Exploitable Redirects on the Web: Identification, Prevalence, and Defense

Craig A. Shue
Indiana University
cshue@cs.indiana.edu

Andrew J. Kalafut
Indiana University
akalafut@cs.indiana.edu

Minaxi Gupta
Indiana University
minaxi@cs.indiana.edu

Abstract

Web sites on the Internet often use redirection. Unfortunately, without additional security, many of the redirection links can be manipulated and abused to mask phishing attacks. In this paper, we prescribe a set of heuristics to identify redirects that can be exploited. Using these heuristics, we examine the prevalence of exploitable redirects present in today's Web. Finally, we propose techniques for Web servers to secure their redirects and for clients to protect themselves from being misled by manipulated redirects.

1 Introduction

While browsing the Web, users often find themselves redirected to a URL other than the one they clicked on. Perhaps the most common reason to redirect users is when content moves from one place to another. A redirect can help the users locate the content at its new location. In fact, when done properly, a Web server can redirect Web clients to the new content location without the users even noticing the redirection occurred. Another popular reason site operators use redirects is to track their users' browsing patterns. Normally, a site will not be able to tell which of the external links a user followed from their Web page. However, if the link contains a redirect which causes the user to contact the site's Web server first, the server can record user activity and then provide the URL for the new destination. Another usage allows popular domains to register variants of their domain name and redirect to their original Web site when users mistype the domain name. This approach can protect clients from unrelated sites that register misspelled (*typo-squatted*) variants of the legitimate domain name in order to provide malicious or inappropriate content.

From a security viewpoint, the most interesting category of redirects are the ones that are *open*. Links containing redirects often use parameters to control the destination of the redirect. If the Web server does not check the parameters appropriately before taking action, one can manipulate the destination. This seemingly inconsequential aspect of open redirects has been abused in phishing attacks [1, 2]. For example, the following redirect, `http://example.com/redirect.php?dest=http://1.2.3.4/`, would be open if a phisher could replace `1.2.3.4` with any host name or IP they desired and `example.com` would honor the new URL. A casual Internet user who receives this link, perhaps in an email, would think that she was visiting `example.com` when in reality she was only being redirected by `example.com` to the phisher's domain. Isolating the actual phishing domain from such URLs requires an understanding of open redirects, which many users on the Internet lack. Further, if encoded, it becomes even more challenging to detect these URLs, even for those familiar with redirects.

Phishers can abuse open redirects on trusted sites to mislead users about the site they are visiting. Given this potential for abuse, it is important to understand the prevalence of open redirects in the Web. However, little is known about these redirects, perhaps because no systematic approach currently exists to find them. If Web browsers are able to identify open redirects, they can warn users when they click on such links in phishing emails. This would be an important step forward in the fight against phishing. In the context of open redirects, we make the following contributions in this paper:

Develop heuristics to identify open redirects. To the best of our knowledge, ours is the first work to prescribe a systematic set of heuristics to identify if a link contains an open redirect. This analysis can be performed without contacting the destination of the redirect, which is important in cases of phishing emails where the phisher should not learn about the validity of the user's email address.

Determine the prevalence of redirects in the Web. We use extensive Web crawls to estimate how many Web pages contain links with open redirects.

Propose techniques to mitigate open redirects. We propose both client and server side techniques to limit the harm of open redirects or to close the redirects with little overhead.

We found that a significant proportion of the redirects on the Web are open. From a sweep of 2.5 million Web pages, we found $557,646$ redirect links using our heuristics. Of these redirects, we found that $161,142$ contained the destination of the redirect within the original URL. These redirects, which we call *simple redirects*, can be easily exploited by miscreants; accordingly, we focused on testing if any of these were open. To our surprise, 79%

$(128,058)$ of them were completely open, implying that anyone could manipulate their destination and cause the server to redirect to the manipulated destination instead of the original one. Using another set of heuristics, we were able to pry open another 2% $(2,346)$ of the simple redirects.

Our observations point to the need for securing redirect links on the Web. We review current approaches to secure redirects at the Web servers and propose two additional approaches that do not share the shortcoming of the present-day approaches. To allow clients to protect themselves against servers that fail to secure their redirects, we propose to enhance the Web browsers with the open redirect detection heuristics we developed. If Web browsers have the ability to detect open redirects and warn the users about the risk, open redirects would be less useful to phishers.

