A Taxonomy of JavaScript Redirection Spam

Kumar Chellapilla
Microsoft Live Labs
One Microsoft Way
Redmond, WA 98052
+1 425 707 7575
kumarc@microsoft.com

Alexey Maykov
Microsoft Live Labs
One Microsoft Way
Redmond, WA 98052
+1 425 705 5193
amaykov@microsoft.com

ABSTRACT

Redirection spam presents a web page with false content to a crawler for indexing, but automatically redirects the browser to a different web page. Redirection is usually immediate (on page load) but may also be triggered by a timer or a harmless user event such as a mouse move. JavaScript redirection is the most notorious of redirection techniques and is hard to detect as many of the prevalent crawlers are script-agnostic. In this paper, we study common JavaScript redirection spam techniques on the web. Our findings indicate that obfuscation techniques are very prevalent among JavaScript redirection spam pages. These obfuscation techniques limit the effectiveness of static analysis and static feature based systems. Based on our findings, we recommend a robust counter measure using a light weight JavaScript parser and engine.

Categories and Subject Descriptors

D.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.m [**Information Storage and Retrieval**]: Miscellaneous

General Terms

Algorithms, Measurement, Performance, Experimentation, Languages.

Keywords

Web search, web spam, JavaScript, redirection spam.

1. INTRODUCTION

Web spam pages can be broadly categorized into employing boosting and/or hiding techniques [1]. While content and link spam comprise common search engine rank boosting techniques, cloaking and redirection spam are hiding techniques. Among the redirection spam techniques, script based redirection is the most notorious and difficult to catch. Script redirection spam presents spam content to a script-agnostic crawler, but automatically redirects a script capable browser to another URL as soon as the page is loaded.

In this paper, we study common JavaScript redirection spam techniques on the web. A review of redirection techniques is presented in the rest of Section 1. Section 2 briefly presents previous work on redirection spam. JavaScript features that facilitate redirection and hiding are presented in Section 3. We present a data set of URLs and estimate the prevalence of JavaScript redirection spam in Section 4 and Section 5, respectively. Section 6 presents a taxonomy along with representative examples.

AIRWeb'07, May 8, 2007, Banff, Alberta, Canada.
Copyright 2007 ACM 978-1-59593-732-2…$5.00

In this paper, we limit our analysis of script redirection spam to client side scripts that run in the browser. Further, we use the term JavaScript [2] interchangeably with JScript [3], which are the Mozilla Foundation's and Microsoft's implementation of the ECMAScript standard [4].

Modern browsers[1] can be redirected in one of three ways, namely, using HTTP protocol status codes, using a meta refresh tag in the page header, or using a client side script.

1.1 Redirection using HTTP Status Codes

Web browsers typically redirect the user whenever they receive certain HTTP status codes in response to a Get, Post, or Head type of web request. The first digit of the status code defines the class of response. A first digit of 3 represents redirection and indicates that further action must be taken in order to complete the request. These HTTP redirection status codes [5] include

a) 300: Multiple Choices
b) 301: Moved Permanently
c) 302: Found, Redirect
d) 303: See Other, Redirect Method
e) 307: Temporary Redirect

The destination of the URL redirect is provided by the Location header entry of the HTTP response as follows:

```
HTTP/1.1 301 moved permanently
Location: http://www.wikipedia.org/
Content-type: text/html
Content-length: 78
```

Note that when a protocol level redirect is in place, the browser redirects even before any actual page content is downloaded. This is the most efficient way to achieve a redirect wherein user latency and network bandwidth are minimal.

1.2 Redirection using META Refresh

Web pages that use a META tag in the head of the web page, such as the following, can also be used to redirect the user.

```
<meta http-equiv="refresh" content="0;url=http://www.destination.com/">
```

A META refresh tag has two attributes http-equiv and content. The http-equiv attribute being set to "refresh" indicates that the browser is to redirect. The content attribute is typically set to an

[1] such as Internet Explorer, Firefox/Mozilla, Safari, Opera, etc.

integer followed by an optional URL. The integer indicates the number of seconds of delay before the redirect and the URL indicates the destination. The delay and URL are separated by a semi-colon. In the case of a missing URL entry, the current page's URL is used instead. So, rather than redirecting, the web page is reloaded. META refresh based redirection schemes are initiated after downloading some part of the web page, typically the head.

