2017 15th Annual Conference on Privacy, Security and Trust

# Hiding Behind the Shoulders of Giants: Abusing Crawlers for Indirect Web Attacks

Apostolis Zarras
Technical University of Munich
zarras@sec.in.tum.de

Federico Maggi
Trend Micro Inc.
federico_maggi@trendmicro.com

***Abstract*—It could be argued that without search engines, the web would have never grown to the size that it has today. To achieve maximum coverage and provide relevant results, search engines employ large armies of autonomous crawlers that continuously scour the web, following links, indexing content, and collecting features that are then used to calculate the ranking of each page. In this paper, we describe how autonomous crawlers can be abused by attackers to exploit vulnerabilities on third-party websites while hiding the true origin of the attacks. Moreover, we show how certain vulnerabilities on websites that are currently deemed unimportant, can be abused in a way that would allow an attacker to arbitrarily boost the rankings of malicious websites in the search results of popular search engines. Motivated by the potentials of these vulnerabilities, we propose a series of preventive and defensive countermeasures that website owners and search engines can adopt to minimize, or altogether eliminate, the effects of crawler-abusing attacks.**

## I. Introduction

It is, at times, hard to imagine that search engines were not always part of the Web. Before the prevalence of search engines, users were finding content either by following links or hearing about websites, or even attempting to guess the domain of a website; for instance, a user searching for "California wine" guessing that the `californiawine.com` is the most appropriate website. Consequently, an abundance of publicly available knowledge remained hidden from the vast majority of the early Internet population who relied to the aforementioned techniques to discover useful information.

This way of navigating the Web has changed dramatically with the advent of search engines. While the primary search engines had an index of few thousand web pages [19], their modern versions count tens of billions of them [28]. Search engines are trusted to provide relevant content to users, in response to their search queries. Moreover, due to their unbiased ranking algorithms, the results the users get are the ones that best match their interests. Undoubtedly, this was a total departure from previous website links, where the users could not know whether a host-website is linking to a destination-website because it has the most relevant content, or because the destination-website is actually paying the host-website a monthly fee for having its link listed.

Today, search engines are responsible for the majority of the content-discovery. The tight coupling of search engines and modern browsers in the form of dedicated input fields next to the browser's URL bar—or piggybacking on the URL bar itself—is further evidence that users rely more and more on search engines. To provide relevant content, search engines employ large armies of automated website crawlers. These crawlers are constantly navigating the web, following links, indexing content, and gathering statistics for each discovered page. The gathered data are combined to produce a rank for each page, which is then used to provide ordered search results when the users search for relevant terms.

The higher a website is listed on search results, the more likely it is that a user would click on that link instead of a competing one [23]. This has given rise to a wide range of techniques used by websites to manipulate the findings of crawlers such that the websites appear on a higher ranking in a search engine's results. These techniques are part of the *search engine optimization* (SEO) toolbox and can range from benign actions (e.g., refactoring a page's HTML code to be easily consumed by crawlers) to blackhat ones (e.g., purchasing of backlinks from other websites or stuffing of each page with multiple keywords) [17], [29].

In this paper, we investigate the extent to which attackers can abuse the logic of search engine crawlers to perform various attacks. We show that an attacker can convince crawlers to launch attacks against third-party websites by crafting the appropriate links. For instance, an attacker who knows that a remote website is vulnerable to a SQL injection, can construct a malicious URL that exploits that vulnerability and have the crawler of a search engine follow that link instead of following it by herself. The attacker can then exfiltrate the results of that attack through numerous ways (e.g., inspecting the cached page of the vulnerable website on the search engine's website). Apart from totally shielding the attacker from a post-mortem analysis of the attack by the operators of the vulnerable website, such attacks create additional problems. If, for instance, a web application firewall detects such an attack and decides to block the traffic coming from the IP address of the attacking host, it will essentially be blocking the crawler of a large search engine, an action with negative effects for the website's visibility on the search results of that specific search engine.

In addition, we show that vulnerable websites can be used to boost the ranking of attacker-owned domains. This is feasible through the careful construction of links that, when followed by a crawler, will provide web pages with backlinks

978-1-5386-2487-6/17/$31.00 ©2017 IEEE
DOI 10.1109/PST.2017.00049

towards the attacker-owned or third-party domains. These injected backlinks will positively affect the ranking of the adversary's website, which can in turn be used for scams and drive-by download attacks. Overall, we examine the 2013 top ten security risks listed on the Open Web Application Security Project (OWASP) website [20] and reveal which web crawlers can be manipulated to perform attacks against those vulnerabilities.

Finally, we propose a series of deterministic and learning-based countermeasures for the detection of malicious outbound links. For the former, we propose the notion of *authorized links* (i.e., links whose legitimacy can be verified by a search engine crawler) and show how they can be realized, in a back-compatible way, using existing web technologies. For the latter, we use anomaly-detection to establish a notion of *normality* for the outbound links of any given website, which can be used by the site operator to detect abnormal outbound links.

In summary, we make the following main contributions:

- We provide a systematic overview of attacks due to the abuse of search engine crawlers and study the consequences of different attacks on search engines as well as the affected third-party websites.
- We deploy vulnerable sites together with attacker-controlled websites and measure the susceptibility of the crawlers of various search engines and the degree to which they unwillingly "collaborate" with an attacker.
- We propose pragmatic, deterministic, and design-based countermeasures along with learning-based mitigations and evaluate their efficiency.

## II. Background

In this section, we provide background information on web indexing and website ranking as well as a quick reminder of the power that modern web application vulnerabilities provide to an attacker.

### A. Web Indexing

Web indexing refers to various methods and algorithms for indexing the contents of a website. Search engines extensively use web indexing to harvest, categorize, and rank previously unknown websites. In other words, search engine indexing collects, parses, and stores data to facilitate information retrieval.

To retrieve the content of websites, search engines rely on crawlers. A web crawler (or web spider) is a program that browses the web in a methodical and automated manner. A web crawler begins its task with a list of seed URLs to visit. As it visits these URLs, it identifies all the hyperlinks in the visited page and adds them to the list of URLs that remain to be visited. Through this process, a crawler not only identifies the subject and the quality of information on each page, but also the pages to which a website links to.

