Fine-Grained Analysis of Compromised Websites with Redirection Graphs and JavaScript Traces*

Yuta TAKATA†, ‡, Nonmember, Mitsuaki AKIYAMA†, Takeshi YAGI†, Takeshi YADA†, ‡, and Shigeki GOTO††, Members

SUMMARY

An incident response organization such as a CSIRT contributes to preventing the spread of malware infection by analyzing compromised websites and sending abuse reports with detected URLs to webmasters. However, these abuse reports with only URLs are not sufficient to clean up the websites. In addition, it is difficult to analyze malicious websites across different client environments because these websites change behavior depending on a client environment. To expedite compromised website clean-up, it is important to provide fine-grained information such as malicious URL relations, the precise position of compromised web content, and the target range of client environments. In this paper, we propose a new method of constructing a redirection graph with context, such as which web content redirects to malicious websites. 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. The result shows that our system successfully identified malicious URL relations and compromised web content, and the number of URLs and the amount of web content to be analyzed were sufficient for incident responders by 15.0% and 0.8%, 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 analysis by our system would contribute to improving the daily work of incident responders.

key words: compromised website, drive-by download, redirection graph, program trace

1. Introduction

Attackers compromise popular websites and integrate them into a drive-by download attack scheme. According to a report [2], approximately 67% of malicious websites originated from compromised websites. One example is Darkleech attack which exploits vulnerable Apache modules. It has successfully compromised a large amount of websites; over 40,000 domains and IP addresses by May 2013, including 15,000 that month alone [3]. If high-reputation websites are compromised, even attentive users will be exposed to drive-by malware infections. An incident response organization such as a CSIRT (Computer Security Incident Response Team) tries to prevent the spread of malware infection by patrolling the Web and warning users. As part of the patrol activities, the organization re-analyzes compromised websites reported by users. They identify evidence of malicious websites and share this information [4]. 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 [5] and security products [6]. However, attackers build a redirection chain to evade analysis as well as to dynamically and selectively infect user’s clients with malware depending on the client environment [7]–[9]. Also, attackers can prevent any disclosure of malicious content by injecting only redirection code that leads to malicious websites, not exploit code or malware on compromised websites. 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, contributes to shortening the incident response time and increasing clean-up rates. Identifying the impact of a compromised website, such as the targeted client environments, 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 [6].

To identify the evidence and impact of compromise, we propose a new 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 access results while minimizing the number of environment profiles by designing them on the basis of known vulnerability information. To the best of 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 web content does a redirection originate, which URLs are associated with attacks, and which client environment is exposed to threats. This fine-grained analysis would provide practical directions to CSIRTs/security vendors for prompt incident response and expedite compromised website clean-up.

In summary, this paper makes the following contributions.

- Our system successfully identified malicious URL relations and the precise position of compromised web content. As a result, the number of URLs and the amount of web content to be analyzed were sufficient for incident responders by 15.0% and 0.8%, 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 organized as follows. In Sect. 2, we provide an overview of compromised website response and explain problems with 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 web compromise techniques 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. However, attackers use various anti-analysis techniques to evade the inspection and detection by defenders. In this section, we explain web compromise and anti-analysis techniques. Also, we provide an overview of compromised website response by CSIRTs/security vendors and explain problems with conventional methods.

2.1 Web Compromise Technique

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

2.1.1 HTML-Based Compromise

HTML-based compromises inject the redirection code of the iframe and script tags listed in Table 1. 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, it is easy to analyze them and find the redirection origin because these tags are directly written in an HTML file.

2.1.2 JavaScript-Based Compromise

JavaScript-based compromises execute code that dynamically generates the above-mentioned HTML tags using document.write, innerHTML, and appendChild, shown in Table 1 (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 on compromised websites. 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, a redirection chain, and browser fingerprinting, to protect their own malicious content against the inspections of CSIRTs/security vendors.

2.2.1 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.