1.3 Redirection using JavaScript

JavaScript based redirection is the most versatile of redirection techniques and usually occurs after the whole web page has been downloaded by the client. Once the page content is downloaded, a JavaScript enabled browser parses the script tags and script attributes present in the web page and starts executing the JavaScript code in stages. Some of the script runs right away, i.e., as soon as the page completes loading, while other pieces run only in response to timer and user events. It is important to note that the executing script code can generate more script code that is injected into the HTML DOM, producing a vicious cycle of self-modifying code whose behavior is hard to predict. Modern web browsers provide a rich event model that allows the page to specify actions. JavaScript present on the page can be executed in response to any of these events. Typical events include document, keyboard, and mouse events:

Document:	onload, onunload,
	onchange, onsubmit, onreset, onselect, onblur, onfocus,
Keyboard:	onkeydown, onkeypress, onkeyup,
Mouse:	onclick, ondblclick, onmousemove,
	onmousedown, onmouseover, onmouseout, onmouseup

1.4 Uses of Redirection

1.4.1 Valid Redirections

There are several legitimate scenarios that require redirection. The most common use is to route traffic while migrating a web site from one host to another. HTTP redirects can be implemented in the web server program itself without altering any web page content. Since the redirect occurs at the protocol level, very little data gets exchanged between the client and the web server.

Web page authors who are not web server administrators may not have sufficient permissions to perform an HTTP redirect. In such scenarios, META refresh redirections come in handy. Legitimate uses of META refresh include reloading the contents of a dynamic web page such as a stock ticker, a database view, or weather map. The reload is typically done every few minutes. It is also common to use META refresh for creating a simple slideshow of web pages by redirecting to the next page every few seconds. META refresh can also be used to support splash pages that display a page for a certain period of time and then move visitors into the site. Splash pages usually also support a "click to enter" link that can be used to skip past the delay. Some web sites also use META refresh to periodically reload ads which can be annoying to users.

Both HTTP and META refresh techniques produce only static redirections. However, JavaScript based redirections can produce dynamic redirections that route traffic more intelligently. For example, you might have multiple versions of a web page designed for different browsers and use JavaScript to check the browser type and redirect the user appropriately. JavaScript can also be used to avoid direct access to pages in a frameset: using the top.frames.length object, you can find out if the page is in a frame or not. In reverse, JavaScript redirection is also useful for breaking a viewer out of someone else's frames if they followed a link that didn't have the target HTML attribute set correctly.

1.4.2 Questionable Redirections

Quick redirections (under 5 seconds) can cause usability problems. Most users cannot hit the "Back" button fast enough, which can be frustrating, especially when the destination page is unavailable. On the other hand, automatically refreshing the page can be confusing to users and spawn security concerns since they did not request the reload.

Search engines vary in their tolerance of META refreshes. AltaVista is known to have banned any web page that used a META refresh with a refresh attribute set to less than 30 seconds. However, presently, many search engines consider anything less than a 5 second delay to be an indication of spam [6-8].

Using redirections to manipulate search engine crawlers with the goal of improving a web site's ranking is considered spam. Domain forwarding and doorway pages are two very common forms of redirection spam.

Domain forwarding can be used to make a common webpage appear more important that it really is. For example, consider a single user's home page on AOL with a URL like http://www.hometown.aol.com/SmallBusiness/. For business, such a simple URL may not inspire confidence in buyers. So, the user purchases a real domain name such as www.SmallBusiness.com and uses a domain forwarding service to redirect traffic. The service immediately redirects visitors who type in http://www.SmallBusiness.com to the real AOL site. This service usually costs a lot less than using a paid Web host. Note that there are some legitimate uses of domain forwarding such as using a short memorable domain name than a long domain name. One such service is offered through www.tinyurl.com.

