YALIH, Yet Another Low Interaction Honeyclient

Masood Mansoori, Ian Welch, Qiang Fu
School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
masood.mansoori@ecs.vuw.ac.nz, ian.welch@ecs.vuw.ac.nz, qiang.fu@ecs.vuw.ac.nz

Abstract
Low-interaction honeypots employ static detection techniques such as signatures, heuristic or anomaly detection in the identification of malicious websites. They are associated with low detection rate and failure to identify zero-day and obfuscated attacks. This paper presents a low-interaction client honeypot that employs multiple signature detection engines in combination with de-obfuscation and de-minification of JavaScript code to improve the detection of attack signatures. Pattern matching in the process of identifying the static malicious code characteristics through using regular expressions, provides additional layer of detection. YALIH can achieve low false positive and false negative rate while significantly reducing scanning time and required hardware resources compared to a high interaction client honeypot. YALIH’s virtual browser can handle cookies, redirection and mimic popular browser headers and imitate referrer information. Our experiments with real-world malicious websites demonstrate that similar to Web Spam, malicious websites utilize referrer tracking and cloaking techniques to deliver malicious content to selected users visiting the target domain from specific referrer websites.

Keywords: Client Honeypot, Signature Detection, Malicious Website, Regular Expression, Pattern Matching

1 Introduction
Client honeypots are security tools used to identify websites that host malicious content. Visitors to the websites are attacked when a website’s active content exploits vulnerabilities within the browser or the browser’s plug-ins. This type of exploit is termed a drive-by-download attack and it is implemented using scripting languages supported by all popular browsers. The scripting languages allow website developers to provide highly-responsive and interactive experiences to the end users, however, they also provide a cross-platform for attackers to target and deliver an exploit to a large number of potential targets. For instance the JavaScript programming language is the most widely used scripting language supported by all operating systems and browsers. A drive-by exploit targeting a vulnerability in the JavaScript engine of the browser can potentially infect every user visiting the malicious website regardless of the browser type or operating system.

Client honeypots are able to detect malicious websites using various techniques, depending on the type and interaction level of the honeypot used. High-interaction client honeypots operate on the principle of monitoring system state changes during a website visit using real browsers installed on the host’s system. Websites are classified as malicious if the browser accesses or attempts to modify monitored security-sensitive directory and files. Low-interaction client honeypots on the other hand rely on simulated browsers to mimic the behaviour of a user’s client and fetch the website content. A common approach is to scan the fetched content and match it against known signatures of exploits. A limitation of this signature-based detection approach is it will fail to detect zero-day attacks – an attack for which a signature does not exist in the signature database – or attacks which are derived from existing attacks. However, low-interaction client honeypots when compared with high-interaction client honeypots offer advantages of faster scanning time; easier customizability; and, detection of time-bombs and user input-driven attacks.

We have developed a low-interaction client honeypot that improves upon previous designs, mainly in its visitor and detection engine components. Its simulated browser allows manual or automatic setting of various attributes such as browser header, user-agent information and referrer setting that are targeted by malicious websites. Other visitor-component capabilities such as automatic redirection, cookies and session management make the client honeypot less susceptible to detection. The analysis engine de-obfuscates extracted JavaScript scripts and employs two signature-based antivirus engines that provide a very low false positive rate of detection for known attacks. Our analysis engine includes a module that allows creation of rules in ASCII, Hexadecimal or Regular Expression. This allows us to extend the signature databases by adding rules to detect a specific attack or group of attacks using suspicious static characteristics of malicious websites.

We tested the detection accuracy and performance of our client honeypot using a large dataset of non-malicious websites and a representative data set of malicious websites. This showed that we could achieve a very low false positive rate with an acceptable false negative rate especially when compared with similar tools.

This paper makes the following three contributions:
- Design and implementation of a low-interaction client honeypot using de-obfuscation of JavaScript, pattern matching through regular expressions, and multi signature-based detection techniques.
- Evaluating the effectiveness and detection accuracy of YALIH’s honeypot against other low interaction client honeypots.
• Evaluating the referrer tracking and cloaking features of malicious websites through real-world experiments.

