Website Forensic Investigation to Identify Evidence and Impact of Compromise

Yuta Takata[1,2(✉)], Mitsuaki Akiyama[1], Takeshi Yagi[1], Takeshi Yada[1], and Shigeki Goto[2]

[1] NTT Secure Platform Laboratories, Tokyo, Japan
{takata.yuta,akiyama.mitsuaki,yagi.takeshi,yada.takeshi}@lab.ntt.co.jp
[2] Waseda University, Shinjuku, Japan
goto@goto.info.waseda.ac.jp

Abstract. Compromised websites that redirect users to malicious websites are often used by attackers to distribute malware. These attackers compromise popular websites and integrate them into a drive-by download attack scheme to lure unsuspecting users to malicious websites. An incident response organization such as a CSIRT contributes to preventing the spread of malware infection by analyzing compromised websites reported by users and sending abuse reports with detected URLs to webmasters. However, these abuse reports with only URLs are not sufficient to clean up the websites; therefore, webmasters cannot respond appropriately to such reports. In addition, it is difficult to analyze malicious websites across different client environments, i.e., a CSIRT and a webmaster, because these websites change behavior depending on the client environment. To expedite compromised website clean-up, it is important to provide fine-grained information such as the precise position of compromised web content, malicious URL relations, and the target range of client environments. In this paper, we propose a method of constructing a redirection graph with context, such as which web content redirects to which malicious websites. Our system with the proposed method analyzes a website in a multi-client environment to identify which client environment is exposed to threats. We evaluated our system using crawling datasets of approximately 2,000 compromised websites. As a result, our system successfully identified compromised web content and malicious URL relations, and the amount of web content and the number of URLs to be analyzed were sufficient for incident responders by 0.8% and 15.0%, respectively. Furthermore, it can also identify the target range of client environments in 30.4% of websites and a vulnerability that has been used in malicious websites by leveraging target information. This fine-grained information identified with our system would dramatically make the daily work of incident responders more efficient.

Keywords: Compromised website · Drive-by download · Redirection graph

R. Deng et al. (Eds.): SecureComm 2016, LNICST 198, pp. 431–453, 2017.
DOI: 10.1007/978-3-319-59608-2_25

1 Introduction

Attackers redirect many unsuspecting users to malicious websites by compromising popular websites and integrating them into a drive-by download attack scheme. One security vendor reported that approximately 67% of malicious websites originated from compromised websites [1]. For example, Darkleech attacks exploiting vulnerable Apache modules had successfully compromised a large amount of websites; over 40,000 domains and IP addresses by May 2013, including 15,000 that month alone [2]. This means that even attentive users will be exposed to drive-by malware infections if high-reputation websites are compromised. An incident response organization such as a CSIRT (Computer Security Incident Response Team) contributes to preventing the spread of malware infection by patrolling the Web and warning users. As part of the patrol, the organization re-analyzes compromised websites reported by users, identifies evidence of malicious websites, and shares this information [3]. This shared information is important for cleaning up compromised websites by reporting abuse to webmasters. Abuse reporting has been conducted as a national project and as a security service that contributes to cleaning up compromised websites by re-analyzing URLs shared from various security vendors [4] and security products [5]. However, attackers build a redirection chain to evade analysis as well as to dynamically and selectively inflict malware on targeted users [6,7]. On compromised websites, attackers can prevent any disclosure of malicious content by injecting only redirection code that leads to malicious websites, not exploit code or malware. This redirection chain also allows attackers to use cloaking techniques to launch drive-by downloads depending on the user's client environment and makes it difficult to analyze [8,9]. Therefore, to mitigate these anti-analysis techniques and expedite the clean-up of compromised websites, it is important to identify the evidence and impact of compromise. Identifying evidence that a website has been compromised, such as the precise position of compromised web content and malicious URL relations in a redirection chain, contributes to shortening the incident response time and increasing clean-up rates. Identifying the impact of a compromised website, such as the target range of client environments and information of vulnerability abused in malicious websites, contributes to shortening the re-analysis time in addition to accelerating security updates to users of the targeted client environments. Li et al. reported that it is important to give more detailed diagnostic information, such as injected content, to webmasters because they lack sufficient expertise to clean up their websites [5]. They also found that the most challenging incident type relates to redirect attacks where websites become cloaked gateways.

To identify the evidence and impact of compromise, we propose a method of constructing a redirection graph by tracing redirection chains and JavaScript executions on websites. After extracting a malicious path, which is a redirection path to a malicious URL, our method identifies the web content that is the origin of the redirection, i.e., compromised web content, by traversing backwards along the malicious path. Our system with the proposed method accesses a website using a multi-client environment to identify targeted client environments.

This environment detects the differences of redirected URLs using these multiple analysis results while minimizing the number of environment profiles by designing them on the basis of known vulnerability information. To the best our knowledge, our system is the first tool for *website forensics* that can automatically identify the evidence and impact of compromise on the basis of useful forensic artifacts, e.g., packet capture data or website data. Specifically, this system can reveal which from web content does a redirection originate, which URLs are associated with attacks, and which client environment is exposed to threats. This fine-grained information would provide practical directions to CSIRTs/security vendors for prompt incident response and expedite compromised website clean-up.

In summary, we make the following contributions.

- Our system successfully identified the precise position of compromised web content and malicious URL relations. As a result, the amount of web content and the number of URLs to be analyzed were sufficient for incident responders by 0.8% and 15.0%, respectively.
- We show that our system can automatically identify client-dependent redirections and the target range of client environments in 30.4% of websites. Using target range information, we can also identify a vulnerability that has been used in malicious websites.

The rest of this paper is structured as follows. In Sect. 2, we provide an overview of compromised website response and explain problems in conventional methods. We introduce our proposed method for addressing the challenges in Sect. 3. In Sect. 4, we explain an experiment conducted to evaluate our method and discuss case studies on our findings in Sect. 5. We discuss the limitations of our method in Sect. 6 and review related work in Sect. 7. We conclude the paper in Sect. 8.

2 Overview of Compromised Website Response

Most of the techniques used by attackers on compromised websites are injections of redirection code to malicious URLs rather than of exploit code or malware. Therefore, identifying web content that is the origin of redirection (redirection origin) is important in the analysis of compromised websites. However, attackers use various anti-analysis techniques to evade a defender's analysis and detection. In this section, we explain web compromise and anti-analysis techniques. We also provide an overview of compromised website response by CSIRTs/security vendors and explain problems in conventional methods.

2.1 Web Compromise Technique

Attackers use redirect code injections using HTML tags or JavaScript to compromise websites.

HTML-Based Compromise. HTML-based compromises inject the redirection code of the `iframe` and `script` tags. These HTML tags are mainly injected into unusual positions in the Document Object Model (DOM) tree such as outside an `html` tag or `body` tag. In the case of an `iframe` tag, many redirections occur without a user being aware by injecting the tag in an invisible state on the browser. A `script` tag is also used in combination with the following JavaScript-based compromise. However, since these tags are directly written in an HTML file, it is easy to analyze them and find the redirection origin.