The rest of this paper is organized as follows. We discuss our heuristics to detect open redirects in Section 2 and our data collection in Section 3. The results are presented in Section 4. We presents approaches to mitigate open redirects in Section 5. Finally, related work is reviewed in Section 6 and Section 7 presents concluding remarks.

2 Heuristics to Identify Open Redirects

We begin by describing the heuristics we developed to identify open redirects. Given a particular link, the heuristics determine 1) if it contains a redirect, 2) if the redirect is *open*, and 3) if it is not, whether it can be pried open.

2.1 Heuristics to Find Redirects

A variety of mechanisms can be used to implement redirects. The most popular technique, also highly recommended by the World Wide Web Consortium (W3C) [3], is the *HTTP redirect*. It uses the HTTP protocol to redirect users. When a user clicks on a link containing an HTTP redirect, the Web server responds with the URL for the destination of the redirect along with a status code indicating to the user that she is being redirected. A less commonly used approach exploits the *HTML refresh* to redirect users. The refresh capability exists primarily to allow sites whose content changes frequently to specify in the HTML code of their pages how often the pages should be automatically reloaded. A redirect exploiting this mechanism basically leverages the fact that it allows the destination of the reload to be specified. Since this destination could be different from the page the user is visiting, it can be used to redirect users. This mechanism is inefficient in that it causes the user to fetch an extra page containing the HTML refresh redirect. Another less common approach uses client-side scripting, such as JavaScript. When a user clicks on a redirect that uses this technique, the Web server sends a new Web page containing JavaScript which specifies the destination of the redirect. This technique also causes an additional HTML page to be fetched. Additionally, the redirect may not happen if the client browser does not support JavaScript or has it disabled. Due to its pervasiveness, we focus on HTTP redirects in this paper.

Regardless of the mechanism used to implement them, a redirect may be either static or dynamic. Links containing static redirects always lead to the same destination. Since these links cannot be manipulated, they are not interesting from the perspective of potential abuse; therefore, we do not consider them in this paper. The links containing dynamic redirects often embed the destination of the redirect in an optional query string contained in the URL itself[1]. When a user clicks on the link, the browser sends the query string with the request. The Web server reads the parameters contained in the query string to decide the destination of the redirect. Dynamic redirects offer greater flexibility and convenience for the Web site operator, but can be abused if not properly secured. Due to their potential for abuse, we focus on dynamic redirects in this paper.

A typical link containing a dynamic redirect has the following structure. In the URL, `http://example.com/redirect.php?dest=http://1.2.3.4/`, `example.com` is the name of the server the client contacts and `redirect.php` is the script the server runs. The script takes the query string, `dest=http://1.2.3.4/`, as parameters. The query string starts with a "?" character and is frequently composed of a series of name and value pairs delimited by the "&" character. In this example, the value, `http://1.2.3.4/`, is associated with the name `dest`.

Based on this observation, our heuristic to find dynamic HTTP redirects is the following: we search for the presence of a query string in the URLs. In the URLs that contain a query string, we search for protocol prefixes, `http://` or `https://`, which signal the destination of the redirect. One caveat is that query strings are sometimes *URL encoded* in which reserved characters are replaced with special hexadecimal notation. To account for this, we unencode the string before looking for the protocol prefix when searching for URL patterns. We consider the link to contain a potential redirect if a query string is found containing a URL pattern.

Due to the myriad of ways links can be composed, it is necessary to validate if what our heuristics considers a potential redirect is indeed a redirect. First, we use

[1] The destination can also be embedded using the URL path. However, for simplicity, we focus on query strings in this work.