The rest of paper is organized as follows. We introduce related work in Section 2. Our honeypot’s design is given in Section 3. In section 4 and 5, the experimental setup, detection accuracy and performance of the honeypot are provided and results and analysis of real-world referrer cloaking experiments are discussed respectively.

2 Background and Related Work

Low-interaction client honeypots generally use signatures to identify potentially malicious webpages. Signature-based systems in general provide a fast scanning speed with very low false positive and high false negative rate of detection. The high false negative rate is due to two main issues: (1) an inability to identify zero-day attacks where signatures do not yet exist; and, (2) an inability to deal with variations of existing attacks (e.g. obfuscated attacks); and their limited reliance on a single analysis engine in favour of lower detection time. We discuss each of these issues in the following Sections.

2.1 Obfuscated Attacks

The false negative detection rate of client honeypots increases significantly with obfuscated attacks that exploit the numerous obfuscation techniques scripting languages such as JavaScript provide (e.g. Base64, simple substitution, string splitting) (Xu et al., 2012). Obfuscation allows attackers to bypass detection and deliver exploits for which a signature already exists. Signature-based detection engines that do not de-obfuscate the script upon analysis will suffer from false negatives regardless of the number of attack signature databases in use. Several honeypots address this problem by dynamic execution of JavaScript content. For example, PhoneyC (Nazario, 2009) executes JavaScript in a limited environment and performs analysis by a vulnerability module architecture that looks for exploit activity against a vulnerable method within the browser. Thug honeypot (Dellaera, 2013) performs rendering of a website’s JavaScript within Google’s V8 JavaScript engine and searches for browser and plugin vulnerability modules and exploit shellcodes. Similar to the Thug, MonkeyWrench (Büsch er et al., 2010) executes JavaScript within Mozilla’s JavaScript engine, SpiderMonkey, and monitors each call for buffer overflows and occurrence of shellcode. It also detects potentially malicious websites by identifying the use of zero-sized inline iframes and scans the fetched website using G-Data Linux antivirus engine. YALIH does not add improvement in de-obfuscation of JavaScript compared to other honeypots but offers improved analysis on JavaScript codes de-obfuscated by its analysis engine.

2.2 Single Analysis Engine

Honeypots mostly rely on a single analysis engine for attack detection. Monkey-Spider (Ikinci et al., 2008), and Spy-Bye (Provos) utilize ClamAV antivirus signature engine while HoneyC (Seifert et al., 2007) makes use of Snort intrusion detection system rules. The reliability on a single analysis engine makes them susceptible to unknown number of false negatives that relate directly to the quality and freshness of attack signatures. Honeyware (Alosefer and Rana, 2010) integrates five antivirus signatures engines on a web-based interface but performs no de-obfuscation or de-minification of scripts and no static analysis of malicious characteristics are performed on the HTML components. Their system, using only antivirus engines, manages to detect a higher number of attacks than the high interaction client honeypot Capture-HPC but takes a longer time to scan each website and cannot detect zero-day attacks. YALIH aims to provide both a high rate of detection and a very low scanning time using multiple analysis engine and low resources. It deliberately does not address the problem of zero-day attacks.

3 Honeyclient Design

Client honeypots are generally comprised of three main components that manage the entire process of website retrieval and analysis (Seifert et al., 2007). These components are: (1) URL Collector (Queueer); (2) Visitor; and, (3) Analysis Engine. The following discussion about YALIH is structured in terms of a discussion of each of these components. YALIH has a modular design to allow additional URL databases and analysis modules to be added in the future. Figure 1 shows an overall design of the YALIH honeypot.