JavaScript-Based Compromise. JavaScript-based compromises execute code that dynamically generates the above-mentioned HTML tags (`iframe` and `script` tags) using `document.write`, `innerHTML`, and `appendChild` (DOM API code). A `location` object that redirects to a different URL is also injected, but the user is aware of the automatic redirection because it explicitly switches the browser frame to a different URL. Therefore, it is rare to use a `location` object as a first step. JavaScript-based compromises can target various web content, e.g., that enclosed by a `script` tag and that of a URL that is loaded by a `script` tag. The DOM API code and code separation make it difficult to analyze JavaScript. In addition, attackers utilize obfuscation techniques, as described in the next section, on JavaScript to conceal the redirection origin.

2.2 Anti-analysis Technique

In most cases, attackers leverage various existing web techniques, such as code obfuscation, redirection chains, and browser fingerprinting, to protect their own malicious content against the inspections of CSIRTs/security vendors.

Code Obfuscation. Code obfuscation is generally used for code protection and code minimization. For example, obfuscated JavaScript is de-obfuscated by string manipulation functions, and this de-obfuscated string is executed as JavaScript code by functions such as `eval`, `setInterval`, and `setTimeout`. Malicious websites try to prevent analysis by using complicated obfuscation techniques combined with compression techniques[1], cryptographic techniques, and browser-specific functions.

Redirection Chain. Drive-by download attacks redirect users of a landing website (landing URL) to malicious websites (exploit URL) via multiple websites (redirection URL). When a client accesses an exploit URL, an attack code that exploits the vulnerabilities of the web browser and/or its plugins is executed and forces the client to download and install malware from a website (malware distribution URL) [6]. This redirection chain is composed of HTTP 3XX in addition to HTML tags and JavaScript. Attackers abuse compromised popular websites and web search results as landing URLs to lure unsuspecting

[1] D. Edwards, "/packer/," http://dean.edwards.name/packer/.

users by constructing an inter-domain redirection chain to malicious URLs [10]. Therefore, they only have to inject redirection code rather than exploit code or malware for website compromises and can prevent any disclosure of malicious content. Moreover, multiple redirection stages contribute to reducing the operation cost of attacks since compromised websites under a chain can be integrated into a different malware campaign by changing only the redirection URLs.

Browser Fingerprint. Browser fingerprinting, which is a method of profiling the environment of a client, i.e., browser and browser plugin, is generally used for user tracking and distributing web content according to the environment. Attackers leverage browser fingerprinting to target clients by redirecting an exploitable user to a malicious URL on the basis of the client's fingerprint. This technique, called "cloaking," is also abused for circumventing the detection of CSIRTs/security vendors by redirecting them to a benign URL [8].

2.3 Problems in Conventional Methods for Compromised Website Response

An incident response organization, such as a CSIRT, constantly patrols whether websites that are under their own organization and hosting services have been compromised, i.e., the active crawls in Fig. 1 [1]–[3]. Such organization also re-analyzes compromised websites that are reported by general public users and sends abuse reports with the detected URL to webmasters after confirming the reproducibility of attacks, i.e., the reactive crawls in Fig. 1 ①–⑤ [3]. However, in many cases, an abuse report with only URLs generated in this way is not sufficient to clean up compromised websites; therefore, webmasters cannot respond appropriately to the report with just URLs. Moreover, malicious websites cannot always be detected due to cloaking. Therefore, to create detailed abuse reports and increase clean-up rates, the following information is required.

- **Redirection origin:** Identifying a fine-grained redirection origin as evidence that a website has been compromised, such as which web content redirects to which malicious website, is important for webmasters when cleaning up compromised web content precisely. Thus, we must handle complicated obfuscations and redirection chains.
- **Targeted client environments:** Identifying targeted client environments as the impact of a compromised website, such as which versions of browsers

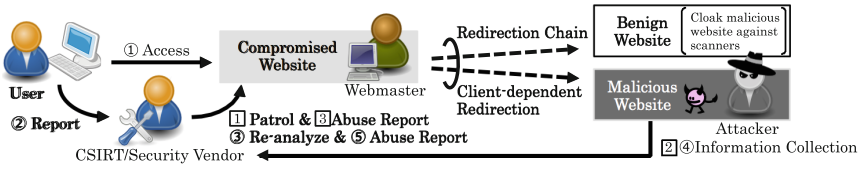

Fig. 1. Overview of compromised website response

and/or plugins are redirected to malicious websites, is beneficial for confirming the reproducibility of attacks. In addition, we can also accelerate security updates by warning users of the targeted client environments. Thus, we must mitigate cloaking techniques.

However, conventional methods are not sufficient for identifying the information stated above. Methods of detecting website compromises that compare original web content to compromised web content have been proposed [11,12]. Moreover, TripWire [13], widely known as a compromise detection tool, can detect file operations, such as modification and deletion, by monitoring files on a web server. However, these methods have limitations in terms of operation; for example, they require the original files and can detect only compromised web content on one's own web server.

Methods for constructing a redirection graph, in which the nodes represent accessed URLs and directed edges represent redirection methods, by using a `Referer` header or a `Location` header [14] and by leveraging some heuristics/features in addition to HTTP headers [15] have been proposed. However, in many cases, the `Referer` header is not set. Additionally, these methods cannot connect tricky links such as a redirection with an inconsistent `Referer` header. This *semantic gap* in the `Referer` header occurs when the redirection results from an external JavaScript.

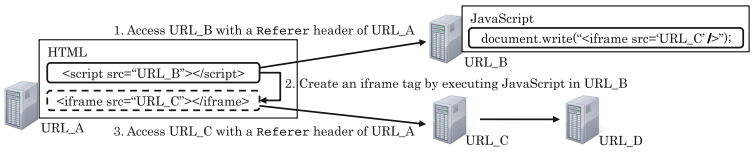

Fig. 2. Semantic gap between Referer header and JavaScript redirection

We now give more details on the semantic gap in a redirection graph using the website in Fig. 2. In this website, a web browser loads JavaScript of URL_B by using a `script` tag in URL_A accessed first. Next, the DOM API code in URL_B is executed. In this case, an `iframe` tag that points to URL_C is inserted into the HTML of URL_A. As a result, an HTTP request to URL_C is generated with the `Referer` header of URL_A. The `Referer` header indicates the base URL, i.e., URL_A, of the web content that is rendered on the web browser, not the external JavaScript URL, i.e., URL_B, that contains the redirection code. This semantic gap occurs due to the general behavior of web browsers and is frequently observed on legitimate websites. However, this gap results in a logically incorrect redirection graph without some edges, for example, an edge from URL_B to URL_C is not connected, which we call a *semantic gap edge*. In other words, when URL_D is a malicious URL, a conventional redirection graph cannot identify the `document.write` statement in URL_B as a redirection origin due to a semantic gap even if traversing backwards along the path from URL_D to URL_A.