the Perl UserAgent library [4] to access the URL. The library automatically follows HTTP-based redirects, allowing us to determine the URL of the page at the end of the redirection chain. If the URL of the final page is different from the original URL requested, we classify the page as using a redirect. The rest are not redirects. We also note that a redirect does not necessarily mean that the destination of the redirect matches the destination contained in the query string of the URL. One such case where this may happen is when multiple cascaded redirects occur. Since we are interested in determining whether it is possible to manipulate the destination of the redirect for attacks, we focus only on the redirects where the final destination matches the destination contained in the query string of the original URL. Subsequently, we refer to such redirects as *simple redirects*. To find these redirects, we first extract the destination of the redirect from the URL's query string. The destination begins with an `http://` or `https://` and either ends with a "&" character, which marks the beginning of the next key-value pair, or when the URL itself ends. Upon traversing the URL, if the final destination URL matches that contained in the redirect, we test it for openness. For example, if the redirect `http://example.com/redirect.php?dest=http://1.2.3.4/` resulted in the final destination URL of `http://1.2.3.4/`, we scrutinize it further; otherwise, we exclude it from further analysis.

2.2 Heuristic to Find Open Redirects

A redirect is *open* if the destination contained in its query string can be altered and the Web server processing the redirect sends the client to the new location without validation. To test if a simple redirect is open, we replace the destination URL contained in the query string of the redirect with a Web site that we control. On that site, we include a randomly generated character string that is unlikely to appear on other Web pages. (In our tests, we used a 200 character string.) We then follow the link to determine whether it causes the browser to return a page containing the string. If so, we consider the page to be an open redirect.

2.3 Heuristics to Pry Redirects

If a redirect employs weak protections, it may be possible to pry it open. For example, some redirects may employ a checksum for the destination contained in the query string, preventing the redirect from being used if the checksum is incorrect. By altering the checksum along with the destination, one may be able to pry open such redirects. Though exploiting such redirects would require some thought on the part of the attacker, they can be easily exploited by others once an algorithm to open them is developed. We now explain the heuristics we use to test if a redirect which was not open according to the heuristic in Section 2.2 can be pried open.

Redirects which have query strings with only one parameter, the URL of the destination, clearly do not have other query string parameters securing the redirect. If they are not open, they must be using some internal mechanism, such as a white list, to secure the redirect. Such redirects cannot be pried open externally. For such cases, we focus on detecting whether such redirects use white lists containing popular Web sites.

Redirects whose query strings have at least one parameter other than the destination required some thought. If altering the destination and leaving the remaining parameters unchanged failed to open the redirect, the redirect is secured internally or there is at least one parameter the server is testing before redirecting. Even though 90% of the redirects had 4 or few parameters including the destination parameter, varying each to infer which of the parameters could be altered would have been cumbersome. Upon manual inspection, we found that some of the parameters are unlikely to be specific to the destination of the redirect. For example, two of the common parameters were related to language of the page and country of origin of the request. However, many sites will have the same language and country or origin. Instead of trying to infer the intent of all parameters to check which ones were specific to the destination, we tried two very simple strategies: we either dropped all the parameters other than the URL, or altered each of them simultaneously in trivial ways. Specifically, if a parameter was a number, we simply incremented it, and if it was a string, we dropped a character from the string. Doing so essentially only checked if the server was checking anything at all for the altered destination. While one may expect that anything outside of the permitted destinations would be denied if the default case was handled properly, we found quite the contrary: many servers were only allowing the permitted destination with a given set of parameters, but allowed arbitrary destinations when these parameters were altered. We describe these and other results in Section 4.

3 Data Collection Methodology

To find the prevalence of exploitable redirects in the Web, we performed extensive Web crawls using three different data sets. To perform the crawls, we used the Java language to write our own crawler based on Jakarta's Http-Client [5] project. For each data set, we obtained a URL for a top page, downloaded that page and any page linked from that page that was within the same DNS domain as the original page. We examined all the links contained in the top-level pages as well as on the pages we followed using the heuristics described in Section 2. We used the Perl SimpleLinkExtor module [6] to extract all links contained in the HTML tags in the document. An alternative method to extract links would have been to look for the presence of *http://* or *https://* on the Web pages. We used this strategy initially but decided to use the Perl module instead because this strategy was causing us to miss links that used relative addressing.

In our first data set (referred to as `Alexa` subsequently), we investigated the presence of vulnerable redirects on popular Web sites. We used the Alexa Web Information Service [7], which ranks the most popular Web sites on the Internet, to obtain the $1,000$ most popular sites in each of 16 top level categories, as well as the top 500 most popular sites overall. Some sites were present in multiple categories; upon removing duplicates, we found $15,341$ unique Web sites. We used each of the sites obtained from Alexa as starting pages for Web crawling. This crawl resulted in $864,628$ Web pages.