2.2.2 Redirection Chain

Drive-by download attacks redirect users of a landing website (landing URL) to malicious websites (exploit URL) via

\(^1\)D. Edwards, “/packer/,” http://dean.edwards.name/packer
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)[7]. This redirection chain is composed of HTTP 3XX in addition to HTML tags and JavaScript as described in Sect. 2.1. Attackers can abuse compromised popular websites and web search results as landing URLs to lure unsuspecting users by constructing a redirection chain to malicious URLs[10]. Therefore, they only have to inject redirection code rather than exploit code for website compromises and can prevent any disclosure of malicious content. Multiple redirection stages also contribute to reducing the operation cost of attacks because compromised websites can be integrated into a different malware campaign by switching only the redirection URLs.

2.2.3 Browser Fingerprint

Browser fingerprinting, which is a method of profiling a client environment, e.g., a web browser and its 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[11].

2.3 Problems with 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 of 1–3 in Fig. 1. Such an 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 of 1⃝–5⃝ in Fig. 1[4]. However, in many cases, an abuse report with only URLs generated in this way is not enough to clean up compromised websites; therefore, webmasters cannot respond appropriately to such reports. Moreover, malicious websites cannot always be detected using analysis environments 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 to determine the impact of a compromised website, such as which versions of browsers 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.

Methods of detecting website compromises that compare original web content to compromised web content have been proposed[12], [13]. Furthermore, TripWire[14], 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.

To identify redirection chains, 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[15] and by leveraging some heuristics/features[16] 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. 1](Overview of compromised website response.)
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 the 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 redirection graph constructed by conventional methods 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 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. Also, 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 access 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).

3.1.1 Monitoring Behavior

Our system accesses websites and collects redirection and JavaScript 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 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(), setTimeout(), setInterval(), 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.
3.1.2 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 the top of Fig. 4, is structured.

- **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, for example, code executed while rendering websites, code executed by an event, e.g., onload() and onclick(), and code dynamically executed by eval(), setInterval(), and setTimeout(). 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, eval, setInterval, and setTimeout. In addition, this edge contains redirection methods to different URLs to identify JavaScript redirections.

- **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 from URL_B to URL_C is set to the DOM API function and HTML tag name, “document.write(iframe)”.

Figure 4 depicts a comparison of Fig. 2 between a redirection graph using the preceding proposed methods 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. However, none of the information mentioned above can be identified from the conventional redirection graph. This information is necessary for incident responders to conduct efficient and effective website forensics.

3.1.3 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 anti-virus. 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 DGA-domains and/or random strings. This phase also extracts malicious paths from identified malicious nodes to the node of the landing URL.

3.1.4 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 a 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 the number of 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
Table 2 Matrix of CVEs and Flash Player versions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.102.64</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.253</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.5.502.149</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7.700.169</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7.700.225</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7.700.252</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7.700.257</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.332</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.0.0.44</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.336</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.7.700.269</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.341</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.0.0.206</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.0.0.125</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.0.0.179</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.0.0.176</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.0.0.244</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.0.0.152</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.0.0.250</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.0.0.189</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.423</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.0.0.239</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.0.0.260</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.438</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.0.0.287</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.2.202.440</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.0.0.264</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.0.0.305</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.0.0.134</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

redirection origin because the domain of compromised websites is different from that of malicious websites [7]. Specifically, JS_1 is detected as a redirection origin by the difference between URL_B’s domain and URL_C’s domain.

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 the behavior of a website being changed 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).

3.2.1 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 (Fig. 5). If we can predict potential targeted vulnerabilities in websites, the number can be further decreased by filtering out the corresponding columns of the matrix. For example, we show a matrix of the matching of known vulnerability information obtained from CVE Details and affected versions of Adobe Flash Player in Table 2. We further decreased the elements of the matrix by utilizing the vulnerability information of exploit kits from 2014–2015 obtained from contagio. In Table 2, the versions of Adobe Flash Player were aggregated from 251 to 31. Note that oldest version is selected from aggregated versions.

---

3.2.2 Matching Results

Our system compares crawl results of various environments and detects differences in the 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 for constructing a RCG 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\(^1\), 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 [8], [9]. Therefore, we collected vulnerability information on these plugins from CVE Details and contagio, mentioned in the previous subsection. The numbers of aggregated versions of JRE, PDF, and Flash are listed in Table 3. The rows of Table 3 represent the number of plugins for the vulnerability information of exploit kits from 2014–2015, exploit kits from 2011–2013, and the number of official installers we found manually. Table 3 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 (see Sect. 6.4).