3 Proposed Method and System

To identify the redirection origin, we propose a method of constructing a redirection graph with context, such as which content redirects to which malicious websites, by tracing the redirection and JavaScript execution processes. The combination of a redirection graph and a JavaScript execution graph, which we call a "redirection call graph" (RCG), can bridge semantic gap edges and contribute to identifying the precise position of redirection origins. We implemented a system with our method, as shown in Fig. 3. Our system accesses a website using a multi-client environment to identify targeted client environments while constructing RCGs. It detects the differences of accessed URLs among multiple analysis results while minimizing the number of environment profiles by designing them on the basis of known vulnerability information. We detail each system component in the following subsections.

3.1 Identifying Redirection Origin as Evidence of Compromise

Our method of identifying redirection origins is composed of a *monitoring behavior* phase, *constructing RCG* phase, *identifying malicious node* phase, and *extracting compromised content* phase (① in Fig. 3).

Monitoring Behavior. Our system accesses websites and collects redirection and JavaScript execution traces by monitoring behaviors during the process of interpreting fetched web content. We explain the behavioral information as follows.

- **HTTP transaction:** An HTTP response with the status code 3XX is captured in HTTP transactions for tracing HTTP redirections. When an HTTP server responds to this status code, the HTTP request URL, URL in the `Location` header, and HTTP status code are recorded as a redirection source URL, redirection destination URL, and redirection method, respectively.
- **HTML parsing:** Our system monitors HTML tags, e.g., `iframe`, `frame`, `script`, `meta`, `object`, `embed`, and `applet`, that redirect to a different URL

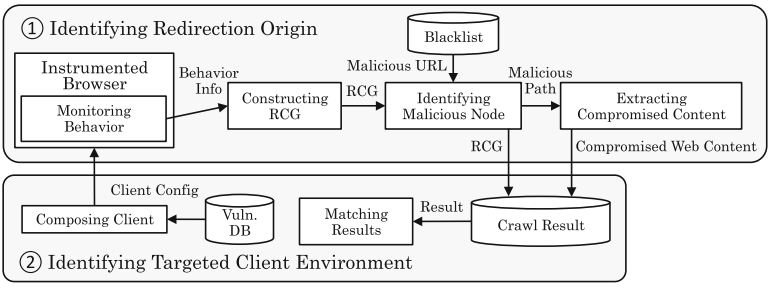

Fig. 3. System overview

during HTML parsing to trace redirections with HTML tags. When these HTML tags are parsed, the URL that contains the HTML tag, URL to which the HTML tag points, and HTML tag name are recorded as a redirection source URL, redirection destination URL, and redirection method, respectively.

– **JavaScript API hooking:** Our system monitors executed JavaScript code and JavaScript function calls, e.g., `eval()`, `setInterval()`, `setTimeout()`, function calls of `window`, `location`, `element`, `node`, and `document` objects, to construct a JavaScript execution graph and connects semantic gap edges. Then, to trace redirections with JavaScript, the JavaScript URL, URL to which the JavaScript points, and JavaScript function name are recorded as a redirection source URL, redirection destination URL, and redirection method, respectively.

Constructing Redirection Call Graph. This phase constructs a RCG based on recorded trace information. As a result, a directed graph with the following nodes and edges, such as in Fig. 4 on the left, is structured.

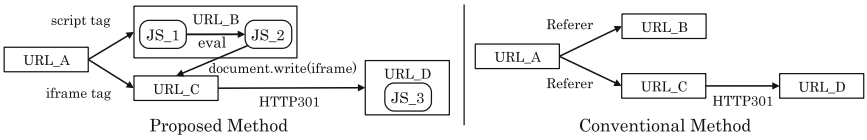

Fig. 4. Comparison of graphs constructed with proposed and conventional methods

– **Redirection node and edge:** A redirection node represents an accessed URL. A redirection edge represents a redirection method and connects redirection nodes. To construct these nodes and edges, we use information obtained from HTTP transaction and HTML parsing in the previous phase.
– **JavaScript execution node and edge:** A JavaScript execution node represents code executed by the JavaScript interpreter. We can identify which code is executed by tracing these code executions. This node is managed by the hash value of the code. Figure 4 shows that a redirection graph contains the hash values of JavaScript execution nodes (JS_1, JS_2, and JS_3 in this case). A JavaScript execution edge represents a JavaScript execution method and connects JavaScript execution nodes, for example, browser rendering, JavaScript events, `eval`, `setInterval`, and `setTimeout`. In addition, this edge contains redirection methods to different URLs to identify JavaScript redirections, e.g., `location.replace()`.
– **Semantic gap edge:** Our method associates an HTML tag generated by JavaScript with the JavaScript URL to bridge a semantic gap edge. When a redirection occurs via the parsing of an HTML tag, e.g., an `iframe` tag and a `script` tag, the source URL is identified from not only the base URL but also the associated JavaScript URL if the HTML tag is generated by JavaScript.

We explain a semantic gap edge using Fig. 2. When `document.write` is executed in URL_B, a pair of URL_B and the `iframe` tag generated by `document.write` is saved. Next, when the `iframe` tag inserted in URL_A is parsed, URL_B is uniquely identified from the pair information. Finally, when the redirection of the `iframe` tag occurs, an edge from URL_B to URL_C is connected. Then, the redirection method of the edge is set to the DOM API function and HTML tag name, "`document.write(iframe)`."

Figure 4 depicts a comparison of Fig. 2 between a RCG and a conventional redirection graph. Our method can identify an obfuscation process from JS_1 to JS_2 by `eval` and connect an edge from URL_B to URL_C by `document.write`. This information is necessary for incident responders to conduct efficient and effective website forensics, but conventional methods cannot identify it.

Identifying Malicious Node. This phase identifies malicious nodes in the RCG constructed in the previous phase using a blacklist of known malicious URLs. These known malicious URLs can be obtained from detection results by using conventional techniques such as a high-interaction honeyclient and antivirus. In addition to matching exact malicious URLs, we detected suspicious URLs of the same domain name and the same number of path hierarchies or the same number of domain name hierarchies and the same path compared with the malicious URLs. This suspicious URL detection helps minimize the effects of URLs using random strings. This phase also extracts malicious paths from identified malicious nodes to the node of the landing URL.

Extracting Compromised Content. A redirection origin is extracted by traversing backwards along a malicious path, which is identified in the previous phase, from the leaf URL to the origin URL. We explain the extraction method in Fig. 4. If the redirection path from URL_A to URL_D is classified as malicious, e.g., JS_3 contains the exploit code, the `script` tag that points to URL_B in URL_A is extracted as a redirection origin. A redirection origin contains the origin/leaf URLs and the redirection method/destination URL. Moreover, to identify the precise position of redirection origins, this phase extracts DOM information, such as the DOM tree structure, in the case of an HTML-based compromise. In the case of a JavaScript-based compromise, the JavaScript execution information is extracted such as executed code.

