Detecting Malicious Websites by Integrating Malicious, Benign, and Compromised Redirection Subgraph Similarities

Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, Takeshi Yada
NTT Secure Platform Laboratories, Tokyo, Japan
Email: shibahara.toshiki@lab.ntt.co.jp, takata.yuta@lab.ntt.co.jp, akiyama@ieee.org, yagi.takeshi@lab.ntt.co.jp, yada.takeshi@lab.ntt.co.jp

***Abstract*—To expose more users to threats of drive-by download attacks, 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 a part 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 a part 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 websites 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 conventional systems.**

I. Introduction

Drive-by download attacks have become the main vector for malware distribution through the Web. 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 [1]. 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 with the old version of content management systems (CMSs) are vulnerable, attackers automatically discover them with search engines using specific search queries, typically called "search engine dorking" [2], [3], and compromise them to turn them into landing URLs. Malicious websites are automatically created with exploit kits [4].

Against 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 based on malicious web content [5], [6], redirection chains [7], [8], and exploit kits [9]. All of the above systems require malicious data to be collected by accessing malicious websites with a honeyclient that is a decoy browser [10]. Unfortunately, it is commonly known that 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) [11]. In addition, if a client environment differs from the attackers' targets, the attackers thwart its accesses by changing the server responses of malicious websites, which is called "cloaking" [12]. 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.

In this paper, we propose a system for detecting malicious websites without collecting all malicious data. Even if we cannot observe a part of malicious data, e.g., exploit code and malware, we can always observe compromised websites. 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 websites but 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 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 it achieved a 91.7% true positive rate (TPR) for malicious websites containing exploit URLs at a

0730-3157/17 $31.00 © 2017 IEEE
DOI 10.1109/COMPSAC.2017.105

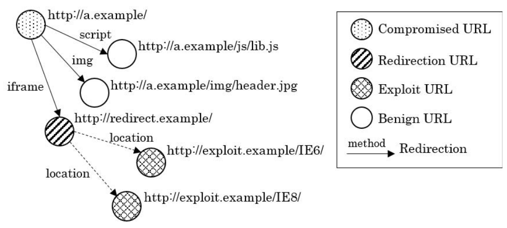


Fig. 1. Redirection graph of motivating example

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4   <meta charset="UTF-8" />
 5   <title>A 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/

low false positive rate (FPR) of 0.1%. Moreover, it detected 143 more malicious websites employing 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 compromised websites, not only those of malicious 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.

II. Motivating example

We use simplified websites to motivate 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. An `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 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 raise no exceptions regarding ActiveXObject at all times [10]. However, since this 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 other clients except for 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.

There are several reasons why conventional systems cannot detect this example. 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 based on URLs, redirections and web content do not work effectively when these honeyclients cannot collect malicious data.

Our system can detect malicious websites employing evasion techniques by utilizing the redirection graphs of all websites observed at the time of access without limiting 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 that the compromised website originally refers to, 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

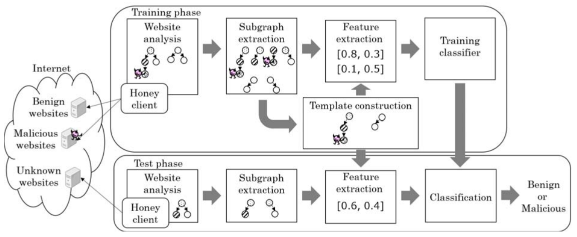


Fig. 4. System framework

we fail to observe a part of malicious redirections and web content due to evasion techniques.

III. 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.

A. 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 of the popular approaches 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 them. This approach leads to high classification accuracy, but it 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 Section III-B, 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 represents 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 $O(M)$.

Figure 4 illustrates the framework of our system. In the training phase, we collect labelled redirection graphs (malicious or benign) with the honeyclient [13]. 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 is trained with their feature vectors. In the test phase, we collect unlabeled redirection graphs with the honeyclient. The redirection graphs of websites are decomposed into their subgraphs. Their feature vectors are extracted on the basis of the similarities between their subgraphs and templates, and classified with the trained classifier.

TABLE I
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

TABLE II
Methods of redirection

No.	Type	Examples of methods
1	HTTP 3xx	HTTP 301, HTTP 302
2	HTML tag	`iframe`, `script`, `link`
3	JavaScript	`document.write`, `innerHTML`

B. 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 $P = \{p_{i,j} | v_j \in c(v_i)\}$, 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 also use the information of vertices and edges for subgraph extraction. From path $p_{i,j}$, we extract subgraph $sg_{i,j}$, which is a tuple of a vector containing the information of vertices and a vector containing the information of redirections. A redirection graph is decomposed into a set of subgraphs extracted from all paths.