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Number of plugin versions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploit kits from 2014–2015</td>
<td>JRE</td>
</tr>
<tr>
<td>Exploit kits from 2011–2013</td>
<td>37</td>
</tr>
<tr>
<td>Official installer</td>
<td>193</td>
</tr>
<tr>
<td>Environment profile reduction</td>
<td>142</td>
</tr>
</tbody>
</table>

4. Experiment and Evaluation

We evaluated the effectiveness and performance of our system using the HTTP communication data of the 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.

4.1 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.

4.1.1 High-Interaction Honeyclient

We used HTTP communication data of websites that were preliminary detected drive-by-download attacks by a high-interaction honeyclient [17]. Exploit URLs and malware distribution URLs detected by the honeyclient were also used as a blacklist in the identifying malicious node phase.

4.1.2 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 [18], 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.

\(^1\)Gargoyle Software Inc., http://htmlunit.sourceforge.net/
4.1.3 Our System

Our system, which is the extended HtmlUnit described in Sect. 3.3, analyzes web content 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. Note that this detection method of browser fingerprinting is straightforward and limited to sophisticated browser fingerprinting such as side-channel inference [19].

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

4.2.1 Constructing Redirection Graph

Our objective is to identify information of compromised websites at a content-level in addition to an URL-level. Since compromised web content, i.e., a redirection origin, can be identified from a redirection path, we evaluated how many nodes (URLs) can be connected with the proposed method compared with conventional methods. In other words, false positives and false negatives in this evaluation are that edges are not connected correctly and that there are no edges to be connected, respectively.

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) [15] and the heuristic-based method (HEU) [16]. As a result, the number of nodes identified by only PRO were 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. 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.

4.2.2 Redirection Graph without Malicious Path

We manually analyzed the causes of the incomplete redirection graphs that did not contain malicious URLs, i.e., malicious nodes. Table 4 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 [19]. Attackers can also maliciously access a file system and check the presence of security vendor products through Internet Explorer by abusing an information 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 4 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

<table>
<thead>
<tr>
<th>Category</th>
<th>#graphs</th>
<th>Reason</th>
<th>Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophisticated browser fingerprinting</td>
<td>231</td>
<td>Anti-virus detection and browser-specific JavaScript property</td>
<td>Analyze it with a real browser</td>
</tr>
<tr>
<td>URLs with DGA-domains and/or random strings</td>
<td>165</td>
<td>Lack of approximate matching and suspicious URL detection ability</td>
<td>Improve accuracy of algorithm</td>
</tr>
<tr>
<td>Emulator evasion</td>
<td>122</td>
<td>Defect of DOM implementation in HtmlUnit</td>
<td>Fix it</td>
</tr>
<tr>
<td>Time-dependent redirection</td>
<td>57</td>
<td>Past crawl data</td>
<td>Analyze it immediately after detection</td>
</tr>
<tr>
<td>VBScript</td>
<td>4</td>
<td>Unsupported in HtmlUnit</td>
<td>Analyze it with real browser</td>
</tr>
</tbody>
</table>
Table 5 Analysis of client-dependent redirection with browser fingerprinting.

<table>
<thead>
<tr>
<th>Detected:Suspicous:Unknown</th>
<th>#crawls</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0:1</td>
<td>359</td>
<td>Client-dependent redirection with browser fingerprinting</td>
</tr>
<tr>
<td>0:1:1</td>
<td>117</td>
<td>Client-dependent redirection with browser fingerprinting</td>
</tr>
<tr>
<td>1:1:1</td>
<td>149</td>
<td>Client-dependent redirection with browser fingerprinting</td>
</tr>
<tr>
<td>0:0:1</td>
<td>209</td>
<td>Emulator evasion, time-dependent redirection, etc. (see Table 4)</td>
</tr>
<tr>
<td>1:1:0</td>
<td>226</td>
<td>Malicious websites using URLs with DGA-domains and/or random strings</td>
</tr>
<tr>
<td>0:1:0</td>
<td>91</td>
<td>Malicious websites using URLs with DGA-domains and/or random strings</td>
</tr>
<tr>
<td>1:0:0</td>
<td>370</td>
<td>Simple malicious websites</td>
</tr>
</tbody>
</table>

and suspicious URL detection ability, time-dependent redirections, and use of VBScript.