It is important to note that redirection origin of the landing URL is not always compromised web content. For example, if JS_1 in Fig. 4 is compromised web content, the `script` tag in URL_A described above is a false positive. Therefore, this phase minimizes false positives by following a malicious path from the landing URL to the URL with a domain that is different from the source URL after traversing backwards. This means that we consider web content that generates such inter-domain edge as a redirection origin because the domain of compromised websites is different from that of malicious websites [6]. Specifically, JS_1 in URL_B is detected as a redirection origin by the difference between URL_B's domain and URL_C's domain.

Table 1. Matrix of CVEs and Flash Player versions

	2013-0634	2013-5329	2014-0497	2014-0502	2014-0515	2014-0556	2014-0569	2014-8440	2014-8439	2015-0310	2015-0311	2015-0313	2015-0336	2015-0359
10.1.102.64	✓													
11	✓	✓	✓	✓										
11.2.202.233	✓	✓	✓	✓	✓	✓	✓							
11.5.502.149		✓	✓	✓										
11.2.202.270		✓	✓	✓	✓	✓	✓							
11.7.700.169		✓	✓	✓	✓									
11.7.700.225				✓	✓									
11.7.700.252			✓	✓										
11.7.700.257			✓	✓	✓									
11.2.202.332			✓	✓	✓	✓	✓							
12.0.0.44				✓										
11.2.202.336				✓	✓	✓	✓							
11.7.700.269					✓									
11.2.202.341					✓	✓	✓							
13.0.0.206						✓	✓							
14.0.0.125						✓	✓			✓	✓	✓	✓	✓
14.0.0.179						✓	✓		✓	✓	✓	✓	✓	✓
14.0.0.176						✓	✓	✓		✓	✓	✓	✓	✓
13.0.0.244							✓							
15.0.0.152							✓			✓	✓	✓	✓	✓
13.0.0.250								✓						
15.0.0.189								✓		✓	✓	✓	✓	✓
11.2.202.423									✓					
15.0.0.239										✓	✓	✓	✓	✓
13.0.0.260										✓				
11.2.202.438											✓			
16.0.0.287											✓	✓	✓	✓
11.2.202.440												✓		
13.0.0.264												✓	✓	✓
16.0.0.305													✓	
17.0.0.134														✓

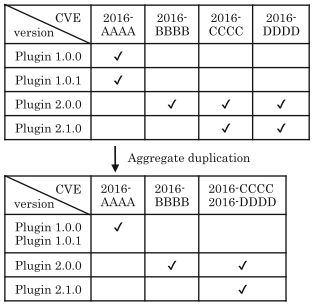

Fig. 5. Aggregation of duplicated CVEs and plugin versions

Table 2. Number of plugin versions

	JRE	PDF	Flash
Exploit kits from 2014–2015	14	1	31
Exploit kits from 2011–2013	37	23	32
Official installer	193	103	251

3.2 Identifying Targeted Client Environment as Impact of Compromise

To identify targeted client environments, our system analyzes a website in a multi-client environment that increases the possibility of changing the behavior of a website by browser fingerprinting, such as boundary testing. The analysis environment is composed of a *composing client* phase and a *matching results* phase (② in Fig. 3).

Composing Client. This phase decides on a client environment from a matrix of vulnerabilities and its affected client environments. Our method can decrease the number of client environments by aggregating the environment's duplications. If we can predict potential targeted vulnerabilities in websites, the number can be further decreased by filtering out the corresponding columns of the matrix (Fig. 5). For example, we show a matrix of the matching of known vulnerability information obtained from CVE Details[2] and affected versions of Adobe Flash Player in Table 1. We further decreased the elements of the matrix by utilizing the vulnerability information of exploit kits from 2014–2015 obtained from contagio[3]. In Table 1, the versions of Adobe Flash Player were aggregated from 251 to 31. Note that oldest version is selected from aggregated versions.

[2] CVE Details, http://www.cvedetails.com/.
[3] contagio, http://contagiodata.blogspot.jp/2014/12/exploit-kits-2014.html.

Matching Results. Our system compares crawl results of various environments and detects differences of accessed URLs among the results, i.e., it investigates whether each crawl result contains malicious URLs. From the matching results, we can identify which client environment is redirected to a malicious URL.

3.3 Implementation

To monitor fine-grained processes of HTML parsing and JavaScript execution and to configure various client environments, we need to be able to hook browser processes and modify the environment profiles. Therefore, we used a browser emulator, HtmlUnit[4], in our system and implemented the monitoring and configuration functions into it. In this paper, we focused on plugins, Java Runtime Environment (JRE), Adobe Reader (PDF), and Adobe Flash Player (Flash), for a multi-client environment because many recent exploit kits check for the presence of vulnerable versions of several plugins [7,9]. Therefore, we collected vulnerability information on these plugins from CVE Details and contagio. The numbers of aggregated versions of JRE, PDF, and Flash are listed in Table 2. The rows of Table 2 represent the number of plugins on the basis of vulnerability information of exploit kits from 2014–2015, exploit kits from 2011–2013, and the number of official installers we found manually. Table 2 shows that our method can dramatically reduce the number of environment profiles by utilizing known vulnerability information. It is important to note that our system can change environment profiles on the basis of not only plugins but also operating systems or browsers in the same way.

4 Experiment and Evaluation

We evaluated the effectiveness and performance of our system using the HTTP communication data of 2,058 compromised websites that were preliminarily detected during a four-year period (2011–2015). Although we can run our system to reveal malicious content and the functions of websites on the *live* Internet, online crawlings, especially with our multi-client environment, place a load on web servers and make it easy to detect inspections by server-side cloaking. Therefore, it is appropriate for utilizing our system in a local environment while leveraging forensic artifacts that have been already detected. In this experiment, we first investigated the impact of semantic gaps to evaluate the effectiveness of an RCG. More precisely, we evaluated whether a RCG can precisely connect more links than a conventional redirection graph. Next, we analyzed redirection origins extracted from malicious paths and investigated the statistical trend regarding website compromises. Finally, we evaluated whether our system can identify targeted client environments and the target range.

[4] Gargoyle Software Inc., http://htmlunit.sourceforge.net/.

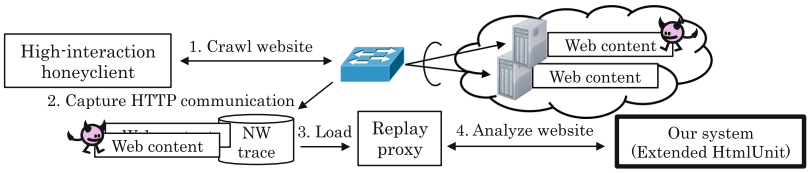

Fig. 6. Experimental environment

4.1 Experimental Environment

The experimental environment for our system was composed of a high-interaction honeyclient, a replay proxy, and our system, as shown in Fig. 6.

High-Interaction Honeyclient. We used the HTTP communication data of websites that were preliminary detected drive-by download attacks by a high-interaction honeyclient [16]. Exploit URLs and malware distribution URLs detected by the honeyclient were also used as a blacklist in the *identifying malicious node* phase.