The latter information is used by many ranking algorithms, including Google's *PageRank* [4]. *PageRank* estimates the popularity of a web page by means of its linking. Every web page has a *rank* that represents its estimated popularity. The page's rank is determined by the number and ranking of its incoming links (called *backlinks*). This means that a web page is highly ranked if it has backlinks with a high rank or a large number of backlinks with a low rank. The rank $R(p)$ of a web page $p$ is equal to the sum of the ranking $R(b)$ of each backlinking web page $b$ divided by the total number of its outlinks $|O(b)|$:

$$R(p) = c \sum_{b \in B(p)} \frac{R(b)}{|O(b)|}$$

where $c < 1$ is a normalization factor, $B(p)$ the set of backlink web pages of page $p$, and $O(b)$ the set of outlinks of a page $b$ (i.e., all the links on a web page which are not navigational links). Another ranking algorithm is the *Hyperlink-Induced Topic Search* (*HITS*) [14], which uses the link structure to identify good web pages related to a specific topic. In contrast to *PageRank*, the web graph is not rated as a whole entity, but only a subgraph that contains web pages relevant to the searched keywords is taken into consideration.

One can straightforwardly understand that websites with a substantial number of backlinks—especially if they come from highly ranked websites—will receive a better ranking in a search engine's results. As a consequence, users typically visit the highest-ranked web pages and ignore the rest [23]. Unfortunately, this fact also attracts miscreants who try to leverage these mechanisms to lure more victims to their (malicious) websites.

### B. Web Vulnerabilities and Exploits

The complexity of modern websites, along with the mixing-and-matching of platforms and extensions, are some of the root causes of web application vulnerabilities. The existence of website vulnerabilities, such as cross-site scripting (XSS) [26], SQL injection [3], cross-site request forgery (CSRF) [2], command injections [24], HTTP parameter pollution (HPP) [1], and HTTP response splitting [13], are among the most pressing security problems on the Internet today.

The attackers use these vulnerabilities to exploit websites. Even vulnerabilities such as SQL injections and XSS, which are well-known and have been studied for years, are still frequently exploited and constitute a significant portion of the vulnerabilities discovered each year [5]. To automate the exploitation process, cyber-criminals use black-box web vulnerability scanners. These tools crawl a website for known security vulnerabilities and if they found one or more of them, they generate specially-crafted input values to exploit them.

A vulnerability that has been exploited can have a negative impact to the website itself and its visitors. For instance, attackers that gain access to a website's database can modify or delete selected entries, or even exfiltrate sensitive data such as user credentials [25]. Additionally, they can modify the content of the website by including malicious scripts, redirecting the traffic to malicious websites, or modifying advertisements to generate revenue for themselves. Consequently, an exploited website can completely destroy its reputation.

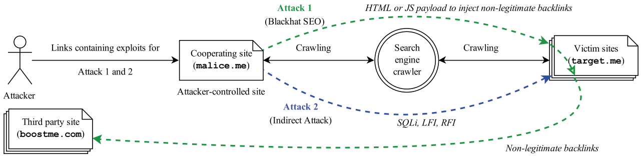

Figure 1. Overview of the attack scheme.

## III. Security Problems

In this section, we describe how an attacker can take advantage of web crawlers against the search engine company or to launch indirect attacks that cloak attacker's true identity.

### A. Attacker Model

Referring to Figure 1, we assume the existence of vulnerabilities in a website (i.e., `target.me`) that is an attractive target for an attacker. For instance, the website can contain valuable information, such as user credentials, or possess a high page rank. Additionally, we assume that an attacker wants to benefit from these vulnerabilities. However, she does not want to leave any traces of her actions. Moreover, she may also know that specific actions can trigger alerts in the website's intrusion detection system (IDS), yet they might appear benign if they come from popular web crawlers (e.g., due to whitelisting). Hence, she decides to take advantage of search-engine crawlers to perform a series of indirect attacks.

The types of indirect attacks that an adversary uses depend both on the targeted website as well as the extent to which the attacker can manipulate search engine crawlers. We classify the attacks in two different categories: $(i)$ attacks that promote a third-party website by abusing a vulnerable website and $(ii)$ attacks that directly affect the targeted website. The first includes scenarios, such as blackhat SEO attempts, whereas the second includes classic HTTP-based attacks against the server-side software of `target.me`. In the following sections, we discuss the details of the aforementioned threats, in which the adversary leverages a cooperating site (i.e., `malice.me`). This could be a site that is unknowingly helping the intruder (e.g., a link aggregator or a blog that allows posts) or created by the attacker herself. In both cases, the requirements are that the adversary can post links with arbitrary GET parameters.

### B. Blackhat SEO Attacks.

To determine the reputation and popularity of a web page, search engines commonly rely on the number and ranking of the other web pages that link to it. In essence, the more websites linking to a page $p$ and the more popular these websites are, the higher will be the rank that the page $p$ will receive from a search engine [4]. Although this is a reasonable way to define page ranking, it can be also exploited by cybercriminals to increase the rank of their web pages by increasing the number of websites linking to them. Although SEO is acceptable by search engines as a way to achieve a more accurate ranking of the websites, blackhat SEO is considered as an "unfair" way of boosting web page ranking. As such, adversaries can leverage *code injection* attacks against vulnerable websites and insert backlinks that point to malevolent web pages.

Table I
OWASP 2013 top ten critical web application security risks.

| | |
|---|---|
| A1 | Injection |
| A2 | Broken Authentication and Session Management |
| A3 | Cross-Site Scripting (XSS) |
| A4 | Insecure Direct Object References |
| A5 | Security Misconfiguration |
| A6 | Sensitive Data Exposure |
| A7 | Missing Function Level Access Control |
| A8 | Cross-Site Request Forgery (CSRF) |
| A9 | Using Components with Known Vulnerabilities |
| A10 | Unvalidated Redirects and Forwards |

### C. Targeted Attacks Against a Website and its Visitors.

In contrast to attacks that abuse a targeted website to increase the ranking of an attacker-controlled or third-party website, there exist others that directly affect the targeted website itself. The main intention behind these attacks is to harm the vulnerable website; either with the manipulation of its data or by stealing sensitive information. The compromised website can then provide to attackers usable resources that can be leveraged to send spam, attack other targets, or even infect with malware the visitors of the website. Even vulnerabilities such as Cross-Site Scripting (XSS), which are seem to be not so harmful and are often underestimated, can act as the launching point for attacks on visitors of the website.