4.2.3 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 (5%) 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. 5.2.

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 3, 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 5). 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 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. 5.3).

4.4 Performance Overhead

We evaluated the total time and the average time taken to analyzing the 2,058 websites with our system. The results
indicated that the time costs were 685,773 sec and 333 sec, respectively. Since 90% of benign website crawlings done by the high-interaction honeyclient that detected compromised websites used in this experiment finished within 154 sec [17], 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 sec on average and these of each website were nearly equal. Therefore, the minimizing of environment profiles, i.e., JRE, PDF, and Flash in Table 3, can reduce \( \frac{142}{193} = 73.6\% \), \( \frac{79}{103} = 76.7\% \), and \( \frac{188}{251} = 74.9\% \) analysis time, respectively. From the above, our system is appropriate for frequent re-analysis of websites because the browser emulator does not require extra analysis time, e.g., 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 Compromised Websites for Malware Campaign

We first show an example of malicious paths constructed from crawl results, which contained the leaf URL of a .jar file extension (Fig. 8). The redirection started from a script tag in the landing URL to an applet tag that points to the leaf URL via a location, meta tag, HTTP302, and iframe tag, as shown in Fig. 8. Since our system cannot execute a Java archive file, it stopped at the URL of a .jar file extension. These above features, characteristic lexical features of URLs, and facts of data observed from Oct. – Nov. 2012 suggest that the landing website was injected with a script tag that redirects to a malicious website built using the Styx exploit kit [20]. Other characteristics of exploit kits appear in HTML tags and JavaScript code in addition to the data observation period and the lexical features of URLs mentioned above³.⁴. Since many attackers pervasively use such exploit kits for malware campaign, the capability to analyze them is important. To show the validity of our method against exploit kits, we investigated signatures and security vendor reports for other malicious paths based on these characteristics. As a result, we have also identified malware campaigns with other exploit kits such as Blackhole, RedKit, Flash Pack, RIG, Nuclear, and Angler.

5.2 Sophisticated Semantic Gap

5.2.1 Obfuscated Semantic Gap Edge

We depict an example of malicious paths that contained dynamically generated code and a semantic gap in Fig. 9. The semantic gap was caused by DOM API code (JS₇) in obfuscated code (JS₆) 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.

5.2.2 Multiple Compromised Web Content

We show an example of a part of RCGs constructed from crawl results, which contain two or more differences in the number of identified URLs between PRO and REF/HEU in Sect. 4.2 (Fig. 10). Compromised web content in Fig. 10 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.

---


5.3 Client-Dependent Redirection with Browser Fingerprinting

The JS 8 of Fig. 9 changed the destination URL by executing the browser fingerprinting code that gets the version of the PDF plugin in Fig. 11. We analyzed the code using our system that emulated 23 individual versions of a PDF based on Table 3 because the code was observed in 2012. As a result, the versions shown in Table 6 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) [21]. CVE-2010-0188 exists in Adobe Reader/Acrobat 8.X before 8.2.1 and 9.X before 9.3.1, and the code has also been implemented to redirect to the URL of DOMAIN6 when a PDF version that has the vulnerability is used.

6. Discussion

6.1 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 incomplete redirection graphs without malicious paths in Sect. 4.2 were also one of the factors preventing the construction of graphs. Naturally, in the case of an incomplete redirection graph, an incident responder must analyze the website in conventional operation. We admit all these issues can affect the performance of our system. However, these issues are not specific to our system and affect all real browsers and browser emulators in some degree. It is also difficult to automatically identify whether a redirection graph is incomplete or not. More importantly, our system could identify the evidence and impact of 71.9% of compromised websites under the limitations. To maximize the disclosure of suspicious/malicious content and suggest the possibility of an incomplete redirection graph, we must combine our system with other techniques such as machine learning discussed in Sect. 7.2.

6.2 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 reduce the response time required for clean-up done by webmasters, such as in an existing user study [6].