Replay Proxy. A replay proxy responds to a HTTP request with web content on the basis of a URL using HTTP communication data. Thus, due to the dynamic nature of modern websites, some HTTP requests may not match any of the original data. This occurs when a URL using time-dependent or random parameters is included in the data. To compensate for dynamically generated URLs, we used an *approximate matching* approach, which was inspired from a method [17], during replay. This approach measures the similarity between a requested URL and URLs with the same domain and the same file path but different parameters in the HTTP communication data. To compute a similarity score, this approach calculates a Jaccard index of the set of parameter names. Finally, the proxy responds to a HTTP request with web content on the basis of a URL that has a score that is higher than a threshold. The threshold was set to a high score, e.g., 0.9, to prevent false positives, and no false positives were observed in this experiment. Note that the purpose of this study is to identify the evidence and impact of compromise, and not to propose a traffic replay method.

Our System. Our system, which is the extended HtmlUnit described in Sect. 3.3, analyzes web content and stores the results through accesses to the replay proxy. Then, to further reduce the analysis time, we used our multi-client environment for only websites that tried to use browser fingerprinting. Browser fingerprinting can be detected by monitoring the use of the name and version strings of the client environment in JavaScript function arguments and object properties. Therefore, we preliminarily detected browser fingerprinting by analyzing a website once. The results of preliminary crawls were also used for analyzing a website that does not use browser fingerprinting.

We obtained the experimental results presented in this section by using two servers, both running Ubuntu 12.01. Our replay proxy replayed the HTTP communication data on one server (2.93 GHz processor and 24 GB of RAM), and our system evaluated web content on the other server (3.16 GHz processor and 4 GB of RAM).

4.2 Evaluation of Redirection Call Graph and Redirection Origin

Constructing Redirection Graph. We evaluated how many nodes (URLs) can be connected with the proposed method compared with conventional methods. We computed the differences between the number of nodes on malicious paths identified by the proposed method (PRO) and the conventional methods. As the conventional methods, we implemented originally the referer-based method (REF) [14] and the heuristic-based method (HEU) [15]. As a result, the number of nodes identified by *only* PRO was 1,068 and 367 compared with REF and HEU, respectively. We found through manual inspection that these nodes were false negatives of the conventional methods caused by a redirection without a `Referer` header or with a semantic gap. The semantic gap edge was included in 16.6% of websites. In addition, the numbers of nodes identified by only the conventional methods were 0 and 9 compared with REF and HEU, respectively. However, these nodes were false positives (noise URLs) caused by linking a *likely* edge with the rule "Domain-in-URL" of HEU. We show in detail the differences of redirection graphs between the proposed method and the conventional methods in Appendix. These results show that the proposed method can accurately construct a redirection graph, and identify malicious redirection chains, but the conventional methods cannot.

In this evaluation, we found several redirection graphs without a malicious path. Therefore, we measured the analysis capabilities of our system by calculating its reachability to malicious URLs that the high-interaction honeyclient detected. As a result, our system identified malicious paths from 1,479 (71.9%) websites among the 2,058 websites. We give more details on the websites that could not reach malicious URLs in the next subsection, i.e., these websites correspond to unknown or false negatives.

Redirection Graph Without Malicious Path. We manually analyzed the causes of the *incomplete redirection graphs* that did not contain malicious URLs. Table 3 shows a breakdown of redirection graphs without a malicious path. The most common sophisticated browser fingerprinting in this breakdown changed behavior on the basis of the presence of a specific property of JavaScript or security vendor products. JavaScript properties exist in only Internet Explorer, e.g., `window.sidebar`, and is abused as an indirect browser fingerprint by attackers. Many methods of such browser fingerprinting are proposed and also known to affect the behavior of not only a browser emulator but also a real browser [18]. Attackers can also maliciously access a file system and check the presence of security vendor products through Internet Explorer by abusing an information

Table 3. Breakdown of redirection graph without malicious path

Category	#graph	Reason	Handling
Sophisticated browser fingerprinting	231	Anti-virus detection and browser-specific JavaScript property	Analyze it with a real browser
URLs with random strings and/or domains with DGA	165	Lack of approximate matching and suspicious URL detection ability	Improve accuracy of algorithm
Emulator evasion	122	Defect of DOM implementation in HtmlUnit	Fix it
Time-dependent redirection	57	Past crawl data	Analyze it immediately after detection
VBScript	4	Unsupported in HtmlUnit	Analyze it with real browser

disclosure vulnerability, i.e., CVE-2013-7331. Our browser emulator could not be redirected to malicious URLs because it did not execute the environment-specific code and exploit code. The emulator evasion in Table 3 was caused by a defect of DOM implementation in HtmlUnit. However, we can mitigate the evasion by improving the behavior emulation since a redirection graph could be accurately constructed by fixing this defect. The other causes were lack of approximate matching and suspicious URL detection ability, time-dependent redirections, and use of VBScript.

Extracting Compromised Web Content. To investigate the statistical trend regarding compromised web content and compromise methods, we analyzed redirection origins extracted from malicious paths. Compromise methods were 43% HTML-based compromises, 9% JavaScript-based compromises, and 47% DOM API code injections. Almost all HTML-based compromises injected automatic redirections to different URLs using `script` and `iframe` tags. The DOM API code also injected 98% `iframe` tags and 2% `script` tags. These injected HTML tags were written in strange positions such as outside the `html` tag or `body` tag

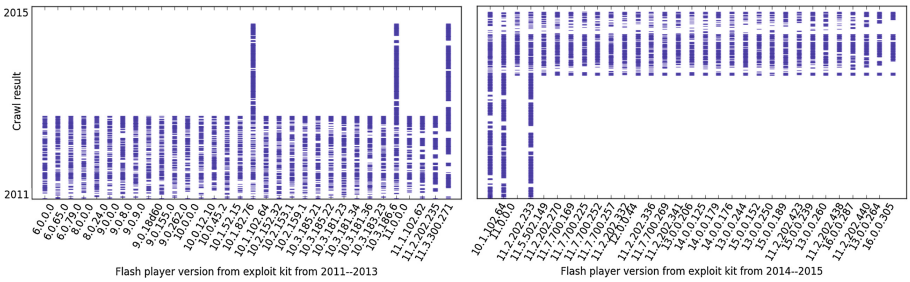

Fig. 7. Identification of target range of Flash Player version

Table 4. Analysis of client-dependent redirection with browser fingerprinting

Detected: Suspicious:Unknown	#crawls	Description
1:0:1	359	Client-dependent redirection with browser fingerprinting
0:1:1	117	Client-dependent redirection with browser fingerprinting
1:1:1	149	Client-dependent redirection with browser fingerprinting
0:0:1	209	Emulator evasion, time-dependent redirection, etc. (see Table 3)
1:1:0	226	Malicious websites with DGA-domain and/or random path
0:1:0	91	Malicious websites with DGA-domain and/or random path
1:0:0	370	Simple malicious websites

(5%) with in a small area (width < 15, height < 15 or area < 30; 20%) or outside the display (72%).