As the motivation behind these attacks is mostly driven by financial criteria, cybercriminals will become even more resourceful in the media used for deploying their attacks. Therefore, if they can effectively "maneuver" popular web crawlers to perform these attacks instead of them and thus concealing their true identity from the targeted website, we believe that in the future will see more and more such types of attacks. Therefore, a study on the feasibility of these attacks is of great importance.

## IV. Susceptibility Assessment

In this section, we describe how we assess the presence of the aforementioned security problems in the real world. More precisely, we show the feasibility of the attack schemes by implementing them against the 2013 top ten critical web application security risks listed by OWASP [20]. As such, we first describe the methodology we use to perform the attacks and then we evaluate the inclination of web crawlers to launch them. We believe that we have constructed a realistic scenario which aims to raise the awareness of web crawlers' programmers to implement better filtering mechanisms.

### A. Methodology and Measurement Infrastructure

A preliminary step for assessing a web crawler's capacity to blindly follow URLs, consists of attacking a vulnerable website. Since it would be unethical to target a real site,

Table II
FEASIBILITY OF EACH ATTACK.

| Bots | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 |
|---|---|---|---|---|---|---|---|---|---|---|
| GoogleBot | 83 | 53 | 91 | 103 | 45 | 41 | 28 | 43 | 0 | 44 |
| MJ12Bot | 14 | 5 | 24 | 18 | 13 | 27 | 0 | 8 | 0 | 6 |
| PaperliBot | 11 | 11 | 17 | 29 | 3 | 9 | 6 | 7 | 0 | 3 |
| FacebookBot | 9 | 6 | 22 | 20 | 10 | 5 | 7 | 6 | 0 | 8 |
| BitlyBot | 8 | 7 | 18 | 18 | 7 | 6 | 6 | 6 | 0 | 6 |
| YahooBot | 11 | 5 | 9 | 6 | 6 | 5 | 5 | 8 | 0 | 3 |
| Yandex | 4 | 4 | 10 | 13 | 10 | 6 | 0 | 1 | 0 | 6 |
| BingBot | 4 | 5 | 12 | 10 | 7 | 5 | 2 | 4 | 0 | 2 |
| TwitterBot | 7 | 5 | 10 | 11 | 2 | 3 | 4 | 2 | 0 | 3 |
| Baidu | 10 | 3 | 5 | 7 | 9 | 0 | 0 | 1 | 0 | 0 |
| SocialRankIOBot | 4 | 3 | 2 | 7 | 3 | 3 | 4 | 4 | 0 | 2 |
| AppleBot | 2 | 2 | 4 | 9 | 2 | 3 | 2 | 2 | 0 | 2 |
| TweetmemeBot | 2 | 0 | 7 | 6 | 1 | 2 | 4 | 1 | 0 | 3 |
| LiveLapBot | 3 | 0 | 3 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

we deployed our own website, which gave us the ability to select the vulnerabilities we wanted to assess. Therefore, it was possible to create a website prone to the OWASP risks (see Table I) and thus allowing a comprehensive assessment.

Next, we developed the attacker-controlled website that targeted all these vulnerabilities by generating the appropriate attack-including links and mixing them with other benign links. To camouflage it as a regular website, we updated its content in daily basis. Finally, we advertised our attacking website in all the major search engines using their appropriate website-submission forms. We noticed that search engines were very eager to index the newly available content. Indeed, our web server logs revealed that the biggest search engines commanded their crawlers to visit our websites on the same day we submitted our links for indexing. In addition, to incorporate crawling from more agents we also created social media accounts and posted our attack-including URLs through them. We also used URL shortening services to shorten the URLs and then share them through our social media accounts. It is worth mentioning that we deactivated the accounts we used for our study once the experiment was over.

After crafting and publishing these URLs, we monitored and analyzed our web servers (i.e., Apache) log files to discover which web crawlers visited our websites and executed the attacks we had embedded.

### B. Findings

We ran our experiment for a period of 40 days. During this time, we noticed 4844 visits to the attacker-controlled website. We observed that the majority of crawlers visited our website only few hours after we had submitted it to search engines. In few cases it took up to 24 hours for the crawling process to start after the original submission of the website. However, it should not be assumed that all the crawlers' visits resulted in execution of some sort of attacks; some visits were just to discover new pages.

On the other hand, the targeted website was not directly advertised to search engines which resulted to less visits compared to the attacker-controlled website. As a matter of fact, the crawlers' visits were heavily depended upon the embedded links in attacker-controlled websites and links shared on social media. We followed this path, because we wanted to observe if web crawlers are willing to visit a website, which has only been advertised through URLs that contain an attack against this website. Overall, we observed 1456 visits.

Table II summarizes our findings, showing the successful attacks that can be performed through different web crawlers. Note that we omitted attacks that originate from real users and those belonging to web crawlers that we could not trace back to a specific service.

**Coverage.** We observed that all web crawlers performed at least one attack, with some launching nearly all attacks. This means that all the examined crawlers are prone on being manipulated by attackers. This is an important finding, because it proves that web crawlers can be manipulated by attackers who want to hide their malicious activities. Another significant finding is that the attacks posted in social media, were executed almost immediately (or at least with some hours delay). This means that an attacker can create fake profiles in social media and leverage them to attack a third-party website. In this case, the adversary does not even need to spent money for hosting the attacker-controlled website. One may notice that the *A9* attack is zero within all visited crawlers. The reason for that is because we did not construct a URL to target this vulnerability, however, based on the other outcomes a well-crafted URL can exploit this vulnerability as well.