As the information of vertices for subgraphs, we calculate the maliciousness of web content because exact matching of URLs and their web content can be easily evaded by

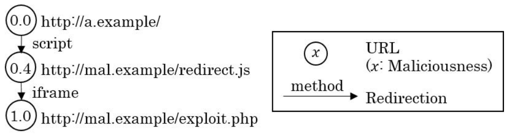


Fig. 5. Example of subgraph

changing the characters of URLs and small pieces of their web content. The maliciousness is calculated with machine learning using the widely used 22 features shown in Tab. I. These features are extracted with the honeyclient [13]. 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). 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 lengthes are 350 or more. To calculate maliciousness, we apply random forest [14] as a classifier. We train the classifier with the same training data in Section IV. As the information of edges for subgraphs, we use redirection methods and destination domains (external or internal). Table II 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 vertex is 0.0, 0.4, and 1.0, respectively. The redirection method of the first and second edges is `script` and `iframe`. The destination domain of the first and second edge is *external* and *internal*. From this example, the subgraph $sg = (m, r)$ is extracted, where $m = [0.0, 0.4, 1.0]$, $r = [\texttt{script}\text{−external}, \texttt{iframe}\text{−internal}]$. The information of edges is represented by the hyphenation of the redirection method and the destination domain. Hereafter, we attach new indexes to subgraphs and represent a subgraph 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 as a similar manner of 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 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 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 sets SG_i and SG_j, is defined as

$$\begin{aligned} S(SG_i, SG_j) &= \frac{2 \times \sum_{k,l \in \Lambda} s(m_k, m_l)}{|SG_i| + |SG_j|}, \\ s(m_k, m_l) &= \frac{1}{1 + \alpha \times |m_k - m_l|^2}, \\ \Lambda &= \{k, l | r_k = r_l, sg_k \in SG_i, sg_l \in SG_j\}, \end{aligned}$$

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-pair possibilities, we adopt the pair whose similarity is higher than the others. The optimal pair can be quickly found using the Hungarian algorithm [15].

This similarity is utilized for clustering. If the maximum similarity between redirection graphs belonging to one cluster and redirection graphs belonging to another cluster 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.

The computational cost of clustering is 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) [16] to avoid calculating the similarities of redirection graphs whose similarities are low. 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 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 x or smaller. 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 whose hash values are the same is not too small.

We construct templates from the clusters composed of γ or more redirection graphs. The template, T, is a set 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 of maliciousness as the maliciousness of subgraphs in the template. Template T is formulated as

$$\begin{aligned} T &= \{(m_r, r) | r \in R\}, \\ R &= \{r | \forall SG \in C, \exists m, (m, r) \in SG\}, \\ m_r &= \frac{1}{|C|} \sum_{SG_i \in C} m_i \text{ s.t. } (m_i, r_i) \in SG_i, r_i = r. \end{aligned}$$

Feature extraction. A high similarity between the 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 subgraphs of redirection graphs and templates. We extract a feature vector x containing similarities between a subgraph set SG and templates T_i: $x = [S(SG, T_1), ..., S(SG, T_N)]$, where N is the number of templates.

Classification. These feature vectors are classified with supervised machine learning. We use random forest [14], which can classify a large amount of data accurately and quickly. We use a `randomForest` package in R [17]. Note that the algorithm of classification is not limited to random forest, but other algorithms of supervised machine learning can be applied.

IV. Experimental setup

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

A. Conventional systems for comparison

We evaluated the effectiveness of our system by comparing it with conventional systems. Some conventional systems detect malicious websites by matching redirection or exploit URLs [7], [9]. These systems suffer from false negatives when targeted URLs are concealed by evasion techniques. Other conventional systems using statistical features [5], [6], [8] are assumed to be more robust against evasion techniques because their targets are not limited to specific URLs. For this reason, we compared our system with conventional systems using statistical features. Note that we cannot compare our system with conventional systems leveraging large-scale user traffic [18], [19] because our system is supposed to use web content and redirections collected with a honeyclient.

We compared 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 on Tab. I such as the conventional system [5] and classifies them with random forest. If one 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 with random forest. Table III shows a list of features that have been proposed for the conventional system [8]. More precisely, the features include 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.

TABLE III
Features of 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

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 systems. A redirection graph is classified as benign if it is detected by neither system.