We also investigated redirection paths from compromised web content. As a result, the semantic gap edge was included in 33% of redirection paths, which made it difficult to analyze it. We will give two case studies of these semantic gap edges in Sect. 5.1.

4.3 Evaluation of Targeted Client Environments

We evaluated whether our system can identify which client environment is redirected to a malicious URL. The client environments emulated each plugin, as shown in Table 2, on the basis of the observation period of the websites and the browser fingerprint acquired by the websites. The crawl results per each *environment* were categorized into three groups: detected crawls that contain malicious URLs, suspicious crawls that contain suspicious URLs, and unknown crawls that contain neither. As a result of comparing crawl results per each *website*, we identified client-dependent redirections that contained detected and/or suspicious crawl results at the same time as unknown crawl results from 625 (30.4%) of the websites (Table 4). These websites changed the destination URL depending on the difference among the plugin versions. We plot these detected and/or suspicious crawl results in Fig. 7, in which the horizontal axis indicates versions of Flash (left is from exploit kits from 2011–2013, and right is from exploit kits from 2014–2015) and the vertical axis indicates crawl results on the order of the time scale. Figure 7 shows that some of the results were widely detected, and the others were detected by only specific versions. We found through manual inspection that these results were derived from the exploit kit periods of 2011–2013 and 2014–2015. This means that client environments based on information of

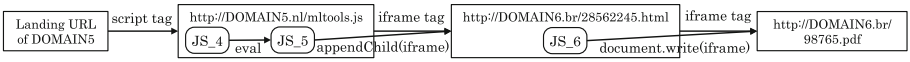

Fig. 8. Malicious path that contains obfuscated semantic gap edge

exploit kits from 2011–2013 were not redirected to malicious websites observed from 2014–2015 and vice versa. These results show that it is important to change a client environment for analysis depending on that attack trend of that time. Furthermore, as a result of analyzing websites of the same detection pattern, we found that these websites used the same browser fingerprinting code and redirection code. Using these multiple analysis results, we can categorize malicious infrastructures, such as vulnerabilities (see Sect. 5.2).

4.4 Performance Overhead

We evaluated the total time and the average time of analyzing 2,058 websites with our system. The results indicated that the time costs were 685,773 s and 333 s, respectively. Since 90% of benign website crawlings done by the high-interaction honeyclient that detected compromised websites used in this experiment finished within 154 s [16], the analysis time of our system took approximately twice as long. The performance of our system, however, clearly depends on the number of environment profiles. The analysis time per one environment was only 12 s on average. Therefore, our system is appropriate for frequent re-analysis of websites because the browser emulator does not require the rendering time of a website and the execution time of exploit code. In addition, since the browser emulator can be more easily deployable and parallelized compared with a high-interaction honeyclient that individually requires a real browser whenever the environment is changed, performance can be further improved.

5 Case Studies

We manually analyzed redirection origins, redirection paths, and client-dependent redirection code. Among these manual inspections, we now describe notable samples.

5.1 Redirection Call Graph with Semantic Gap

Obfuscated Semantic Gap Edge. We depict an example of malicious paths that contained dynamically generated code and a semantic gap in Fig. 8. The semantic gap was caused by DOM API code (JS_5) in obfuscated code (JS_4) injected by compromising. The conventional methods could not completely identify these malicious paths because the link to the URL of DOMAIN5 could not be connected due to the semantic gap and the destination URL of DOMAIN6 is concealed in the obfuscated code, i.e., JS_4.

Multiple Compromised Web Content. We show an example of a part of RCGs constructed from crawl results, which contain two or more differences in

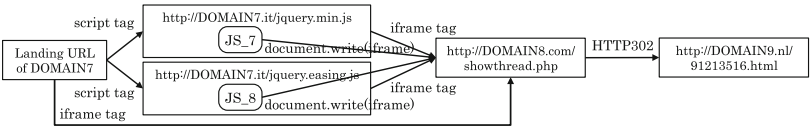

Fig. 9. Malicious path that contains multiple semantic gap edges

Table 5. PDF version range detected by website analysis in multi-client environment

4.0.5	7.0.0	7.1.0	7.1.1	8.0.0	8.1.0	8.1.1	8.1.2	8.1.3	8.1.4	8.2.0	8.2.4	9.0.0	9.1.0	9.1.1	9.3.0	9.3.1	9.3.3	9.4.0	9.4.1	10.0.0	10.0.3	10.1.1
				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					

the number of identified URLs between PRO and REF/HEU in Sect. 4.2 (Fig. 9). Compromised web content in Fig. 9 was injected into multiple files such as an HTML file of the landing URL and JavaScript files referred from the landing URL. The conventional methods could not identify URLs of these JavaScript files because DOM API code were injected into all files and semantic gaps occurred on all of them. In other words, this means that JavaScript files remain compromised even if we deleted only the `iframe` tag of the landing URL identified by the conventional methods.

5.2 Client-Dependent Redirection with Browser Fingerprinting

The JS_6 contained in the redirection path of Fig. 8 changed the destination URL by executing the following browser fingerprinting code that gets the version of PDF plugin.

```
pdf_ver = PluginDetect.getVersion("AdobeReader");
pdf_ver = pdf_ver.split(",");
if ((pdf_ver[0] == 8 && pdf_ver[1] <= 2) || (pdf_ver[0] == 9 && pdf_ver[1] <= 3)) {
    document.write("<iframe width=10 height=10 src='http://DOMAIN6.br/98765.pdf'></iframe>");
}
```

We analyzed the above code using our system that emulated 23 individual versions of a PDF based on Table 2 because the code was observed in 2012. As a result, the versions shown in Table 5 reached malicious URLs and the behavior was along the condition of the above branch code. In addition, these code features and characteristic lexical features of URLs suggest that these malicious paths were built using RedKit, which is known to exploit a PDF's vulnerability (CVE-2010-0188) [19]. CVE-2010-0188 exists in Adobe Reader/Acrobat 8.X before 8.2.1 and 9.X before 9.3.1, and the above code has also been implemented to redirect to the URL of DOMAIN6 when a PDF version that has the vulnerability is used.

6 Discussion

Browser Emulator Limitations. The analysis of malicious websites with a browser emulator, such as our system, is known to have some limitations.

For example, a browser emulator is known not to be able to execute attack code that exploits the vulnerabilities of a web browser and/or its plugins. Our system also cannot execute exploit code as described in Sect. 4.2. In other words, our method cannot construct a complete redirection graph including a malware distribution URL because a malware distribution URL is accessed due to exploit code execution. Similarly, improving behavior emulation is challenging in browser fingerprinting and the diversity of browser implementations. The redirection graphs without malicious paths in Sect. 4.2 were also one of the factors preventing the construction of graphs. We admit all these issues can affect the performance of our system. However, these issues are not specific to our system and affect in some degree all real browsers and browser emulators. More importantly, our system could identify the evidence and impact of 71.9% of compromised websites under the limitations. To maximize the disclosure of malicious content and detect it, we must combine our system with other techniques, which we discuss in Sect. 7.