**Speed and Frequency.** We observed that some web crawlers visited the vulnerable website in a periodic manner and launched the attacks more frequently compared to others. An attacker can benefit from this fact to make some attacks more persistent. For instance, if the attacker knows the exact vulnerability she wants to target, then she can advertise the co-operating site to the most persistent web crawler to maximize the efficiency of the attack. Also the time that the attack will be launched by crawlers can be a approximately estimated. An adversary can leverage this to know exactly when to target a website; for instance, during night when the maintainer or administrator of the website is probably sleeping.

## V. DEFENSES

In the previous sections, we showed how an attacker can abuse crawlers to conduct attacks against third-party servers as well as to boost the rank of her sites. In this section, we discuss how these attacks can be mitigated. We have purposefully chosen to expand more on the attacks involving the addition of backlinks since they are harder to detect using existing technologies.

### A. Stopping Targeted Attacks

For an attacker to successfully manipulate a crawler to conduct server-side attacks, a remote web application must be vulnerable to a server-side attack. One could argue that it is the web application's responsibility to protect itself and even if a crawler has been used to conduct an attack, the search engine behind it cannot be held responsible.

One solution that can be utilized by a web application to protect itself is through the use of a *Web Application Firewall* (WAF). Upon detecting an attack, the WAF will drop the request, and potentially blacklist the IP address of the offending host. However, blocking the offending IP address can lead to complications when an attacker is conducting her attacks through crawlers which can cause the unwilling blacklisting of the site from that specific search engine. A naive way of handling this corner case is to consult the user-agent string of the incoming HTTP request and block the requests coming from bots. This, however, could be straightforwardly abused by attackers in order to fully bypass the firewall. As such, we reason that this strategy must be combined with reverse-DNS lookups. A WAF that received an offending request by a supposed search-engine bot can use a reverse-DNS lookup in order to establish that the bot does in fact belong to the claimed search engine. If it does, then the WAF should avoid blacklisting its IP address and merely drop the request, understanding that the bot is merely an unwilling actor in this attack.

### B. *Stopping Blackhat SEO Attacks*

We now focus on the problem of non-legitimate backlinks, where an attacker abuses vulnerabilities to inject backlinks to attacker-controlled sites. We discuss two possible solutions that place most of the burden either on the website owner or on the search engine. These solutions are compatible with each other and thus can be both used at the same time to improve the overall accuracy of a malicious-link detecting system. Also, they are all backwards-compatible—in the sense that sites and search engines that choose not to adopt them will continue working seamlessly—and require no changes in the web technologies.

**❶ Deterministic Solutions: Authorized Links.** A deterministic solution to this problem is to provide mechanisms to website administrators that can be used to denote which links are legitimate. If any links are found on a page that are not explicitly approved by a page's policy, the crawler can treat them accordingly. To this end, we propose the concept of *authorized links*, which are links augmented with authorization information that a crawler can inspect and verify. We posit that the use of *message authentication codes* (MACs), nonces, and whitelisting are good mechanisms for realizing the notion of authorized links as all have been successfully used in the past to achieve integrity goals. Moreover, the nonces and whitelisting method can be realized using the *Content Security Policy* (CSP) mechanism, thus alleviating the need for implementing yet another server-driven security policy mechanism as well as training developers on how to properly use it. In the following we discuss these solutions in detail.

*MAC-based Solutions.* Each link is augmented with a hash of that link concatenated with a shared secret between the website and the search engines. Since website administrators need to interact with search engines to submit their websites' URL for crawling, it is straightforward to add an extra step where an administrator, after verifying that she is indeed the owner of a specific domain, shares a secret key with that search engine. This key can be used in the future to verify the integrity of the links found on her website.

The scheme works as follows: the owner of domain $D$ agrees on key $K$ with a specific search engine. For every link $L$ toward remote domains, the owner of $D$, augments the anchor tag with the result of $H(K||L)$ where the function $H$ is a strong cryptographic hash function. Given an outbound link to `example.com` and a key of "secret", the HTML markup for an anchor tag would be the following:

```
<a href="http://example.com"
     data-mac="26b0c646651045370bf849a0170097c8">Click
     here</a>
```

where the `data-mac` attribute is the result of the MD5 hash function on the string "`secrethttp://example.com`". All `data-*` attributes are invisible to normal users of a website and are part of the HTML5 specifications. However, a crawler belonging to a search engine that knows in advance the key of this website can recompute a link's MAC and disregard the link, if the computed MAC does not match the MAC available on the website.

Even though the process of creating MACs for every outgoing link is likely to be arduous, it does not need to be performed manually. Unless the owner of a website is writing HTML by hand, "What You See Is What You Get" (WYSIWYG) editors that are available for all modern content management systems can fully automate this process. For instance, when the user is writing a new blog post in WordPress and clicks on the button that enters a new link, the editor can automatically fetch the key from the website's database, compute the proper MAC, and append it to the generated markup. Even for developers who decide to write HTML by hand, a script that receives as input the secret key can parse the HTML file, compute the appropriate MACs and rewrite the HTML code to support the notion of authorized links.

*Nonce and Whitelisting.* In a nonce-base solution, each link toward a remote domain is augmented with a non-predictable identifier that is different for every page load. The Content Security Policy (CSP) 1.1 draft allows developers to include inline JavaScript in their web pages (forbidden in the original CSP specification) as long as each inline script specifies the correct nonce [27]. The correct nonce is communicated to the browser through the CSP header. For example, a browser that receives the following HTTP response:

```
HTTP/1.1 OK
Content-Security-Policy: script-src self ‘nonce-1q2w3e4r’;
[...]

//Legitimate script
<script nonce="1q2w3e4r">[...]</script>

//Malicious injected script
<script>[...]</script>
```

will allow the first inline script to execute but will stop the second one which is not carrying the proper token. As such, these nonces essentially allow capability-based access control. As long as a script has the appropriate nonce (as specified in the CSP header), the browser will allow it to execute. The same functionality can be extended to protect links. That is, the CSP header can denote an `ahref-src` attribute, which will specify a nonce for the legitimate links. Assuming that the nonce is sufficiently random and changes on every page load, the attacker who is injecting backlinks will have no knowledge of the nonce that will be given by the victim server to the crawler. As with the MAC-based solution, both nonces can be implemented by WYSIWYG editors and content management systems, which will automatically emit the appropriate nonces without the user even being aware of their existence.