Doorway pages are used by both legitimate and spam sites to improve rankings for certain search terms. The doorway page is specifically designed and optimized to rank high for certain search terms. Doorway pages can improve user experience by introducing the site to the user and clearly stating what the site is about. However, the problem occurs when the site targets terms that are completely inappropriate to the site's topic. Visitors who search on those terms may click on the doorway page, but then are quickly redirected to a spam site.

1.4.3 Detecting Redirection

Web pages that redirect using protocol status codes and META refresh tags are very easy to detect. Simply disabling auto-redirection on the HTTP requests and checking the returned status codes will catch HTTP redirects. Similarly, issuing HTTP HEAD requests or quickly scanning the HEAD of the HTML page for META refresh tags is sufficient to catch all META tag based redirects. On the contrary, detecting JavaScript redirects is a very challenging problem. JavaScript is a very versatile scripting language and can produce a variety of execution behaviors. You can also implement advanced hiding techniques that will easily fool any simple token based parser or static code checker.

Server side scripting can also be used to perform redirection. Server side redirection is implemented through scripts on the web server. The most common use of server site redirects is to send visitors to a custom error document when they enter an invalid URL. It can also be used to gradually migrate users from an older version of the web site to a newer version. It is technically more demanding than META tags and JavaScript redirects. In this paper, we limit our discussion to client side scripting and redirections.

2. PREVIOUS WORK

Gyöngyi and H. Garcia-Molina [1] present a taxonomy of web spam and describe redirection as a spam hiding technique. Wu and Davison [9] conducted a preliminary study of the distribution of redirection spam among the four types: HTTP 301, HTTP 302, META refresh, and JavaScript onload. They could successfully detect all 301, 302, and META refreshes. However, due to the complicated nature of JavaScript redirects, they limited their approach to detecting location.replace and window.location substrings in the script tags of the web page. Benczúr et al [10] found a number of doorway spam pages which used obfuscated JavaScript code to redirect to their target. They comment that detecting such redirections may already require certain expertise. More recently, Niu et al [11] studied spamming in forums using a "monkey program" [12] to visit each page using a web browser and analyzed network traffic for redirections.

We could not find much previously published literature that directly studied how prevalent, successful, or varied JavaScript redirection is on the web. In this paper, we attempt to study JavaScript redirections in detail. Our findings may also aid spammers, but we hope that our observations and recommendations make it easy to implement countermeasures. The fight against web spam is an arms race. So, techniques identified in the current work are representative of current practices which may become obsolete in the future. However, our recommended countermeasure for identifying script based redirections is robust and will be successful in the long term.

3. JAVASCRIPT REDIRECTION

HTML pages use tags to describe the formatting, layout, and structure of text-based information in a document. Tags exist for specifying headings, paragraphs, lists, interactive forms, embedded images, and other objects. Dynamic HTML (DHTML) pages improve upon simple HTML pages by making them interactive. They achieve this through the use of client-side scripting languages (such as JavaScript), a presentation definition language (Cascading Style Sheets, CSS), and a Document Object Model (DOM).

Browsers that support DHTML expose two objects: window and document. They can be accessed using the DOM and used to retrieve information about the current browser window and the currently loaded HTML document, respectively. When a window is contained in one or more frames (frame/iframe), the window.parent and window.top properties provide access to the immediate- and top-most parent windows.

In the following, we present commonly used JavaScript features that facilitate redirection, time delay, hiding, and dynamic code injection. The treatment is neither exhaustive nor meant to be recommended practice. Further, it is biased towards JavaScript features we commonly encountered while analyzing JavaScript redirection spam. In this regard, it is helpful in understanding several redirection examples that we present in this paper.

3.1 JavaScript Features that Facilitate Redirection

Redirection using JavaScript is achieved by changing the location property on the window or document object.