Evaluation of Compromised Content. In this study, we did not conduct a user study on how the evidence and impact information identified by our system can contribute to remedying compromised websites and preventing malware infections because we evaluated our system using past crawl data in our experiments. As future work, we will perform a user study on how much and how long this identified information can increase the response rate and the reduces response time required for clean-up done by webmasters, such as in an existing user study [5].

Instead of a user study on webmasters, we calculated the content reduction rate (CRR) and the URL reduction rate (URR) to evaluate how our system can contribute to the work of incident responders. The CRR is how much web content on compromised websites would not be analyzed by extracting compromised web content using our method. The URR is how many URLs our method can filter out by extracting malicious redirection paths from the entire redirection graph of each crawling. These rates were obtained with the following formulas.

$$\mathrm{CRR} = 1 - \frac{1}{n}\sum_{k=1}^{n}\left(\frac{\#\text{ of bytes of } \textit{compromised content}_k}{\#\text{ of bytes of } \textit{original content}_k}\right),$$

$$\mathrm{URR} = 1 - \frac{1}{n}\sum_{k=1}^{n}\left(\frac{\#\text{ of access URLs in } \textit{path}_k}{\#\text{ of access URLs in } \textit{crawl}_k}\right)$$

As a result, our method could reduce 99.2% of bytes on the basis of the value in a `Content-Length` header (16,568 bytes on average). Furthermore, the URR was 85.0% (23 URLs on average), i.e., the amount of web content and the number of URLs to be analyzed were sufficient for incident responders by 0.8% and 15.0%, respectively. The results show that our method can identify malicious websites both at a content-level and a URL-level. However, web content dynamically injected, for example, from database and a .htaccess file cannot be accurately identified. Although we must cooperate with webmasters to remove

the root cause of compromise in the case of dynamic compromises, our method can still provide practical directions for prompt incident response.

Accuracy of Vulnerability Database. Our system chose a client environment for emulation on the basis of known vulnerability information. The vulnerability information in Table 1 showed a correlation that old versions have old vulnerabilities and new versions have new vulnerabilities but non-consecutively, e.g., CVE-2014-0497 exists in Adobe Flash Player before 11.7.700.261, but 11.7.700.225 is not checked in Table 1. We can infer that these are derived from the omission of information or untested plugin versions. Therefore, it is important to note that our analysis method using a multi-client environment cannot identify completely the target range of client environments on malicious websites. However, the target range clearly depends on the implementation of malicious websites, and even our method can get enough beneficial information, as described in Sect. 5.2.

7 Related Work

Detecting Compromised Websites. The methods of detecting website compromises are generally used for comparing original and compromised web content. For example, a comparison method [11] using HTML files as original content and a comparison method [12] using well known libraries and frameworks of JavaScript as original content have been proposed. Moreover, TripWire [13] can notify webmasters of changes on websites by e-mail when file operations are detected on a web server on which TripWire is installed. However, these methods have limitations in terms of method application. For example, original content is necessary for compromise detection, and these methods can detect only compromised web content on the web server under control. These limitations prevent websites using external content such as third-party libraries and advertisements from performing effectively. However, using these methods with compromised web content identified by our method can contribute to finding more malicious websites and detoxifying them.

Detecting Malicious Websites. Over the past few years, many researchers have proposed methods of detecting drive-by downloads. A high-interaction honeyclient [16,20] crawls websites with a vulnerable real browser and detects malware downloads by monitoring unintended processes and file system accesses, whereas a low-interaction honeyclient [21,22] crawls websites with a browser emulator and detects malicious behaviors by signature matching and machine learning. Many learning-based detection methods of malicious websites have also been proposed and leveraged features from HTML, JavaScript, URL, and social-reputation [23–25]. However, these methods cannot identify which web content is the redirection origin of a malicious path. In comparison, we can extract malicious paths more effectively using these research results because all methods can detect malicious websites with high accuracy.

Detecting Malicious Redirection. Many methods of detecting a redirection graph on malicious websites rather than for detecting exploit code and malware have also been proposed [15,26–28]. Graph-based methods [26,27] using the behavioral information of web browsers construct a redirection graph on the basis of redirection information collected from a number of honeyclients or a user's clients. These methods detect malicious websites by leveraging co-occurring URLs in graphs and a diverse dataset of graphs. Others [15,28] focus on HTTP redirections and executable file downloads on a network and apply a classifier to detect malicious redirection paths. However, these methods fail to construct a redirection graph of many malicious websites (see Sect. 4.2 and Appendix A) because of the coarse-grained redirection information. These methods can also identify malicious URLs but cannot identify malicious content as well as stated in the previous subsection.

8 Conclusion

To identify the evidence and impact of compromise, we proposed a method of constructing a fine-grained redirection graph. Our system with the proposed method analyzes a website in a multi-client environment while minimizing the number of environment profiles. The evidence and impact information includes which from web content does redirection originate, which URLs are associated with the attacks, and which client environment is exposed to threats. Our evaluation with compromised website data obtained during a four-year period showed that our system can successfully identify the precise position of compromised web content, malicious URL relations, and targeted client environments. We also showed that it can effectively identify an exploit kit and a vulnerability that has been used in malicious websites by leveraging the information. We believe that our system can contribute to improving the daily work of CSIRTs/security vendors and expediting compromised website clean-up done by webmasters.

A Appendix: Difference Between Proposed Graph and Conventional Graph

We show redirection graph examples constructed with the referer-based method [14], the heuristic-based method [15], and the proposed method in Figs. 10, 11, and 12, respectively. Figure 10 depicts a graph smaller than the other graphs because a redirection without a `Referer` header was caused by a function of the `location` object. In this case, the referer-based method cannot connect the any of the following redirections. The heuristic-based method can connect all redirections. However, semantic gaps between `Referer` headers and JavaScript redirections occur. As a result, we cannot identify precise redirection origins, e.g., the web content of URL "http://DOMAIN10/gzcr?t=[a-zA-Z0-9]{118}," due to the gaps. Our method can connect all redirections and precisely identify all of their redirection origins.

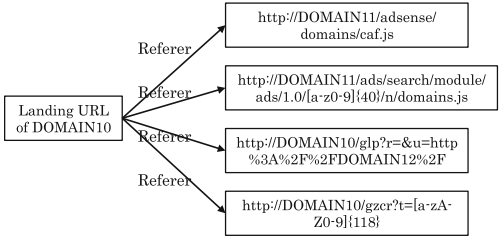

Fig. 10. Redirection graph constructed by referer-based method.