The benefit of this solution over the MAC-based approach is that website owners do not need to exchange any long-term secret keys with search engines. At the same time, since these nonces have to change upon each page load, the server should be careful when caching pages at the server-side. A caching of nonces can, in principle at least, allow an attacker to inspect a nonce and then, assuming that the nonce is reused, inject nonce-including backlinks on the vulnerable websites. Fortunately, several server-side frameworks allow the caching of fragments of a web page [8] (as opposed to caching an entire page) and thus servers do not need to forgo the performance benefits of cached web pages.

*Hybrid Solution.* An alternative solution that combines the benefits of no-secret exchange with search engines, as well as worry-free caching, are whitelists of allowed outbound links. As in CSP, a website can send to the client a list of authorized remote domains through its CSP headers. As long as each outbound link belongs to a domain that is part of the header-specified domains, the crawler can treat it as an authorized list. Depending on the nature of each website, the whitelist can be a global one for the entire domain, or it can be specified per subdomain or per path. In all cases, the WYSIWYG editors should update this list as new URLs are added to each page.

*Server-side versus Client-side Links.* All aforementioned mechanisms require that the web server is responsible for all the links that arrive at a user's browser. As such, if a significant fraction of links are dynamically created at the client side, our deterministic server-side approaches will not be able to account for them. However, we found out that the vast majority of links are server-created links, with only 283 out of 10,000 cases of dynamic, client-side-generated links. To quantify the dynamic creation of links at the client side 10,000 sites, we crawled 10,000 websites from `alexa.com` and filtered-out all the links that led to an advertisement, as well as same-site, navigation links.

❷ **Learning-based Mitigation.** The problem of automatically distinguishing non-legitimate from legitimate data in web applications has received ample attention in the anomaly detection literature. At the price of some false alerts, anomaly detection methods fill the gap left by misuse-based solutions (e.g., blacklists, classic signature-based WAFs). Given the state of the art, and the possibility of combining anomaly- and misuse-based solutions on a modern web application, we can reasonably assume that the HTTP requests toward large and popular websites are already screened to mitigate suspicious payloads.

*Threat Model.* A conservative threat model must at least assume that non-legitimate outbound links have somehow "slipped through the cracks" and are now dangerously displayed on a page, waiting for the crawler to follow them. This can happen, for instance, on websites that allow anyone to post arbitrary links (e.g., comments on blog posts, directories, and bookmarks). In these cases, the website operator simply does not want to invest resources to scrutinize every posted link, especially *a posteriori*. For example, Maggi et al. [18] showed that once attackers succeed in bypassing the first line of defense and creating malicious shortened aliases, the operators never check such aliases a second time. Another example is that an attacker could have bypassed any security measure along the way. There may be several reasons for this, such as the attacker can obfuscate the backlinks using redirections or adopt other sophisticated techniques.

The bottom line is that there exist ways for an attacker to bypass the first line of defense without the operator noticing it. Unfortunately, once the first line of defense is bypassed, the website operator misses the chance of accurately detecting a malicious link, because the contextual information (e.g., source IP, request headers) is not retained forever.

In all these cases it is be beneficial to perform an *a posteriori* analysis on existing outbound links to differentiate the legitimate from the non-legitimate backlinks, based exclusively on the available information. In a conservative approach, such information is simply the link itself, thus lacking any provenance meta-data. The threat posed by search engine bots presented in this work is one, relevant case. However, we believe that having such a countermeasure would be very useful for forensics purposes and other investigation tasks (e.g., periodic housekeeping of free blogging platforms).

*Modeling Legitimate Links.* Given an arbitrary web page, `target.me/page.php`, our goal is to detect outbound links pointing to a website outside the control of the web page owner. Thus, we focus on links that have paid-level domains (PLD) different from `target.me`. Of course, we do not consider hyperlinks across distinct domains from the same organization as outbound links, because these can be easily whitelisted. For instance, links from `youtube.com` to `google.com` are certainly legitimate. Moreover, we only consider links that can be used to encode an exploit or, in other words, those that contain a query string (e.g., `boostme.com/path/p.php?par=var`), including dynamically-generated links (e.g., via JavaScript). In general terms, we are interested in the URL contained in those GET or POST requests originating when the search engine bot follows a link referred by the source page on `target.me`.

We analyze all these backlinks that the website operator does not know how to handle; these that have already gone through whatever countermeasures and whitelisting filters employed up to now. The challenges we face are: $(i)$ there is little contextual information attached on a standalone link and $(ii)$ there are multiple classes of links under the same domain (e.g., long links with many parameters, short links with few but long parameters, other links with just integers, floats or tokens as parameters). The features proposed in the related literature (e.g., [15]) for detecting anomalous HTTP requests are inapplicable in this context. First, they are designed to detect anomalous requests directed *toward* a web application as opposed to unexpected outbound links. Second, state-of-the-art methods assume that there is quite a regular structure in the analyzed URLs, since they all encode a request to a single web application or to a small set of web applications. In this setting, anomalous requests can be detected quite easily by finding out-of-sequence parameters, long parameters, special characters in the payload, etc. However, when this assumption is removed, the problem becomes harder. In principle, it is possible to apply state-of-the-art web application anomaly detection techniques by creating one model per outbound domain, striving to learn the regularities of the requests directed toward each external site. However, scarcity and uneven distribution of data may limit the applicability of such an approach. Last, these approaches work well when used *in conjunction* with other models (e.g., by creating correlated request-response models [15], timing features, previous knowledge on the request handler, etc). However, this information is not available on the referring pages hosted on `target.me`.

*Characterizing Features.* For the referring site that wants to perform checks on the outbound links, we propose a set of lightweight features and a simple but effective learning technique that makes no assumptions about the non-legitimate links. The only requirement is a set of legitimate links.

We do not aim to provide perfect recognition, nor to create an alternative for detecting malicious payloads, as we are well aware that having only the link's string representation gives us a very limited view. Perfect protection can be obtained with the deterministic solution described earlier, at the price of design changes. When this is not feasible, the solution described in this section gives reasonable protection at zero cost, which is already a benefit when compared to the baseline (i.e., unprotected site). Moreover, the website operator may combine our solution with other site-specific filters that leverage domain knowledge to mitigate errors.