B. Ground truth

We collected the redirection graphs used for the evaluation by accessing websites listed on public blacklists [20], [21] or a popular website list [22] with the low-interaction honeyclient [13]. Some websites listed on public blacklists no longer contain malicious web content, and websites listed on a popular website list can be compromised and forced to engage in attacks. Since we need ground truth of redirection graphs, we accessed websites with low-interaction and high-interaction honeyclients [23] almost at the same time. The high-interaction honeyclient detected exploit URLs on the basis of system behavior such as unintended process creation and file/registry accesses. We labelled redirection graphs that were detected by the high-interaction honeyclient as malicious, and redirection graphs that were listed on the popular website list and were not detected as benign. We did not use the redirection graphs that were listed on public blacklists but were 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 did not necessarily contain exploit URLs due to evasion techniques even if the high-interaction honeyclient accessed exploit URLs. We used malicious redirection graphs that did not contain exploit URLs as evasive malicious redirection graphs.

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

C. Dataset

We used one training dataset and two test datasets for our evaluation as shown in Tab. IV. The training dataset consisted of 2,170 malicious redirection graphs collected from Jan.–Apr. 2016 and 199,982 benign redirection graphs collected in Aug. 2016. We confirmed that each malicious redirection graph in the training dataset contained at least one malicious redirection. Note that we could collect malicious redirection graphs because their targeted environment and the environment of the low-interaction honeyclient were identical or evasion techniques were not employed. The benign training dataset should be collected during the same period when the malicious dataset was collected. However, we did not collect benign redirection graphs during that period; hence, we used benign redirection graphs collected in Aug. 2016 as a training dataset. Note that collection period is not assumed to affect the evaluation results because the redirection graphs of benign websites are not subject to change.

We used 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 consisted of 365 malicious redirection graphs collected from May–Sep. 2016 and 349,958 benign redirection graphs collected in Aug. 2016. We used the second test dataset to evaluate the detection performance with evasive malicious redirection graphs. It consisted of 3,385 evasive malicious redirection graphs collected from Jan.–Sep. 2016. We used evasive malicious redirection graphs that were collected from Jan.–Apr. 2016 and not used for training data to expand the number of data. These datasets did not mutually overlap.

D. 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 optimized 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 optimized the percentage θ of malicious redirection graphs that we used for training. The number of malicious redirection graphs was limited, but we could collect a large number of benign redirection graphs. We optimized 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 included 90% of the training dataset selected from the oldest and used for training a classifier. The validation dataset included 10% of the training dataset selected from the newest and used for evaluating the classification performance. We selected the hyperparameters whose classification performance was the highest. To evaluate the performance, we used the f-measure defined as: f-measure $= \frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$, precision $= \frac{\text{TP}}{\text{TP+FP}}$, recall $= \frac{\text{TP}}{\text{TP+FN}}$, where TP, FN, TN, and FP denotes 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, were optimized with the `tuneRF` function of the `randomForest` package in R [17] when a classifier was trained.

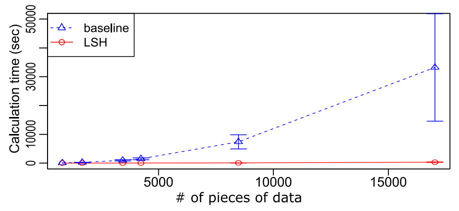


Fig. 6. Calculation time of clustering. Error bars represent standard deviation.

Our system leverages LSH to reduce the computational cost of clustering. We evaluated the effectiveness of LSH by comparing the calculation times with clustering without LSH (baseline). Figure 6 shows the average and standard deviation of calculation time. If the number of data was the same, the calculation time differed depending on the hyperparameters. This result shows that LSH drastically sped up clustering.

V. Experimental results

We now report the experimental results of the evaluation using datasets of malicious redirection graphs containing exploit URLs and evasive malicious redirection graphs. The prototype version of our system was installed on an Ubuntu server with four 12-core CPUs and 128-GB RAM.

A. Detecting malicious websites containing exploit URLs

We evaluated our system using malicious redirection graphs containing exploit URLs. From the perspective of deployment, we also evaluated the degradation of classification performance over time and calculation time. We further manually inspected false positives and false negatives to verify the validity of our system. A case study of server-side cloaking is illustrated to show the effectiveness of our approach.