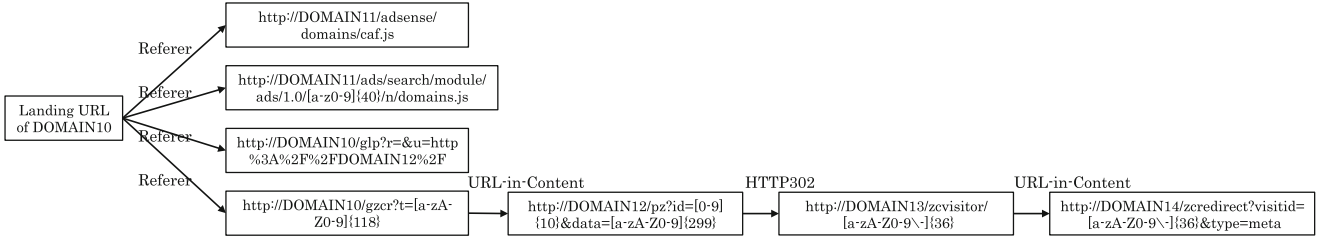

Fig. 11. Redirection graph constructed by heuristic-based method (WebWitness).

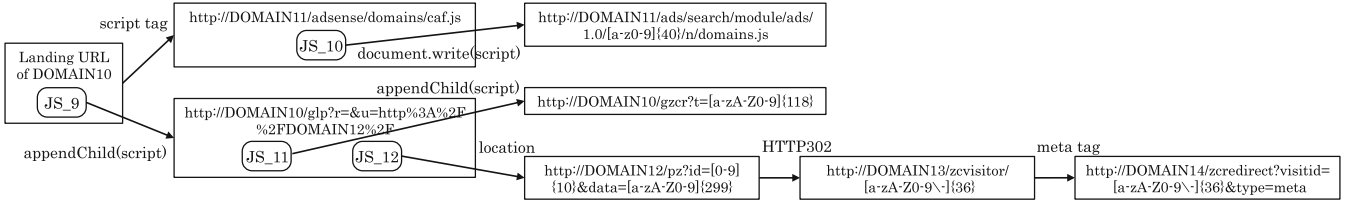

Fig. 12. Redirection graph constructed by proposed method (redirection call graph).

References

1. Symantec Corporation: Internet Security Threat Report 2014: Volume 19 (2014). http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf
2. Sophos Ltd.: Security Threat Report 2014 (2014). https://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf
3. Kobayashi, H., Uchiyama, T.: Keeping eyes on malicious websites - 'ChkDeface' against fraudulent sites. In: The 27th Annual FIRST Conference (2015)
4. Japan's Ministry of Internal Affairs and Communications: ACTIVE: Advanced Cyber Threats response InitiatiVE. http://www.active.go.jp/en/
5. Li, F., Ho, G., Kuan, E., Niu, Y., Ballard, L., Thomas, K., Bursztein, E., Paxson, V.: Remedying web hijacking: notification effectiveness and webmaster comprehension. In: Proceedings of the International World Wide Web Conference (WWW) (2016)
6. Mavrommatis, N., Monrose, M.: All your iFRAMEs point to us. In: Proceedings of the USENIX Security Symposium (2008)
7. Eshete, B., Venkatakrishnan, V.N.: WebWinnow: leveraging exploit kit workflows to detect malicious URLs. In: Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY) (2014)
8. Kolbitsch, C., Livshits, B.: Rozzle: de-cloaking internet malware. In: Proceedings of the IEEE Symposium on Security and Privacy (SP) (2012)
9. Min, B., Varadharajan, V.: A simple and novel technique for counteracting exploit kits. In: Tian, J., Jing, J., Srivatsa, M. (eds.) SecureComm 2014. LNICSSITE, vol. 152, pp. 259–277. Springer, Cham (2015). doi:10.1007/978-3-319-23829-6_19
10. Lu, L., Perdisci, R., Lee, W.: SURF: detecting and measuring search poisoning categories and subject descriptors. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2011)
11. Borgolte, K., Kruegel, C., Vigna, G.: Delta: automatic identification of unknown web-based infection campaigns. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2013)
12. Li, Z., Alrwais, S., Wang, X., Alowaisheq, E.: Hunting the red fox online: understanding and detection of mass redirect-script injections. In: Proceedings of the IEEE Symposium on Security and Privacy (SP) (2014)
13. TripWire. http://www.tripwire.com/
14. Xie, G., Iliofotou, M., Karagiannis, T., Faloutsos, M., Jin, Y.: ReSurf: reconstructing web-surfing activity from network traffic. In: IFIP Networking Conference (2013)
15. Nelms, T., Perdisci, R., Antonakakis, M., Ahamad, M.: WebWitness: investigating, categorizing, and mitigating malware download paths. In: Proceedings of the USENIX Security Symposium (2015)
16. Akiyama, M., Yagi, T., Kadobayashi, Y., Hariu, T., Yamaguchi, S.: Client honeypot multiplication with high performance and precise detection. IEICE Trans. Inf. Syst. **E98–D**(4), 775–787 (2015)
17. Neasbitt, C., Perdisci, R., Li, K., Nelms, T.: ClickMiner: towards forensic reconstruction of user-browser interactions from network traces categories and subject descriptors. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2014)
18. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless monster: exploring the ecosystem of web-based device fingerprinting. In: Proceedings of the IEEE Symposium on Security and Privacy (SP) (2013)

19. Intel Security Inc.: Red kit an emerging exploit pack, January 2013. https://blogs.mcafee.com/mcafee-labs/red-kit-an-emrging-exploit-pack/
20. Lu, L., Yegneswaran, V., Porras, P., Lee, W.: BLADE: an attack-agnostic approach for preventing drive-by malware infections. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2010)
21. Cova, M., Kruegel, C., Vigna, G.: Detection and analysis of drive-by-download attack and javascript code. In: Proceedings of the International World Wide Web Conference (WWW) (2010)
22. Dell'Aera, A.: Thug. http://buffer.github.io/thug/
23. Curtsinger, C., Livshits, B., Zorn, B., Seifert, C.: ZOZZLE: fast and precise in-browser javascript malware detection. In: Proceedings of the USENIX Security Symposium (2011)
24. Canali, D., Cova, M., Vigna, G.: Prophiler: a fast filter for the large-scale detection of malicious web pages categories and subject descriptors. In: Proceedings of the International World Wide Web Conference (WWW) (2011)
25. Eshete, B., Villafiorita, A., Weldemariam, K.: BINSPECT: holistic analysis and detection. In: Proceedings of the International Conference on Security and Privacy in Communication Networks (SecureComm) (2013)
26. Zhang, J., Seifert, C., Stokes, J.W., Lee, W.: Arrow: generating signatures to detect drive-by downloads. In: Proceedings of the International World Wide Web Conference (WWW) (2011)
27. Stringhini, G., Kruegel, C., Vigna, G.: Shady paths: leveraging surfing crowds to detect malicious web pages categories and subject descriptors. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2013)
28. Mekky, H., Torres, R., Zhang, Z.L., Saha, S., Nucci, A.: Detecting malicious HTTP redirections using trees of user browsing activity. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM) (2014)