Figure 1: Overall design of the YALIH honeypot

3.1 URL Collector

The URL Collector module is responsible for gathering the hyperlinks of potential malicious websites to be visited by the honeypot. Depending upon the implementation, these hyperlinks can be gathered from single or multiple sources. For example, HoneyC and Monkey-Spider gather hyperlinks from search engines while Monkey-Spider also employs crawlers (Ikinci et al., 2008). A problem with using web crawlers is that it
increases the chances that a honeyclient may be detected by a malicious web server. Crawlers may be detected because they initiate a high number of connections and fetch many URLs in a very short period of time. This behaviour can be detected by a malicious server through IP-Tracking and may result in throttling the connection or serving the honeyclient with benign contents.

YALIH uses the Mechanize visitor module (Lee, 2010)(Lee, 2010)(Lee, 2010)(Lee, 2010) (see Section 3.2 for more detail) that provides built-in functions to crawl and extract links from the visited websites to a depth defined by the user. However, the crawling functionality is not used because of the problem identified above. Only the HTTP client functionality of Mechanize is used.

URL Collector module is responsible for gathering the hyperlinks of potential malicious websites to be visited by the honeyclient. Depending upon the implementation these hyperlinks are gathered from single or multiple sources. HoneyC and Monkey-Spider for instance use results from search engines while Monkey-Spider also employs crawlers (Ikinci et al., 2008). The Mechanize visitor module (Lee, 2010)(Lee, 2010)(Lee, 2010)(Lee, 2010) – described in Section 3.2 - provides built-in functions to crawl and extract links from the visited websites to depth defined by the user. URL collection and queuing in YALIH can be achieved in several different ways:

1. **Direct Feed** - A user may input a single URL directly, or use a file containing a list of URLs to visit. The List is initially inspected to remove duplication.

2. **Search Engine API Integration** - Search engine APIs are possibly the most widely used technique for URL collection used by client honeypots. Monkey-Spider for instance utilizes Bing and Yahoo APIs for URL collection. YALIH incorporates the Bing search engine but Bing only allows a limited number of queries for a particular developer in a certain period of time (i.e. 2000 queries per month). It should also be expected that search engines return few malicious websites as various strict security filtering are applied to their returned results.

3. **Spam and Phishing** - Spam emails are a well-known source of malicious contents, luring users into browsing websites aimed at phishing scams or containing malware and drive-by download attacks. YALIH’s unique spam and phishing component integrates URL harvesting from email addresses using IMAP protocol. Provided with email credentials and supporting IMAP protocol, the honeypot fetches the contents from the user’s Inbox and Spam folder - if available – and excludes the attachments. To minimize the traffic, “Sent folder” and user created folders are omitted. The consensus is that users are wary of the nature of emails filtered and placed on their IMAP generated folders and would not send out emails containing malicious contents consciously. The fetched emails are then parsed for URLs, duplicates are removed and sent to the visitor agent for content retrieval.

4. **Malicious Website Database** - To assist researchers with exploit code collection and illustrate the effectiveness of the honeypot, Blacklisted URL collection module downloads suspected malicious website lists from three databases and analyses them accordingly. These databases are updated constantly and contain links to websites serving drive-by exploits, malicious executables and malicious Portable Document Format (pdf).

3.2 **Visitor Agent**

The visitor component in a low-interaction honeyclient is a virtual browser which mimics the functionalities of a real browser; fetching the contents of the potential malicious website and storing it for analysis engine.

YALIH’s Visitor agent is based on Mechanize module, which is available for various programming languages and provides a wide range of capabilities.

The Mechanize visitor agent allows a user to set different personalities in the form of browser headers while communicating with a malicious webserver. Browsers’ headers may represent various browser characteristics such as type, version, underlying operating system, language and extensions. Exploit code targeting a specific vulnerability available for a particular browser type or version may not execute on a different browser unless it targets a common rendering engine or extension such as Java Runtime Engine. Multiple exploits targeting a specific browser and operating system are delivered to a user during a single visit to increase the probability of infection. Browser Exploit Packs (BEP) which are widely available online, facilitate and automate this feature. Examples of such tools are CrimePack, Phoenix Exploit Kit, Eleonore exploit pack, IcePack and The Blackhole exploit pack.