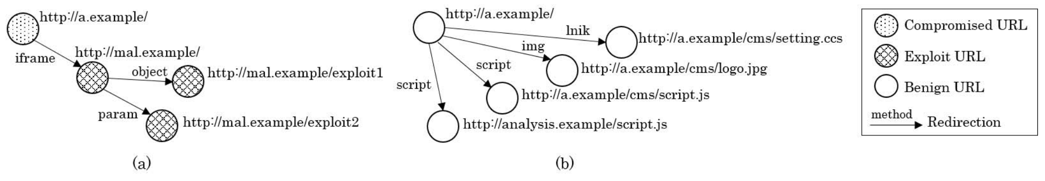
Classification performance. We evaluated our system by using widely used metrics: TPR, which is also known as recall, FPR, precision, f-measure, area under the Receiver Operating Characteristic curve (AUC), and TPR at a fixed FPR of 0.1%. As shown in Tab. V, our system outperformed all conventional systems for all metrics. These results show that leveraging the redirection graphs of all websites contributes to improving the classification performance.

TABLE IV
DATASET

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

TABLE V
CLASSIFICATION PERFORMANCE WITH MALICIOUS REDIRECTION GRAPHS CONTAINING EXPLOIT URLS

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



http://a.example/
iframe
http://mal.example/
object
http://mal.example/exploit1
param
http://mal.example/exploit2
(a)
http://a.example/
lnik
http://a.example/cms/setting.ccs
img
http://a.example/cms/logo.jpg
script
http://a.example/cms/script.js
script
http://analysis.example/script.js
(b)
Compromised URL
Exploit URL
Benign URL
method
Redirection

Fig. 7. Redirection subgraphs of templates

TABLE VI
STATISTICS OF TEMPLATES

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

We analyzed the templates of our system to elucidate the improvement in classification performance. Table VI 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 amount of contribution of each template to classification. More than half of templates were constructed from only benign websites. This is because the number of benign redirection graphs was larger than that of malicious redirection graphs. However, the *importance* of malicious templates tended to be larger than that of benign and compromised templates. Some benign and compromised templates also had a high order of *importance*.

One of the malicious templates having a high order of *importance* contained redirection subgraphs relevant to the Angler Exploit Kit as shown in Fig. 7(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).

One of the compromised templates having a high order of *importance* contained redirection subgraphs relevant to a CMS as shown in Fig. 7(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' administrator use it.

One of the benign templates having a high order of *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 skip 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.

Classification performance degradation over time. The prevalence of new exploit kits and updates to exploit kits degrades the classification performance. Therefore, we evaluated the performance degradation over time. Figure 8 shows the TPR of malicious redirection graphs collected for the first,

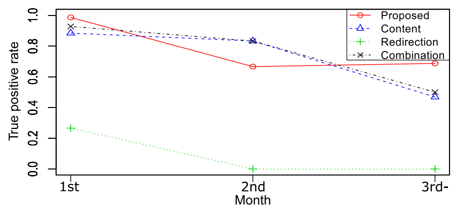


Fig. 8. True positive rate degradation over time

TABLE VII
Calculation time for one website (sec)

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

second, and third or following month of the test dataset. The TPR of our system and the redirection-based system degraded after the first month of the test dataset. The TPR of the content-based and combination systems degraded after the second month of the test dataset. 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 that of conventional systems because our system utilizes more types of websites. That is, our system is focused on malicious, benign, and compromised websites, and change in structure of any one of them degrades classification performance.

Calculation time. To detect malicious websites, we must analyze a large number of websites on the Web. To confirm the capability of large-scale analysis, we evaluated the calculation time of feature extraction and classification. Table VII 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 is shorter than that of the download and execution of web content. Therefore, our system can classify a large number of redirection graphs collected with a honeyclient.

False positives and false negatives. We identified two main cases of false positives with manual inspection. The first one 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 one 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

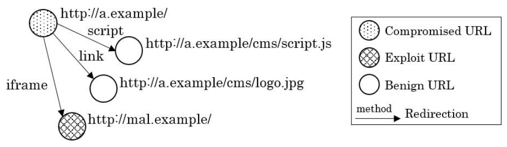


Fig. 9. Redirection graph of website launching server-side cloaking

TABLE VIII
Number of TP and TPR at fixed FPR of 0.1%

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

customized CMS websites as benign.

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

Case study of server-side cloaking. We describe a website for which capturing redirection subgraphs of compromised websites is necessary for detection. The redirection from the landing URL to the exploit URL was triggered by an injected `iframe` tag as shown in Fig. 9. 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.

B. Detecting evasive malicious websites

We evaluated our system using evasive malicious redirection graphs, which did not contain exploit URLs. We further manually inspected false negatives, and a case study of an evasive malicious website is illustrated.