An incident responder generally determines whether a website is malicious by identifying URLs that should be analyzed based on the redirection graph structure and analyzing web content of these URLs [16]. Therefore, instead of a user study on webmasters, we calculated the URL reduction rate (URR) and the content reduction rate (CRR), which were inspired from the evaluation method of the existing research [18], to evaluate how our system can contribute to the work of incident responders. The URR is how many URLs our method can filter out by extracting malicious redirection paths from the entire redirection graph of each crawling. The CRR is how much web content on compromised websites would not be analyzed by extracting compromised
web content using our method. These rates of all n websites were obtained with the following formulas.

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

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

As a result, our method could reduce 85.0% of URLs (23 URLs on average). Furthermore, the CRR was 99.2% (16,568 bytes on average) on the basis of the value in a Content-Length header, i.e., the number of URLs and the amount of web content to be analyzed were sufficient for incident responders by 15.0% and 0.8%, 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 an .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.

6.3 Immediate Online Crawling After Detection

We evaluated our system using data of compromised websites that were preliminarily detected in Sect. 4. In this subsection, we evaluated the effectiveness of our system by crawling compromised websites on the live Internet immediately after a high-interaction honeyclient detected the websites. Our system emulated the same client environment as the high-interaction honeyclient and crawled ten compromised websites that were detected during one month, July 2016. As a result, our system identified malicious paths from two websites that contained malicious Flash files. The other eight websites were not identified due to empty content (probably server-side cloaking) and advertisements (probably malvertising). These results show that our system can successfully identify compromised web content even for online crawlings. However, it is also important to leverage forensic artifacts that have been already detected to minimize the effects of dynamic web content, as described in Sect. 4.

6.4 Multiple Analysis Using Various User-Agents

We focused on browser plugins (JRE, PDF, and Flash) and evaluated whether our system can identify client-dependent redirections and the target range of client environments in Sect. 4.3. In this subsection, we expanded our multi-client environment to user-agents and further investigated the impact of compromised websites, i.e., whether malicious websites change behavior depending on the user-agent.

Our system emulated nine user-agents, Internet Explorer (IE) 6 and 7 on Windows XP, IE 8, 9, 10, and 11, Google Chrome (Chrome), Mozilla Firefox (Firefox) on Windows 7, and Firefox on Linux. In this experiment, we evaluated all 2,058 compromised websites regardless of the use of browser fingerprinting because the number of user-agents is lower than the number of plugins.

We show the results of multiple analysis using various user-agents in Table 7. Only 158 (7.7%) websites contained detected and/or suspicious crawl results at the same time as unknown crawl results. We found the browser fingerprinting code in Figs. 12 and 13 through manual inspection of these websites. The code in Fig. 12 determines whether to redirect clients to the following URL of DOMAINDOMAIN10 depending on the user-agent information collected from BrowserDetect object. This code also changes behavior by identifying clients that access the website multiple times using a cookie. Another example (Fig. 13) determines whether to redirect clients to the URL of DOMAINDOMAIN11 by ex-

<table>
<thead>
<tr>
<th>Detected</th>
<th>Suspicious</th>
<th>Unknown</th>
<th>#crawls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0:1</td>
<td>0</td>
<td>1</td>
<td>147</td>
</tr>
<tr>
<td>0:1:1</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1:1:1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0:0:1</td>
<td>1</td>
<td>0</td>
<td>323</td>
</tr>
<tr>
<td>1:0:0</td>
<td>0</td>
<td>1</td>
<td>71</td>
</tr>
<tr>
<td>0:1:0</td>
<td>1</td>
<td>0</td>
<td>119</td>
</tr>
<tr>
<td>1:1:0</td>
<td>0</td>
<td>1</td>
<td>1,387</td>
</tr>
</tbody>
</table>

Fig. 12 Browser fingerprinting code using user-agent information.

Fig. 13 Indirect browser fingerprinting code.
The methods of detecting website compromises are generally used for comparing original and compromised web content. For example, a comparison method [12] using HTML files as original content and a comparison method [13] using well-known libraries and frameworks of JavaScript as original content have been proposed. Moreover, TripWire [14] 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.