Mechanize module in YALIH honeypot is configured to provide various capabilities to minimize detection of virtual browsers through cookies, redirection and session handling. The Referrer setting was also tuned to minimize exploit delivery blocking based on the referrer data. Browser exploit Kits such as Blackhole BEP allow an attacker to block the delivery of exploit to a visitor if the referrer does not match a predefined set of referrers or matches a set of referrer domains for which there is the suspicion of existence of a honeyclient. YALIH allows the user to automatically set the referrer to predefined strings of: 1) A particular search engine results 2) Top domain of the visiting URL or manually set the referrer string. Figure 2 shows a snapshot of the referrer blocking features of Blackhole browser exploit pack. The visited and fetched website contents are saved on the local disk for analysis.
Referrer and IP blocking features of Figure java engine which de is then scanned by multiple antivirus and SandBox client honeypots. Once the output is generated, a certain character such as Malicious Pattern script is evaluated. AVG Alvarez, 2008-2009, AVG, which is a free antivirus engine that searches for malicious patterns in JavaScript files. As they did not provide signatures for Linux operating systems, they were tested but removed to reduce the scanning time, as they did not provide significant detection improvement, considering the delay they introduced. Modular design of our honeypot however allows several scanning engines and specifically antivirus engines to be added with nearly two lines of code.

Multiple antivirus engines comprise a part of the overall analysis engine of YALIH and are complemented by Yara (Alvarez, 2008). Yara is a malware identification and classification library designed to detect and classify malware samples using binary, textual and regular expression pattern matching. It is utilized in systems such as VirusTotal and Cuckoo SandBox (Inoue et al., 2008).

Malicious Pattern Detection - Malicious websites exhibit certain characteristics that set them apart from legitimate and benign websites. Although some of these characteristics can be used in the design of complex websites, their presence can be an indication of the maliciousness of the website. Some of these characteristics are hidden, zero or small sized iframes, availability of obfuscated scripts or shellcode patterns in JavaScript files. Thug client honeypot which has had a development timeline as our client honeypot also integrates Yara but only performs scanning to detect signatures of malicious websites infected by Browser Exploit Packs (BEPs).

Utilization of Regular expressions allows the system to detect any variations of these characteristics. As an example the following rule is:

```javascript
function Check() {
    var s = "AAAA",
        while (s.length < 768 * 768) s=s+s;
    var obj = new ActiveXObject("SWCtl.SWCtl"); // [233C1507-6A77-46A4-9443-F871F943D238]
    obj.ShockwaveVersion(s);
}
```

**3.3 Analysis Engine**

The analysis engines in low-interaction client honeypots generally identify malicious content through a single or multiple detection techniques (e.g. Antivirus Engine, Snort Rules). The analysis engine on YALIH also relies on several detection methods:

1. **De-Obfuscation and De-Minification of JavaScript** - Scripting languages provide numerous obfuscation techniques that allow attackers to encode the exploit code and bypass signature-based detection, even attacks for which signatures are available. The obfuscation can be in forms of base64 encoding, simple misplacement of characters or strong encryption algorithms (Feinstein and Peck, 2007, Heyman, 2007, Howard, 2010, Nazario, 2009). Minification, while being mostly used to compress the size of the script; can also be utilized to bypass simple signature based detection tools (Nazario, 2009).

The emphasis on JavaScript attack detection, besides obfuscation capabilities, is due to the fact that browser Java engine vulnerabilities account for a vast number of exploits in the last few years and provides a multi-platform environment for malicious code to be executed on nearly all browsers and operating systems. JavaScript replaced Adobe Flash as the most targeted platform and accounted for 50% of all the targeted exploits in 2012 (Kaspersky, 2013).

