan in 2012 and 2014. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2014, he has been engaged in research on cyber security and machine learning.

Yuta Takata received his B.E., M.E., and Ph.D. degrees in computer science and engineering from Waseda University, Japan in 2011, 2013, and 2018. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2013, he has been engaged in research and development on cyber security using honeypot/honeyclient and program/content analysis techniques.

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

Takeshi Yagi received his B.E. degree in electrical and electronic engineering and his M.E. degree in science and technology from Chiba University, Japan in 2000 and 2002. He also received his Ph.D. degree in information science and technology from Osaka University, Osaka, Japan in 2013. He joined the Nippon Telegraph and Telephone Corporation (NTT) in 2002, where he was previously engaged in research and design on network architecture and traffic engineering and is currently researching network security, web security, honeypots, security-data analysis based on machine learning, 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).

Kunio Hato received his B.E. and M.E. degrees in information processing from Tokyo Institute of Technology in 1997 and 1999, respectively. He joined the Nippon Telegraph and Telephone Corporation (NTT) in 1999, where he was previously engaged in research and development of IP VPNs, wide area Ethernet. He is now a Senior Research Engineer, Supervisor, in Cyber Security Project of NTT Secure Platform Laboratories. He was with the Network Services of NTT communications from 2014 to 2017. He is a member of IEICE.

Masayuki Murata received the M.E. and D.E. degrees in Information and Computer Science from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. From September 1987 to January 1989, he was an Assistant Professor with Computation Center, Osaka University. In February 1989, he moved to the Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University. In April 1999, he became a Professor of Graduate School of Engineering Science, Osaka University, and is now with Graduate School of Information Science and Technology, Osaka University since April 2004. His research interests include information network architecture, performance modeling and evaluation. He is a member of IEEE, ACM and IEICE.