7.2 Detecting Malicious Websites

Over the past few years, many researchers have proposed methods of detecting drive-by downloads. A honeyclient is a decoy client system for crawling and detecting malicious websites. It is classified as high-interaction or low-interaction. A high-interaction honeyclient [17], [22] crawls websites with a vulnerable real browser and detects malware downloads by monitoring unintended processes and file system accesses, whereas a low-interaction honeyclient [23], [24] crawls websites with a browser emulator and detects malicious behaviors by signature matching and machine learning. Also, learning-based methods of detecting malicious web content have been proposed and leveraged features from HTML, JavaScript, and URL [25], [26]. 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 these methods can detect malicious websites with high accuracy. Similarly to our method, methods of analyzing a redirection graph on malicious websites leverage a diverse dataset of redirection graphs and co-occurring URLs in graphs [27], [28]. Others [16], [29] 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) because of the coarse-grained redirection information.

7.3 Website Analysis Using Multiple Clients

Wang et al. [11] examined the dynamics of cloaking and uncovered the lifetime of cloaked websites using a system designed to crawl search results three times with different user-agents and referers. They measured and characterized the prevalence of cloaking on different search engines and search terms in addition to user-agent cloaking and referer cloaking. Invernizzi et al. [30] developed an anti-cloaking system that detects when a web server returns divergent content to two or more distinct browsers. This system fetches content via multiple browser profiles as well as network vantage points to trigger any cloaking logic and distinguish benign cloaking from blackhat cloaking. These systems focus on cloaking techniques and perform a complementary role to our system.

8. Conclusion

We proposed a new method of constructing a new fine-grained redirection graph to identify the evidence and impact of compromise. Our system with the proposed method analyzes a website in a multi-client environment while minimizing the number of environment profiles. Our evaluation was performed with compromised website data obtained during a four-year period. The result showed that our system could successfully identify the precise position of compromised web content and targeted client environments on 71.9% of websites although there were websites that our system cannot construct redirection graphs due to
the browser emulator evasion. We also showed that it could effectively identify an exploit kit and a vulnerability that has been used in malicious websites by leveraging the evidence and impact of compromise. Our system can contribute to improving the daily work of CSIRTs/security vendors and expediting compromised website clean-up done by webmasters.

References


Yuta Takata received his B.E. and M.E. degrees in computer science and engineering from Waseda University, Japan in 2011 and 2013. He is currently a Ph.D. student in the Department of Computer Science and Engineering, Waseda University. Since joining the Nippon Telegraph and Telephone Corporation (NTT) in 2013, he has been engaged in research and development on network security, especially honeypot/honeypot and malicious code analysis. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.
Mitsuaki Akiyama received his M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology, Japan in 2007 and 2013. Since joining the Nippon Telegraph and Telephone Corporation (NTT) in 2007, he has been engaged in research and development on network security, especially honeypot and malware analysis. He is now with the Cyber Security Project of NTT Secure Platform Laboratories.

Takeshi Yagi received his B.E. degree in electrical and electronic engineering and his M.E. degree in science and technology from Chiba University, Japan in 2000 and 2002. He also received his Ph.D. degree in information science and technology from Osaka University, Osaka, Japan in 2013. Since joining the Nippon Telegraph and Telephone Corporation (NTT) in 2002, he has been engaged in research and design on network architecture, traffic engineering, and his current research interests include network security, web security, honeypots, security-data analysis based on machine learning, and security intelligence technologies such as URL/domain/IP blacklisting and reputation. He is now a senior research engineer in the Cyber Security Project of NTT Secure Platform Laboratories. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Institute of Electrical Engineers of Japan (IEEJ).

Takeshi Yada is a senior research engineer and supervisor in the Cyber Security Project, NTT Secure Platform Laboratories. He received his M.S. degree in engineering from the Tokyo Institute of Technology, Japan, in 1991. Since joining NTT in 1991, he has been engaged in research and development on network architecture, measurement and inference for network traffic, and network management. His current research interests include network security. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE), the Operations Research Society of Japan, and the Japan Statistical Society.

Shigeki Goto is a professor in the Department of Computer Science and Engineering, Waseda University, Japan. He received his B.S. and M.S. in mathematics from the University of Tokyo. Prior to becoming a professor at Waseda University, he worked for NTT for many years. He also earned his Ph.D. in information engineering from the University of Tokyo. He is the president of JPNIC. He is a member of ACM and IEEE, and he was a trustee of the Internet Society from 1994 to 1997.