YALIH parses, extracts and downloads the Java script files (.js) embedded within the visiting websites. It is assumed that malicious websites avoid placing the exploit code directly within the HTML code to avoid detection. The downloaded .js file is rendered by the Rhino Java engine which de-minifies and de-obfuscates the JavaScript scripts encoded using popular techniques (i.e. base64, base62, numeric (base10), high ASCII (base 95), string substitution and/or concatenation) (see Figure 3). Once the output is generated, it is written to a file that is then scanned by multiple antivirus and signature/pattern matching tools.
designed to detect hidden iframes or Styles within the HTML code of the website. Any single or combination of these can be used depending on the level of detection required by the user (see Figure 4).

\[
\text{styl} = \langle iframe, \text{?visibility} = \text{?hidden}, \text{?height} \rangle
\]

Figure 4: Example of Regular Expressions to detect hidden or 0-2 sized iframes

Users are also able to develop their own signatures in Yara that match an exploit pattern either in ASCII, Hexadecimal strings or Regular Expression (RE, Regex) format. For instance, a signature for the vulnerability “Blackice Cover Page SDK insecure method DownloadImageFileURL() exploit” can be single or combination of these in a website content (see Figure 5). These signatures must be unique to this exploit to avoid raising false positive alerts. Yara rules are easier to create than Snort rules of HoneyC and vulnerability modules of Thug and PhoneyC, as only a set of strings and a Boolean expression to determine its logic are required. These signatures are language independent and can detect signatures in malicious code written in any client-side scripting language (e.g. JavaScript, VBScript).

Rules:

\[
\text{\$1 = 'vbscript' fullword}
\text{\$2 = 'BiDiB.dll' nocase}
\text{\$3 = 'DownloadImageFileURL' fullword nocase}
\text{condition:}
\text{two of them}
\]

Figure 5: Example of user created string pattern rule to detect a specific attack (Exploit-DB, 2011)

4 Detection Accuracy and Performance

The detection accuracy of our designed honeypot is measured by experiments with two control sets of data; an approach similar to (Cova and Kruegel, 2010).

- A “clean dataset” containing 10,000 websites gathered from Alexa’s top 1 Million most visited websites.
- A “malicious dataset” containing 110 exploits gathered from

The experiment focused on several aspects of the detection, including:

- The number of incorrectly identified websites as malicious (False Positive)
- The number of malicious websites not detected (False Negative)
- The speed of detection

The initial step was to determine the false positive rate of YALIH’s individual analysis engines against our clean dataset. YALIH’s antivirus detection engines were updated to the latest version and pattern matching rules were created for Yara Module. Pattern matching using regular expression involved rules determining the static characteristics of a website and its embedded scripts to identify a malicious website. The integrated rules were:

1. Java Shellcode Pattern – Long set of integer, special character or string can be an indication of shellcode presence (Egele et al., 2009).
2. Java Obfuscation identified by “eval”, “unsafe” functions – Referred to as the “dangerous functions”, they can be used to encode and decode shell strings code to obfuscate an exploit (Hou et al., 2010).
3. Java Obfuscation using base64 encoding – JavaScript malicious content can be encode using base64 encoding to obfuscate and avoid detection (Nazario, 2009).
4. Decoded or hidden iframe – iframes are a popular method of delivering malicious contents to a user browser through redirecting part of the browser to a malicious website (Hou et al., 2010, Lam et al., 2013, Provos et al., 2008). This is generally achieved through loading a malicious website in an iframe with size zero or a small size integer. Any webpage with an iframe tag with one or more properties such as “Visibility: Hidden”, “Display: None” or Width and Height of “0” are considered suspicious. This rule was however removed in later experiments as it produced high number of false positives. It was observed that hidden and small-sized iframes were widely used in large number of websites in our clean dataset. Number of zero and small-sized iframes was also relatively similar between our “clean dataset” (i.e. 2673 small-size iframes) and 10,000 randomly selected website from our “suspicous dataset” used in our real-world experiment (i.e. 2458 small-sized iframes) (see section 5).