The second data set (referred to as `DMOZ` subsequently) uses links from the DMOZ Open Directory Project [8]. The DMOZ project is a categorized collection of user-submitted links, allowing users to use a directory tree to locate relevant Web sites, rather than use search engines. To obtain a similar number of sites as in the first data set, we randomly selected $16,500$ unique links from an October 23, 2007 snapshot of the DMOZ project to perform our crawl. We obtained $216,812$ Web pages from this crawl.

In the third data set (referred to as `DNS` subsequently), we focused on actual user behavior. To create this data set, we captured all the DNS queries issued on our departmental network for a one-week period. We used the host names contained in the `A` (Address) record queries as the base for URLs for Web crawling. This data contained $164,145$ unique host names. From this crawl, we obtained $1,368,198$ Web pages.

Table 1 shows the number of pages and links contained in each data set.

4 Results: Prevalence of Open Redirects on the Web

Recall from Section 2 that potential redirects specify a destination in the URL's query string (identified by the presence of `http://` or `https://`). When eliminating URLs that fail these tests, between 0.92% and 1.7% of the links remained in our data sets, as shown in Table 1. We note that even links that are not regarded as potential redirects could be involved in redirects; these pages may use a mechanism to obfuscate their functionality. Short of traversing each of them individually, there is no way to find such redirects. Since visiting over 140 million links from our three data sets would have been very time consuming, we simply excluded these cases which did not have a query string or a destination specified. Accordingly, the results we present serve as a strict lower-bound on the actual number of redirects on the Web.

Data Set	Source Pages	Total Links	No Query String	No URL Pattern	Potential Redirects
Alexa	864,628	53,833,400	71.00%	27.92%	1.07%
DMOZ	216,812	5,745,145	70.09%	28.99%	0.92%
DNS	1,368,198	81,186,127	73.07%	25.24%	1.70%

Table 1: Classification of the links extracted from each data set. A total of 815,779 unique potential redirects were found.

Potential redirects totaled $2,007,253$ URLs across the three data sets. Removing duplicates left us with $815,779$ unique links. These span $4,978$ unique domains, and 82 unique TLDs. Validation of these redirects through an actual traversal, as described in Section 2.1, confirmed that $557,646$ (68%) were actual redirects. A further $100,191$ (12%) of the links were broken and could not be retrieved and the rest did not appear to use redirection even though they contained a query string with a URL pattern. We examine this by individual data set in Table 2. We see that a significantly lower portion of potential redirects are actual redirects in the `DMOZ` data set, which is composed of random sites, than in the other data sets.

	Potential Redirects	Actual	Not Redirects	Broken
Alexa	283,001	68.75%	23.01%	8.24%
DMOZ	20,364	58.61%	19.28%	22.10%
DNS	562,118	68.48%	18.17%	13.35%

Table 2: Classification of potential redirects from each data set. A total of 557,646 actual redirects were found.

Recall from Section 2.1 that *simple redirects* are easily manipulated by the attackers. These are redirects where the destination of the redirect is the same as the URL included in the parameters of the query string. Of the actual redirects, $177,284$ (32%) passed this test. From a manual inspection of a small subset of the links that did not match the destination, we found several cases of independent redirection: redirection to site authentication pages, transitions of the protocol from HTTP to HTTPS, search pages, and blog posting pages. While these links do have query strings that contain URL patterns and use redirection, these factors are independent, suggesting these redirects were statically configured even though they looked like dynamic redirects. Since these cannot be manipulated, we exclude these links from testing for open redirects.

Next, we tested if the $177,284$ actual simple redirects were open, as described in Section 2.2. Replacing the destination of the redirect with our custom page was not possible for $16,142$ of the URLs because they used non-standard character encoding; we excluded these redirects from subsequent analysis. Of the remaining $161,142$ entries, $128,058$ (79%) of the redirects were completely open: traversing them caused the server to redirect to our custom page instead of the original one contained in the destination of the redirect. Another $5,108$ (3%) returned an error. From this, it is clear that sites that use parameters to determine the destination of the redirect fail to secure their redirects a vast majority of the time. These results are shown by individual data sets in Table 3. Results across the data sets were similar, with popular `Alexa` sites containing the highest percentage of open redirects. The `DNS` data set had the lowest percentage of open redirects.