Classification performance. Table VIII 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

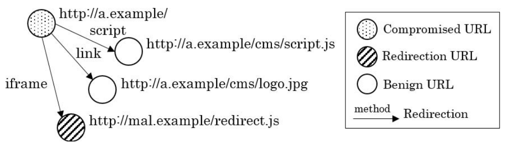


Fig. 10. Redirection graph of evasive malicious website

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

Fig. 11. Evasion code at http://mal.example/redirect.js

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. On these websites, the evasion code was used at landing URLs. Therefore, the low-interaction honeyclient was not redirected to malicious URLs, and it was redirected to only benign URLs. The redirection graphs of these websites were the same as those of benign websites. For this reason, our system could not detect them. Note that we did not find any websites containing malicious redirections with manual inspection of 100 false negatives.

Case study of evasive malicious website.

We describe the evasive malicious website shown in Fig. 10. This website (redirection URL) was pointed by the `iframe` tag injected at the landing URL. Figure 11 shows the evasion code created with Angler Exploit Kit.A different value was assigned to `flag` depending on the error when the 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 no anti-virus software was installed in the high-interaction honeyclient, 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.

VI. Limitation

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 whose subgraphs were different from those of typical malicious websites (as discussed in Section V-A) and malicious redirection graphs that evasion code was used at landing URLs (as discussed in Section V-B). This is a general limitation, which all machine-learning-based systems suffer from. To detect uncommon malicious redirection graphs, the number of malicious training data must be increased. To detect malicious redirection graphs that evasion code is used at landing URLs, systems that detect injected code in compromised websites [2], [3] can be utilized complementary. 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 with 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 that of the conventional systems as discussed in Section V-A. The evaluation result shows our system kept a high classification performance on the first month of test data. Our system can maintain this performance with a classifier that is retrained every month using data labelled by high-interaction honeyclients. High-interaction honeyclients are not suitable for large-scale analysis due to their slow processing speed but are useful for labelling a limited number of websites.

VII. Related work

A. Malicious website detection

From different perspectives, many systems have been proposed 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 [18], [19]. 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 it is generally difficult to obtain these logs. From the perspective of deployment, we designed our system to use data collected with a honeyclient.

System behavior. Decoy client systems employing actual browsers have been proposed to detect exploit accurately by monitoring unintended process creation and file/registry accesses [23]–[25]. 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 Section I and II. 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 with browser emulators and classify them with machine learning. Browser emulators developed for lightweight collection are known as low-interaction honeyclients. Malicious web content has distinctive features to exploit known CVE-ID (Common Vulnerability and Exposures identification) or trigger malicious redirections. Some systems are focused on JavaScript code and HTML tags [5], [6]. Other systems are focused on multistage redirections such as the difference in domains and duration of redirections [8], URLs shared across malicious websites [7], sequences of URLs [26], and URLs/HTTP headers and redirections between them [9]. 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.

B. Compromised website detection

Detecting compromised websites is another approach to preventing damage caused by drive-by download attacks. Soska et al. [2] detect websites that become malicious in the future by focusing on web content that is not generated by users, such as vulnerable CMSs. Li et al. [3] 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.

VIII. Conclusion

We proposed a system for detecting malicious websites on the basis of the redirection graphs of all websites, albeit some malicious web content is concealed. We extracted redirection subgraphs shared across malicious, benign, and compromised websites as templates and classified websites using feature vectors containing similarities between their subgraphs and the templates. We confirmed that templates contained redirection subgraphs of exploit kits, compromised websites, and advertiser websites. These templates enabled 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 found that it achieved a 91.7% TPR for malicious websites containing exploit URLs at a low FPR of 0.1%. Moreover, our system detected 143 more evasive malicious websites compared with the conventional system.

References

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

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

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

[4] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis *et al.*, "Manufacturing compromise: the emergence of exploit-as-a-service," in *Proceedings of the 19th ACM Conference on Computer and Communications Security*, 2012, pp. 821–832.

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

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

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

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

[9] 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," in *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*, 2016, pp. 255–266.

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

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

[12] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: De-cloaking internet malware," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012, pp. 443–457.

[13] Y. Takata, M. Akiyama, T. Yagi, and S. Goto, "Website forensic investigtion to identify evidence and impact of compromise," in *Proceedings of the 12th International Conference on Security and Privacy in Communication Networks*, 2016.

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

[15] 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.

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

[17] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org/

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

[19] 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." in *Proceedings of the 21st Network and Distributed System Security Symposium*, 2014.

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

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

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

[23] M. Akiyama, M. Iwamura, and Y. Kawakoya, "Design and implementation of high interaction client honeypot for drive-by-download attacks," *IEICE Transactions on Communications*, vol. 93, no. 5, pp. 1131–1139, 2010.

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

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

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