Subsequent analysis for hidden iframes in both clean and suspicious dataset revealed that they were widely used for seamless display of content and bypassing same-origin security policy for advertisement display purposes and were not necessarily a reliable indication of a maliciousness of a website (Auberger, 2011). YALIH currently extracts and retrieves .js scripts which may be embedded within hidden or small-sized iframes but the functionality to extract other types of contents (i.e html links) were disabled based on our observation of follow-up links.

Scanning was performed on the de-obfuscated and de-minified content, retrieved and saved on local disk by Yalih’s visitor agent. Signature-based Antivirus engines achieved a 0% false positive rate on the clean dataset. A false positive (FP) occurs when a file is incorrectly flagged as malicious when in fact it is benign. Regular expression rules created to detect shellcodes and base64 obfuscated script generated 47 false positive alarms on the clean dataset (10,000 websites). Table 1 shows the false positive rate of each scanning engine on the clean dataset.
Table 1: False positive rate of each scanning engine on the “clean dataset” (10,000 Websites)

<table>
<thead>
<tr>
<th>Engine</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClamAV</td>
<td>0.0%</td>
</tr>
<tr>
<td>AVG</td>
<td>0.0%</td>
</tr>
<tr>
<td>Regex Rules</td>
<td>0.47%</td>
</tr>
</tbody>
</table>

Determining the false negative rate of our honeypot involved scanning a malicious set of websites containing 110 drive-by download exploits we had collected. YALIH’s performance was also measured against other open source client honeypots (i.e. Monkey-Spider, HoneyC, Thug). MonkeySpider and Thug allow analysis of fetched web content on the local disk. On the other hand HoneyC’s analysis engine operates based on Snort’s rules and therefore scans network streams for signatures of malicious activity. To emulate malicious webserver for HoneyC’s analysis, a webserver was placed on the local network and malicious files in our dataset were uploaded. HoneyC’s rules were subsequently updated to the latest rules obtained from Snort’s official website and configured to retrieve the malicious content from the local webserver. It should be noted that individual signatures were not created in Yara for each exploit and only regular expression rules were generated to detect malicious characteristics of a website or script. Table 2 shows the false negative rate of each client honeypot’s analysis on the malicious dataset and the corresponding analysis time.

Table 2: False Negative rate of several low interaction honeypots in comparison to YALIH, on the “malicious dataset”

<table>
<thead>
<tr>
<th>Honeypot</th>
<th>False Negative Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monkey-Spider</td>
<td>53.6%</td>
</tr>
<tr>
<td>HoneyC</td>
<td>100%</td>
</tr>
<tr>
<td>Thug</td>
<td>35.4%</td>
</tr>
<tr>
<td>YALIH</td>
<td>19%</td>
</tr>
</tbody>
</table>

HoneyC’s subpar performance can be due to the fact that Snort as an intrusion detection and prevention system primarily focuses on server-side attacks and less on client-side browser-based exploits and malware. Its open source rules are also not updated as frequently as those of popular antivirus engines.  

4.1 Scanning Speed 
Low interaction client honeypots and specifically those based on signature databases have relatively very low scanning time. The scanning time depends on the size of the signature database; however the difference is so insignificant that, the difference can be ignored. 

Table 3 shows the scanning time for each detection engine on the clean dataset. The experiments were run on a 3.0 GHz system with 8 GB of memory and 64 bit edition of Debian Linux.

<table>
<thead>
<tr>
<th>Engine</th>
<th>Total Time (Sec.)</th>
<th>Each Website (Avg/Website)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total (Seconds)</td>
<td>1450</td>
<td>0.022</td>
</tr>
<tr>
<td>Antivirus Engines</td>
<td>220</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

These times are considerably lower than honeyware (Alosefer and Rana, 2010) and Capture-HPC high interaction client honeypot, which based on our tests achieved an average visiting time of 10 seconds and an average revert time of 15 seconds per URL. A Hybrid system could utilize YALIH’s fast scanning capabilities and a selection of broad pattern matching rules to detect suspicious content as the front detection system and pass them to a high interaction client honeypot for further analysis. Initially broad pattern matching rules of YALIH (e.g. hidden or small-sized iframes, <object>, <embed> tag variables) ensure minimum false negative rate while further scanning with the slower high interaction client honeypot is a measure to reduce false positive rate within the initial batch of detected websites.  