	Simple Redirects	Open	Closed	Broken
Alexa	65,012	83.00%	12.97%	4.03%
DMOZ	3,117	81.30%	13.86%	4.84%
DNS	98,138	77.65%	19.65%	2.70%

Table 3: Classification of the 161,142 unique simple redirects from each data set. 128,058 (79%) were *open*.

We then tested if the remaining simple redirects, $27,976$ (17%), could be pried open. As described in Section 2.3, we used three approaches to subvert these protections: alter all parameters simultaneously except the URL (which reflects the new destination), drop all the parameters except the URL, and replace the redirect URL with a popular site possibly on the server's white list (we used `google.com`). To our surprise, $2,346$ (8.4%) could be pried open by at least one of these approaches. Dropping the non-URL parameters was the only effective approach in 965 cases. Altering the non-URL parameters was the only effective approach in 682. In 699 cases, either dropping or altering the non-URL parameters successfully resulted in prying open the redirect. There were no cases where simply changing the URL to a popular site opened the redirect. *In all, simple redirects were either open or pried open in 81% of the cases we examined, yielding a total of* $130,404$ *unique redirect links*.

5 Mitigating Open Redirects

An ideal solution to guard against the risks posed by open redirects would be for Web designers to avoid introducing them in the first place. Short of that, the risks of open redirects can be mitigated by server or client-side modifications. In this section, we describe existing approaches to combat open redirects and propose additional approaches.

5.1 Server-side Modifications

Web servers sometimes modify their redirect pages to warn users that they are being redirected. If detailed information about the transition from one Web site to another is provided to the user, it may alert them about phishing traps. However, with the ubiquity of redirect pages, constant redirect warnings may lead to users simply disregarding them. Some servers use whitelists of approved destinations to ensure their redirects are not misused. This approach requires the site to store a database of valid third-party redirect destinations, which may be extensive in some cases, and requires administrative overhead to keep the list current. Time sensitive tokens have also been used by some sites to limit the usage of a redirect link.

We propose two additional approaches Web servers can use to ensure that their redirects cannot be abused by phishers or other miscreants. The first leverages the `referer` header supplied by the client browser. When following a link from one page, which we call the *source page*, to another page, the *destination page*, the client supplies a `referer` header to destination page. This header indicates the URL of the source page. We propose that the server check the referer header when processing redirects. If it indicates that the client was at another page on the server, the redirect can be processed normally. Otherwise, it can be halted with an error message because it indicates that the client is coming from a third-party site, which could be potentially malicious. Though simple and attractive, this approach has a shortcoming: though popular Web browsers provide this information by default, security suites and other browsers may simply omit the `referer` header.

Our second technique does not depend on the browser behavior. In this technique, the Web site operator creates an authentication token for each redirect. This token is the hash of a concatenation of a secret value for the site and the destination of the redirect. This token is then included as a part of the query string of the redirect. To verify that a redirect processed by the server is legitimate, the server verifies the authentication token contained in the URL by comparing it with a hash of the destination of the redirect and the secret value. In additional to universal client coverage, this approach has an additional advantage: server-side scripts could automatically re-write links to include the authentication tokens before serving pages to clients. This allows for easy incorporation without going through an overhaul of existing Web sites. Further, the low hashing overheads can be further reduced through caching.

5.2 Client-side Modifications

Client-side defenses are necessary in cases when servers fail to protect their redirects. Today, phishing toolbars can detect the final destination of the redirection chain and block access to known phishing sites [9]. However, the phishing site must be known before such toolbars can operate, which does not immediately protect users from deception. Likewise, a blacklist of open redirect Web pages would have similar limitations. Redirect Remover [10], a Firefox browser extension, analyzes links on the page client is visiting and rewrites them to expose the actual destination. Unfortunately, this breaks some of the legitimates uses of redirects. Further, it does not protect against phishing, where the redirected links come from email messages.