Specifically, for each link, we represent the string after the first slash with the following feature vector:

- $l$ (integer): number of symbols, including any character class;
- $d$ (integer): the depth of the path, that is the number of "/";
- $s$ (integer): number of special characters;
- $u$ and $U$ (integer): number of lower and uppercase alphabetical characters;
- $p$ (float): mean length of parameters' names, counting all symbols prior "=";
- $v$ (float): mean length of parameters' values, counting all symbols after "=";
- $n$ (float): number of parameters.

In addition to these features, we run a pilot experiment including the frequency of each symbol in `[a-zA-Z0-9]` and special characters as features. However, we obtained unsatisfactory results and significant speed penalties due to the increased dimensionality (above 104 features).

*Training and Detection.* We use the aforementioned features to fit a model that can be used to decide whether a new outbound link is non-legitimate. Our model can be trained at various aggregation levels, depending on the working environment. For instance, a site with regular links throughout all the pages can train one model, whereas larger websites can train one model per site section. Given the problem setting, three broad modeling approaches can be applied.

In the optimal case when the website operator has knowledge about the characteristics of the non-legitimate links she wants to detect, a supervised learning approach can be used. Using terminology from the machine learning field, this is essentially a binary classification problem. This yields the best recall and precision, although the assumptions underneath this approach are not always realistic. Indeed, if one of the two classes of links is not well represented during training or if it changes dramatically during operation, the quality of detection may decrease over time.

Another approach consists in mapping the problem to a one-class classification task, or semi-supervised learning. Essentially, we ignore non-legitimate links and train the classifier exclusively on legitimate links. Although this approach has a recall close to one hundred percent, it may suffer from many false positives.

A third, and more realistic, approach consists of not mapping this problem to a classification task. Instead, we show that a simple outlier detection technique performs very well, without requiring any assumption on how the feature values are distributed. Additionally, we do not require any knowledge about the outliers.

Our technique is inspired by the histogram-based outlier score (HBOS) [10]. We split the list of legitimate links available for training in two batches. On the first batch, for each feature, we calculate the relative-frequency histogram using a fixed number of equally-sized bins on the training data. The number of bins, as well as other parameters, can be easily tuned on a per-site basis. Interestingly, this method is suitable for online learning, as the frequencies can be updated without batch re-training as new samples come in. As result, we obtain $M = 7$ histograms, where $M$ is the size of the feature vector.

On the second batch, we calculate the following score:

$$\mathrm{HBOS}(v) = \sum_{i=0}^{M} \log \frac{1}{\mathrm{freq}_i(v_i)} \tag{1}$$

where $v$ is an $M$-sized vector holding the feature values for each link in the batch and $\mathrm{freq}_i(v_i)$ is the frequency of the

$i$-th component of the vector $v$, which is actually the height of the corresponding bin in the $i$-th histogram. If a value has zero frequency, we assign it an arbitrarily low value to allow the calculation of the fraction and logarithm. Any low value close to zero yields a very high HBOS component, to account for the never-seen-before value.

We now calculate the mean and standard deviation of the HBOS, which essentially expresses the allowed values of outlier score of legitimate links. Since the legitimate links have a lower variability than non-legitimate links (HBOS values are instead more distant from the mean), we can create a decision function for determining whether a new link is legitimate, given its feature vector $v'$:

$$\text{Legitimate}(v') = \text{HBOS}(v') \leq \alpha \cdot \mu + \beta \cdot \sigma \qquad (2)$$

This is based on Chebyshev's inequality, where the $\alpha$ and $\beta$ parameters can be tuned on a per-site basis. Note that this decision boundary works without assuming any specific underlying distribution of features. Overall, this lightweight technique allows flexible tuning, explanation of the reason for considering a link as an outlier, and excellent results. Indeed, the website owner can examine each alert and *see* which feature(s) contributed most to a high $\text{HBOS}(\cdot)$ score (e.g., link with unexpected number of parameters, or too many special characters).

*Feasibility Evaluation.* We implemented a proof-of-concept of our approach in about 600 lines of Python code (including code required for automating the experiments), leveraging the SciPy [12] framework for statistical computation. Our prototype parses the path and query string of each outbound link and calculates the aforementioned features. Next, it performs the training and estimation of the $\mu$ and $\sigma$ for a given site. These values are then used for deciding if new links are legitimate or not.

Using the JavaScript-supporting, headless PhantomJS [11] browser, we collected 795,274 outbound URLs from the top 5,000 Alexa websites. We excluded websites with no public pages, which required registration and login (e.g., Facebook, Twitter, LinkedIn), as they would not be targeted by public search engine bots, and sites with only a handful of outbound links. Our crawling script, based on CasperJS [21], started from the initial seed, enqueued inbound links to continue the crawling process on each site, and saved outbound links. As a source of non-legitimate outbound links, we used the dataset from our measurement setup described in Section IV. On average, we collected 244.1 (± 319.97) outbound links per site, with peaks up to 2,268 links. Given this skewed distribution, we focused our experiments on the top 400 sites having at least 200 outbound links each. The following experiments were repeated ten times for each data point on a randomized and shuffled train-test split (i.e., 10–fold cross validation).

Feature extraction is extremely fast, even in our proof-of-concept prototype written with an interpreted language running on a laptop. On average, on our entire dataset, 0.7513ms (± 0.3375ms) are required to extract the features from one link, using a few megabytes of main memory.

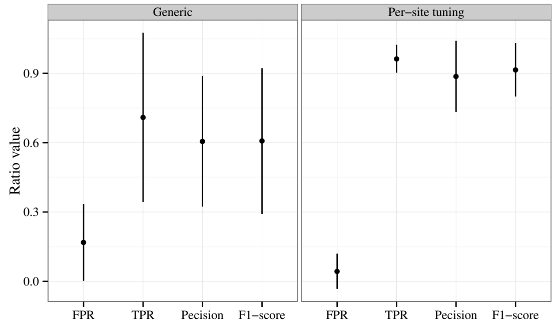

Figure 2. Overall final detection quality in terms of F1 score, precision, TPR and FPR (before and after per-site tuning).