5 Referrer Tracking and Malware Delivery 
Delivering different content to a visiting user based on the referrer information is a practice widely used by web spam websites and referred to as referrer cloaking (Lin, 2009, Wang et al., 2011, Wang and Ma, 2006). Web spam techniques are used to boost the ranking of a website in a search engine result by presenting spam contents to users in order to attract clicks, and bogus content to a search engine crawler to avoid detection. 

Referrer cloaking is one of many web spam techniques and occurs when a remote website attempts to distinguish a regular visiting user from a search engine crawler using the referrer information transmitted by the visiting browser’s HTTP header information. Referrer information notifies the remote server which website was used by the user to reach their site. We believe referrer information is also one of the primary techniques to deliver malicious website to selected users and bypass detection tools in the process. Referrer black and whitelisting functions are seen in multiple Browser Exploit Packs (Figure 2). A crawler to detect cloaked websites must be able to adjust referrer information and follow multiple redirections, as redirection is one of the main techniques of web spam delivery. YALIH’s browser module is capable of following redirections and adjusting referrer information based on user’s needs.  

5.1 Real-World Experiment 
We gathered 24,000 suspicious websites - we refer to as “Real-World dataset” - from three public ad, tracking and malicious website databases (i.e. spyeyetracker, zeustracker, hosts-file.net) using the built-in Blacklisted URL function. The websites were subsequently retrieved using two different referrer settings and saved on local hard disk for analysis.
Our experiment involved adjusting the referrer information to reflect a user reaching the remote website from two different locations:

- Google’s search result: To reflect a user who has searched for a keyword and accessed the remote server through the Google results’ page.
- Top-Domain page – To reflect a user who has either directly entered the remote website’s address into the browser’s address bar or has reached the page through a top domain (e.g. http://www.yahoo.com/mail/ through http://www.yahoo.com).

In order to minimize the risk of other factors such as IP-tracking or geo-location tracking that might influence our results, each malicious dataset visit was performed within the same region, similar time and network subnet but using a different IP address.

Analogous to results from web spam studies, our data collection and analysis on real-world dataset using Google result’s search string as referrer, returned more fetched websites than when referrer was set to the top domain. Determining the top domain of a URL was achieved using “urlparse” standard python module. Although the data collection was performed simultaneously, a large number of websites with a top domain as referrer, returned “URL or connection time-out error”. While 717 websites in top domain as referrer information either throttled or refused content delivery.

Subsequent analysis also revealed higher number of malicious contents in the Google referred experiment. This illustrated how malicious websites change their behaviour based on a visitor’s specific attribute. (see Table 4, Figure 6).

<table>
<thead>
<tr>
<th>Referrer</th>
<th>Fetched websites</th>
<th>ClamAV</th>
<th>AVG</th>
<th>Yara/Regex Rules</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>223023</td>
<td>8</td>
<td>74</td>
<td>14</td>
<td>96</td>
</tr>
<tr>
<td>Top Domain</td>
<td>21306</td>
<td>8</td>
<td>60</td>
<td>12</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 4: Detection result of YALIH on 24,000 websites (Real-world dataset) using two different referers

![Figure 6: Detected malicious websites with various detection engines with two referrer variables.](Image)

YALIH performs JavaScript .js file extraction and analysis on visiting websites. As expect, a high portion of the detected malicious content in our real-world experiment consisted of infected .js files (see Table 5).