3.1.1 Location property

One can control which page is loaded into the browser using the JavaScript property window.location. The current web page can be changed to a new URL by setting the location property to the desired URL. If you wished to redirect all your visitors to www2007.org when they arrived at your site you would just need the following script:

```
<script type="text/JavaScript">
window.location = http://www2007.org/
</script>
```

Other variants include

```
document.location = "http://www2007.org/"
location.href = "http://www2007.org/"
location.replace("http://www2007.org/")
```

3.1.2 Time Delay

Delayed redirection can be achieved through several JavaScript time delay functions. Several legitimate script redirections use time delays in conjunction with notifications to walk the user through the redirection and avoid confusion and security concerns. The most common one we encountered is through the use of the setTimeout function.

```
<script>
function delayed_redirect() { window.location = "http://www2007.org" }
</script>
</head>
<body onLoad="setTimeout('delayed_redirect()', 5000)">…</body>
```

3.2 JavaScript Features that Facilitate Hiding

A script is said to employ hiding techniques if its behavior cannot be easily predicted using static analysis. We have observed that legitimate JavaScript redirection rarely needs to use any obfuscation or hiding techniques. Simple static analysis can be used to not only identify whether the web page redirects, but also which page(s) it redirects to. Static analysis can be as simple as the use of well tailored regular expressions, or a light weight JavaScript parser.

Programming languages vary in how amenable they are to static analysis. While the behavior of some programs can be completely determined through static analysis, others make it very difficult if not impossible to do so (Turing's halting problem [13] and similar problems). JavaScript has several features that put it in the latter category. Obfuscation, dynamic code generation, and self-modifying JavaScript code are the three common recipes to creating web pages that completely break static analysis.

3.2.1 Obfuscation and Eval

Simple obfuscation is typically achieved through short variable names with random characters and bad formatting to throw off a human reader. Advanced obfuscation techniques use random looking static data in the script that is decoded to generate code which is finally executed through eval. Obfuscation techniques in programming languages (especially C) have enjoyed a lot of recreational interest [19]. In JavaScript, eval takes a string of JavaScript code and executes it. The string can contain an expression, statement, or sequence of statements. The prevalent misuse of eval in the JavaScript language has caused many programmers to label eval as being evil[2]. The static data is usually

[2] A quick search: http://www.google.com/search?q=eval+is+evil brings up several results.

a long string of seemingly random characters. Decoding is achieved through simple string concatenation, URL unescaping, or custom string deciphering methods.

3.2.2 Dynamic Code Injection

The document.write and document.writeln methods can be used to inject arbitrary HTML expressions into the specified document. They can be used to add new script nodes to the HTML DOM and queue up new script code for execution. Advanced script injection schemes may add hyperlinks or forms and even simulate clicks and form submits. The ability to examine existing script nodes that have not yet been executed and modify or append to them makes it virtually impossible to predict the final behavior of the script without actually executing it.

4. DATA SET

To understand the prevalence and different types of JavaScript redirection spam techniques on the web we adopted a three step procedure. First, we sampled a large collection of web pages from the web. Second, we tested them for JavaScript redirection spam. In the third and final step, we manually analyzed a random subsample to find commonly used techniques such as obfuscation and dynamic code injection.

4.1 Labeling

Each web page was downloaded and tested for JavaScript redirection using the procedure outlined in Figure 1. The web page was loaded once in a browser with JavaScript disabled and a second time with JavaScript enabled. After loading the page in the browser and allowing all redirections (if any) to complete, the destination URL (DstURL) was noted. If the destination URL for the script enabled browser, DstURL(E), was the same as the original URL, then the page was labeled as using no redirection. Similarly if the destination URL for the script disabled browser, DstURL(D), was the same as DstURL(E) the page was also labeled as not using JavaScript redirection. If DstURL(D) $\neq$ DstURL(E), then the page was labeled as using JavaScript redirection: URL $\Rightarrow$ DstURL(E). These two URLs were compared to determine if the JavaScript redirection was suspicious. Simple rules were used to eliminate common non-spam redirections. These included examples such as the following:

```
www2007.com => www2007.com/
www2007.com => www2007.com/{index,default,…}.{htm,html,asp,aspx,php,…}
```

JavaScript redirections within the same host were also labeled as benign. All remaining JavaScript redirections were labeled as JavaScript redirection spam. The above procedure is approximate but was chosen as it is relatively straight forward to understand and implement.