The default parameters in Equation 2, $\alpha = 1.0$ and $\beta = 2.0$, the arbitrarily low frequency to be assigned to novel feature values in Equation 1, $0.0001$, and the number of bins of the histogram, result in a very diverse detection performance, ranging from sites with very high TPR to sites with too many FPRs. This led us to consider that each site could optimize the detection by choosing the combination of parameters that maximizes the precision, or the F1 score (high F1 score means both high precision and high recall, as $F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$). We found out that each site has indeed a distinct combination of parameters that works better than others. Globally, with these per-site tuned models, we obtain a better detection as shown in Figure 2. In addition to optimization, each site owner can pre-group outbound URLs based on domain knowledge (i.e., URLs going to site X or Y), so that group-specific models can be fitted.

From Figure 2, the vast majority of websites can detect non-legitimate outbound links with more than 90 percent of TPR and less than 6 percent of FPR. Depending on whether handling a false positive costs more than missing a true alert, the website owner can easily tune our model to minimize the respective costs. In our experiments, we tuned the models to maximize both precision and recall, and thus to obtain the highest possible F1 score.

Among the false positives we found $(i)$ inbound URLs with a different domain but still affiliated to the same organization, $(ii)$ well known banner circuits, or $(iii)$ social buttons, which can be easily filtered out with a simple whitelisting.

## VI. Discussion

Web crawlers manipulation is a new field of study in the context of web crawlers. As we showed in this study web crawlers can be exploited to launch attacks against vulnerable websites. However, as with any other study there exist some limitations. In the following we expose these restraints and discuss how we can potentially address them.

**Attacker Model.** For our attacker model, we considered two important issues that were caused by the behavior of crawlers:

blackhat SEO and targeted attacks. Current crawlers, however, are expected to deal with increasingly complex web applications that require both JavaScript support as well as increased computing resources. Therefore, we envision other attacks, where the CPU cycles of web crawlers could be abused to run complex client-side software that performs malicious tasks such as crypto-currency mining and password bruteforcing.

**Measurement.** Our measurement focused on web crawlers as they are the most important category of "link followers". However, the security issues that we highlighted in this work can in principle affect any environment where an automated browser follows a link. Expanding our analysis further can only reveal *more* instances of the very same, concerning problem.

**Deterministic Defenses.** We proposed simple yet effective design principles that could prevent abusing web crawlers entirely. The assumption is that the website owner or web application developer can decide which links are safe to be crawled and which ones are not. Our solution is perfectly compatible with the current Web and works for both statically and dynamically generated links. However, it is less applicable to websites that host massive amounts of arbitrary user-generated content. In these cases, we had to resort to a non-deterministic mitigation approach, which is intrinsically imperfect.

**Learning-based Defenses.** Our learning-based mitigation approach assumes that there is a limited number of "classes" of legitimate outbound links. On websites with many different "classes" of links, our technique can yield errors. Although we showed that we could ignore this problem, we believe that our technique may yield better results if applied with pre-processing pipelines, which group outbound links according to their provenance (e.g., per user, per page, or section of the website). This would ensure much more uniform histograms during training and thus increased detection capabilities and fewer errors.

## VII. Future Work

This study is to the best of our knowledge the first attempt to investigate the manipulation of web crawlers by attackers. Yet, future studies could shed more light to this potential threat. In this section we present possible research paths and highlight topics that worth to be further investigated.

**Exploring Other Attacks.** A future research direction will be to extend the attack models beyond the ones described in this paper. The first step is a careful systematization and fingerprinting of the capabilities of the crawlers to understand to which extent they can execute arbitrary client-side code. Then, possibly in cooperation with the search-engine operators, the second step consists in testing a list of increasingly complex attacks to evaluate their feasibility.

**Exploring Other Vectors.** As a continuation of the previous point, exploring other web services that can be used as a trampoline for our attacks is another interesting research direction. The challenge here is that the interaction with such web services could be hard to automate in a generic way. Interesting web services worth exploring include, for instance, link checkers or chat bots that process posted links and security products such as automated scanners used to analyze a website to categorize its content (e.g., Web of Trust).

**Web Frameworks.** Automatic cross-site request forgery protection has now landed in commercial and open-source (CSRF) web application frameworks (e.g., Flask, Django). All the developer has to do is use them, and forms are automatically protected against CSRF. Similarly, we believe that our proposed deterministic solutions can be implemented and included in web development libraries to ensure that all on-premise links are authenticated out of the box.

**Supervised Learning.** An interesting research direction consists in exploring the application of supervised learning techniques in order to incorporate pre-existing domain knowledge in the models that we propose. This may improve the recall of the non-legitimate links, while keeping a high precision. However, given the variety of non-legitimate outbound links that an attacker can craft, applying these techniques requires an ample set of samples as well as a robust set of features to model them.

## VIII. Related Work

Web vulnerabilities [20] are a longstanding problem in the modern digital world. Previously, researchers focused on every aspect of web vulnerabilities, including design-based solutions [22] to minimize the chances of errors in the development phase, code-analysis approaches that try to find [26] and remove [9] application bugs, black box approaches to detect vulnerabilities in embedded web interfaces [6], [7], and runtime defenses that strive to detect and block their exploitation. Although technically simple to understand and implement, the abuse of web crawlers as an indirect "exploiter" of web vulnerabilities is a serious threat that creates a trade-off between "protection" vs. "popularity" in modern websites.

The attacks that we present in this work can be categorized as abuses of public web services for malicious purposes. A recent and interesting work in this direction is [16], which presents a composition-based attack put together by the authors by leveraging benign web services (e.g., Google Docs, Facebook, URL shorteners). The approach presents a series of low-level HTTP primitives that can be created by adding certain URLs, for example, in a Google spreadsheet or Facebook status update, and waiting for the service to crawl such URLs. The net result is that an attacker can send arbitrary HTTP requests while remaining anonymous, well hidden behind the many levels of indirection created through the combination of such services. Interestingly, this method could be leveraged in our attacks to further increase their power.