<table>
<thead>
<tr>
<th>Referrer</th>
<th>No. of Malicious Websites</th>
<th>HTML files</th>
<th>.js Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>96</td>
<td>39</td>
<td>57 (59.3%)</td>
</tr>
<tr>
<td>Top Domain</td>
<td>80</td>
<td>34</td>
<td>46 (57.5%)</td>
</tr>
</tbody>
</table>

Table 5: Ratio of malicious code within HTML and .js files 24,000 websites (Real-world dataset)

It should be noted that our real world dataset comprised of URLs fetched from three public ad, tracking and malicious website databases at a certain date and time. These databases are frequently updated to include newly found malicious websites or remove old and benign URLs. Similar experiments using the same blacklisted databases with different retrieval time may result in different results, as detected website may be offline, cleaned or removed from the databases. However the similar pattern of higher number of fetched content and malicious code using Google search string as referrer was observed in all the experiments that were performed on various dates using variations of our real-world dataset (i.e. the time list of URLs were fetched from the blacklisted databases resulting in updated and altered sets of URLs).

6 Discussion and Future Work

Signature-based detection has always been associated with high false negative rate of detection. This is because of their inability to detect zero-day attacks, variations of attacks and obfuscated attacks. We demonstrated that by combining various detection engines comprising of signature-based and pattern matching techniques, false negative rate of detection can be decreased while still maintaining the fast scanning capability of low interaction client honeypots.

Yara signature generation is a capability currently in development which allows YALIH to detect not only obfuscated and minified attacks but improve the detection of attacks based on variation of a malicious code. This is achieved in conjunction with signature databases of antivirus engines that identify malware families with very low false positive rate. Malicious files detected by antivirus engines that belong to the same family of malware or threats are isolated and parsed. Malicious code within the isolated website contents is then extracted and a set of common keywords/variables - present in all files within that family of malware - are extracted and used to create Yara rules (see Figure 7). This is believed to increase the false positive rate of client honeypot but ultimately result in lower false negative rate of detection.
Figure 7: Steps in automatic Yara rule generation

Research on client honeypots has primarily focused on improving the detection rate of the analysis engines. Malicious websites however employ simple yet effective techniques to bypass detection by client honeypots analysis engines entirely by restricting malware delivery to certain users through various cloaking techniques (e.g. identifying suspicious networks of honeypots, referrer cloaking, IP-tracking). Referrer cloaking and cloaking in general poses a challenge in the detection of malicious websites. Regardless of the type or the interaction level of a client honeypot, if a malicious website is not retrieved using the proper header, network or environment information, benign content can be served or connection be refused, resulting in a false negative. Our experiment focused on only two referrer information (i.e. Google search result string, top-domain string) to retrieve websites. The actual number of malicious websites using referrer cloaking in our real-world database is unknown since there are a high number of referrer variation which can be used by a malicious website. The challenge in detection of malicious websites using referrer cloaking is to properly identify the referrer information required by the website to serve malicious content, which can result in minimizing the number and cost of retrievals.

7 Conclusion

We have presented the design and analysis of a low interaction client honeypot that integrates a combination of multiple antivirus engines and pattern matching using string or regular expressions for detection. The honeclient is capable of extracting embedded JavaScript files and performs de-obfuscation and de-minification of scripts. Subsequent signature and pattern matching analysis are performed on fetched contents and de-obfuscated and de-minified scripts to detect signatures of attacks and static and potentially malicious characteristics of a website. Its embedded virtual browser reduces the possibility of detecting a virtual environment through handling cookies, redirection, sessions and imitating browser personality and referrer settings. We demonstrated through experiments with benign and malicious datasets that our designed honeclient was capable of achieving low false positive and false negative rate with very low scanning time compared to currently available client honeypot systems.

The real-world experiments on suspicious websites revealed that malicious websites monitor and target users with malicious content based on referrer attribute of a browser while visiting the website. This is in par with previous research on spam websites which employed referrer cloaking as a mean to increase their search engine ranking and avoid detection. Setting browser’s referrer attribute to a search string of a search engine seems to be one of the techniques to imitate a user who is visiting a website through a search engine result page. This technique can result in higher number of malware delivery by the malicious website and consequent detection.

8 References