4.2 Sampling

Our initial goal was to understand how often JavaScript redirection was being used on the web. However, a random sampling approach did not yield enough samples[3]. Based on sample URLs and corresponding domains that contained significant concentrations of JavaScript redirection, we adopted two biased sampling methods that were based on web page popularity and blogs. The former is also motivated by previous work that showed that popular pages are more likely to contain web spam [14]. Blogging sites provide a shared authoring environment that can become an attractor for web spam.

Popularity based sampling was achieved as in [14]. A list of the top 5000 most popular English queries and their corresponding top 200 search result URLs were collected using Live Search (search.live.com). This resulted in a set of 782,937 unique URLs that we labeled as being popular.

Based on our initial sampling, we chose blogspot.com as the domain to explore JavaScript redirection spam in blogs. Others have also indicated the high incidence of redirection spam in Blogspot [11,15]. Using the top 100 most monetizable keywords from Live Search (see [14]), we extracted 934,876 blog sites of the form <name>.blogspot.com, where <name> contained one or more of the keywords as a substring.

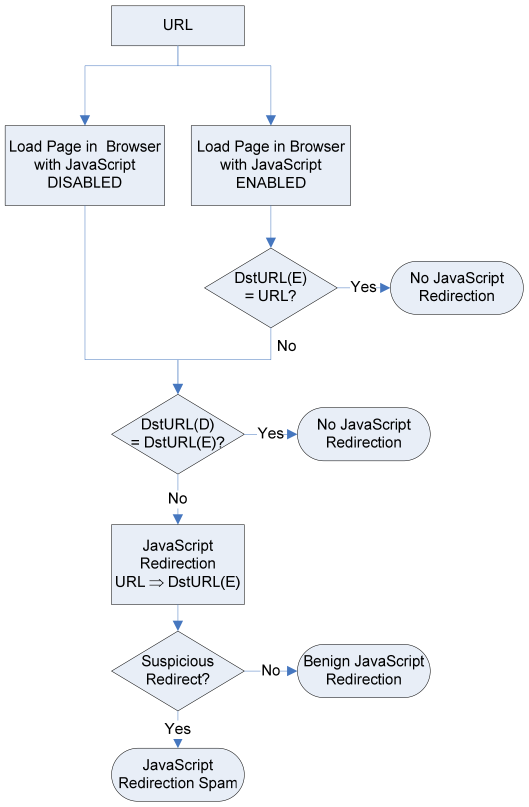
URL
Load Page in Browser with JavaScript DISABLED
Load Page in Browser with JavaScript ENABLED
DstURL(E) = URL?
Yes
No JavaScript Redirection
No
DstURL(D) = DstURL(E)?
Yes
No JavaScript Redirection
No
JavaScript Redirection URL ⇒ DstURL(E)
Suspicious Redirect?
No
Benign JavaScript Redirection
Yes
JavaScript Redirection Spam

Figure 1. Labeling JavaScript Redirection Spam

5. PREVALENCE OF JAVASCRIPT REDIRECTION SPAM

The 782,937 popular URLs and 934,876 Blogspot URLs were downloaded and tested for JavaScript redirection spam. Table 1 presents the number of pages with JavaScript redirection spam that were detected.

Table 1. Percentage occurrence of JavaScript redirection among popular and blogspot.com pages

URL Type	Count / Total	Percentage
Popular	2,712 / 782,937 = 1 in 288	0.35%
Blogspot	7,196 / 934,876 = 1 in 130	0.77%

[3] Randomly sampling 100,000 URLs from a search engine's index (http://search.live.com) produced only a few tens of URLs using JavaScript redirection.

In comparison with popular pages, Blogspot pages are more than twice as likely to contain JavaScript redirection spam. However, in both cases, the occurrence of JavaScript redirection is less than 1 in 100. In [17], the average amount of spam in English web pages was observed to be around 15%. Based on this, one can very approximately estimate that about 2.3% (1 in 43) of popular English spam pages and 5.1% (1 in 19) of Blogspot English spam pages employ some form of JavaScript redirection.