We propose that client browsers apply the heuristics we used to identify open redirects. Essentially, the browser would examine the link for any URL patterns in the query string. If a destination is specified, the browser would replace the URL pattern with a test verification Web page. If upon following that link, the client is redirected to the verification page, it has confirmed that the redirect is open. At that point, the browser can either refuse to connect or warn the end-user of the open redirect. This approach has the advantage that the client does not actually follow the redirects, which ensures that the phishing sites cannot confirm that a user attempted to follow a link. Otherwise, the phisher could confirm the destination of a phishing email was valid and include the address in further attacks. Given the prevalence of open redirects on the Web today, the overhead of this approach could harm the user experience. To avoid warning users about inconsequential open redirects they encounter on the Web, the browsers could perform the proposed checks only when a user is redirected from a third party site or application, such as a mailer program or Web-based email sites.

6 Related Work

Web security has many facets, of which open redirects are one. In the interest of brevity, we focus on works directly related to open redirects in this section.

Fette et al. [1] describe open redirects in email as an indication of phishing. They use this to motivate one of their heuristics for detecting phishing emails based on the number of dots contained in URLs in the email. In our work, we focus on the prevalence of redirects on the the Web instead.

Wang et al. [11] analyze Web sites to detect malicious sites that exploit browser vulnerabilities. In doing so, they analyze whether Web site redirects are being used to obfuscate the attack. They find that many sites hosting exploits hide behind redirects. When following redirects, their list of exploit providers grew 263%. In our work, we focus on characterizing the redirects themselves.

¡Netcraft provides a commercial service to check Web sites for open redirects [9]. They additionally provide examples of previously found open redirects. However, they do not provide details of their methods or any information on what they find beyond the small set of motivating examples.

7 Conclusion

Our analysis found that a large proportion of redirects on the Web can be manipulated and exploited. While our analysis is a lower bound on the threat posed by open redirects, it opens several avenues of future work. Specifically, we would like to follow each of the links we find in order to detect redirects we may have missed. We further plan to consider redirects using JavaScript and the HTML refresh approaches. We also would like to explore how complex redirects can be exploited in order to deceive users and suggest methods to thwart these attacks. Finally, the feasibility and overheads of the client and server-side approaches we proposed to defend against open redirects deserve close scrutiny.

Acknowledgments

We would like to thank Gregory Blanton, Pablo Santa Cruz, and Jenett Tillotson for contributing their Web page crawl data, which was used to find redirect links in the project. We would also like to thank Yonghyun Yoon and Dalvinder Singh, who worked on an earlier project that showed the viability of this work.

References

[1] I. Fette, N. Sadeh, and A. Tomasic, "Learning to detect phishing emails," in *International World Wide Web Conference (WWW)*, 2007.

[2] J. Nagle, "SiteTruth - blacklisting collateral damage," in *MIT Spam Conference*, 2008.

[3] World Wide Web Consortium (W3C), "Use standard redirects - don't break the back button!" http://www.w3.org/QA/Tips/reback.

[4] M. Langheinrich, "LWP::Parallel::UserAgent - a class for parallel user agents," http://search.cpan.org/~marclang/ParallelUserAgent-2.57/lib/LWP/Parallel/UserAgent.pm.

[5] Apache Software Foundation, "Jakarta commons HTTP client," http://hc.apache.org/httpclient-3.x/.

[6] B. Foy, “HTML::SimpleLinkExtor - extract links from HTML,” http://search.cpan.org/~bdfoy/HTML-SimpleLinkExtor-1.19/lib/SimpleLinkExtor.pm.

[7] Amazon.com, Inc, “Alexa web information service (AWIS),” 2008, http://aws.amazon.com/awis.

[8] DMOZ, “Open directory project,” http://www.dmoz.org/.

[9] Netcraft, “Netcraft: Anti-fraud open redirect detection service,” http://news.netcraft.com/open-redirect-detection.

[10] Xeen, “Redirect remover,” http://redirectremover.mozdev.org/.

[11] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, “Automated web patrol with Strider HoneyMonkeys,” in *Internet Society Network and Distributed System Security Symposium (NDSS)*, 2006.