SURF [17] is a recent work on the detection of web search engine poisoning. The authors study search redirection graphs, obtained with an instrumented browser, and extract robust features that indicate rogue redirections typical of poisoning campaigns. Such features include, for instance, the total redirection hops, the number of cross-site redirections, the presence of page-rendering errors. A system like SURF can be effectively adopted on a global scale by search engine operators to find and hide rogue results. In fact, in our work we focus more on detecting the origin of the indirect attacks described in Section III-C. Regarding the search engine poisoning attacks described in Section III-B, we focus specifically on those that are made possible thanks to the presence of vulnerabilities. Moreover, SURF tackles the problem of detecting *existing* campaigns in general, whereas we analyze the causes of the problem under the specific condition of a vulnerable website that offers the attacker a low-hanging fruit to create such campaigns. In principle, our results could be applied to characterize and detect vulnerability-enabled search engine poisoning campaigns right at the origin.

## IX. Conclusion

In this paper, we explored a new category of attacks, which rely on the fact that search engine bots, or third-party web services in general, trust and follow links that are presented to them. The challenge here is that there is a trade-off between the primary goal of a bot (i.e., explore every corner of the web) and the risk of following a malicious link. The security problem that arises is amplified by the increased computational power demanded by modern websites, which require complex crawling capabilities. We discovered that the most popular crawlers (i.e., GoogleBot, BingBot, YahooBot) are blindly following links that could potentially end up exploiting a vulnerability against the target host. This empowers the attacker with the possibility of hiding her true location when launching an attack. Moreover, the existence of this attack venue creates a delicate and complex issue of responsibility: which party, between website owner and crawler operator, is liable in case a malicious outbound link disrupts a web service? Finally, we proposed countermeasures that can be adopted gradually and independently by each involved party, which, depending on their deployment, can mitigate or altogether eliminate this problem.

## Acknowledgement

The research was supported by the German Federal Ministry of Education and Research under grant 16KIS0327 (IUNO).

## References

[1] M. Balduzzi, C. T. Gimenez, D. Balzarotti, and E. Kirda. Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2011.

[2] A. Barth, C. Jackson, and J. C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *Conference on Computer and Communications Security (CCS)*, 2008.

[3] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In *Applied Cryptography and Network Security*, 2004.

[4] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 1998.

[5] S. Christey and R. A. Martin. Vulnerability Type Distributions in CVE. *Mitre report*, 2007.

[6] A. Costin, A. Zarras, and A. Francillon. Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2016.

[7] A. Costin, A. Zarras, and A. Francillon. Towards Automated Classification of Firmware Images and Identification of Embedded Devices. In *International Conference on ICT Systems Security and Privacy Protection (IFIP SEC)*, 2017.

[8] Django. Django's Cache Framework. https://docs.djangoproject.com/en/dev/topics/cache/#template-fragment-caching, Mar 2015.

[9] A. Doupé, W. Cui, M. H. Jakubowski, M. Peinado, C. Kruegel, and G. Vigna. deDacota: Toward Preventing Server-Side XSS via Automatic Code and Data Separation. In *Conference on Computer and Communications Security (CCS)*, 2013.

[10] M. Goldstein and A. Dengel. Histogram-Based Outlier Score (HBOS): A Fast Unsupervised Anomaly Detection Algorithm. In *German Conference on Artificial Intelligence*, 2012.

[11] A. Hidayat. PhantomJS. http://www.phantomjs.org, Mar 2015.

[12] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open Source Scientific Tools for Python. http://www.scipy.org/, 2001.

[13] A. Klein. Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics. *Sanctum whitepaper*, 2004.

[14] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM (JACM)*, 1999.

[15] C. Kruegel, G. Vigna, and W. Robertson. A Multi-Model Approach to the Detection of Web-Based Attacks. *Computer Network*, 2005.

[16] F. Lu, J. Zhang, and S. Savage. When Good Services Go Wild: Reassembling Web Services for Unintended Purposes. In *USENIX Summit on Hot Topics in Security (HotSec)*, 2012.

[17] L. Lu, R. Perdisci, and W. Lee. SURF: Detecting and Measuring Search Poisoning. In *Conference on Computer and Communications Security (CCS)*, 2011.

[18] F. Maggi, A. Frossi, S. Zanero, G. Stringhini, B. Stone-Gross, C. Kruegel, and G. Vigna. Two Years of Short URLs Internet Measurement: Security Threats and Countermeasures. In *International Conference on World Wide Web*, 2013.

[19] O. A. McBryan. GENVL and WWWW: Tools for Taming the Web. In *International Conference on World Wide Web*, 1994.

[20] Open Web Application Security Project. Top Ten. https://www.owasp.org/index.php/Top_10_2013-Top_10, 2013.

[21] N. Perriault. CasperJS, a Navigation Scripting and Testing Utility for PhantomJS and SlimerJS. http://casperjs.org, Mar 2015.

[22] W. Robertson and G. Vigna. Static Enforcement of Web Application Integrity Through Strong Typing. In *USENIX Security Symposium*, 2009.

[23] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a Very Large Web Search Engine Query Log. *ACM SIGIR Forum*, 1999.

[24] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In *Symposium on Principles of Programming Languages (POPL)*, 2006.

[25] The Register. Hacker Crew Nicks '1.2 Billion Passwords' – But WHERE Did They All Come From? http://www.theregister.co.uk/2014/08/05/russians_amass_1_2bn_stolen_passwords/, Aug 2014.

[26] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross Site Scripting Prevention With Dynamic Data Tainting and Static Analysis. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2007.

[27] W3C. Content Security Policy 1.1. http://www.w3.org/TR/2014/WD-CSP11-20140211/, Feb 2014.

[28] WorldWideWebSize. The Size of the World Wide Web (The Internet). http://www.worldwidewebsize.com/, Apr 2015.

[29] A. Zarras, A. Papadogiannakis, S. Ioannidis, and T. Holz. Revealing the Relationship Network Behind Link Spam. In *Annual Conference on Privacy, Security and Trust (PST)*, 2015.