6. JAVASCRIPT REDIRECTION SPAM TECHNIQUES

In the following section, we examine current JavaScript redirection techniques used by spammers and attempt to categorize them based on their features. Along with descriptions of the categories, we present real short examples from real web pages that serve to effectively communicate how advanced some of the techniques are. The original source URL is presented at the bottom of each example. All examples are taken from those reported in Table 1. They were chosen to be representative samples from different categories. In the interests of saving space, we biased the selection of representative samples in favor of succinctness. Simple techniques are presented first and are followed by more advanced ones, with simplicity being analogous to ease of detection (through static analysis). After describing these techniques, we present quantitative results on their prevalence.

6.1 Plain

Plain JavaScript redirection is the simplest form we encountered. The web page redirects the browser by directly changing the location property (see Section 3.1.1) or uses an innocuous conditional to redirect the user as in this example:

```
var1=24;  var2=var1;  if(var1==var2)
document.location="http://www.topsearch10.com/search.php?aid=59731&q=bad
+credit+auto+loan";
                    http://bad-credit-auto--loan.blogspot.com/
```

6.2 String Manipulation and Eval

JavaScript redirection requires an eventual change of the location property. In order to avoid presenting the whole destination URL as a single string, a script can incrementally build up the URL from pieces using string concatenation. Once the URL has been pieced together, a simple call to eval can be used to set the location property. One very straight forward example is presented below. It is interesting to note that even in this simple example eval is repeatedly applied in a for loop to retrieve each of the "a" variables. This appears to be designed to throw off static parsers that do not have the ability to aggregate constants in a JavaScript program.

```
var a1="win", a2="dow.", a3="loca", a4="tion.", a5="replace",
a6="('http://www.partypoker.com/index.htm?wm=2501068')";
var i,str="";
for(i=1;i<=6;i++){
   str += eval("a"+i);
}
eval(str);
                    http://party-poker-bonus.cjb.net/
```

6.3 Unescape

Simple string manipulation to build up URL strings exposes certain features such as substrings of "window.location" or "location.replace" etc. This would allow such feature based systems to have partial success in detecting string manipulation. Obfuscation can be used to avoid discovery through direct checks for substrings. URL encoding or percent-encoding is a common mechanism intended for encoding reserved characters such as !*'();:@&=+$,/?%#[] etc and binary data for script arguments. However, the standard [18] discourages encoding for regular characters, but without restricting the use of encoding to reserved characters. Thus, unreserved characters such as "A" can be represented as "%7E" and are expected to be processed correctly. This feature is exploited by the following script for redirection purposes.

```
var s =
'%5CBEOD%5C%05GDHJ_BDE%16%0CC__%5B%11%04%04%5C%5C%5C%05
SMYNNFD%5DBNX%05HDF%04%0C';
var e = '', i;
eval(unescape('s%3Dunescape%28s%29%3Bfor%28i%3D0%3Bi%3Cs.length%3Bi%2
B%2B%29%7Be%2B%3DString.fromCharCode%28s.charCodeAt%28i%29%5E43%29
%3B%7D%3Beval%28e%29%3B'));
http://freegayporntodays.blogspot.com/2006_10_01_freegayporntodays_archiv
e.html
```

Expanding the unescape produces the following which becomes the argument for eval

```
s=unescape(s);
for(i=0;i<s.length;i++){e+=String.fromCharCode(s.charCodeAt(i)^43);};
eval(e)
```

which in turn takes us to decoding custom strings. The for loop decodes the unescaped string (e) which is then evaluated to

```
window.location='http://www.xfreemovies.com/'
```

6.4 Decode

URL encoding makes it difficult for humans to read the URL substrings. However, the original substring patterns present in the unencoded function argument are left unchanged except for syntactic substitutions. Thus a machine learning system [16, 17] can just as easily learn to observe these URL encoded substring patterns as the original patterns. To break this possibility, several spammers appear to employ custom decoding schemes like the one above (Section 6.3). Two interesting examples of custom decoding are presented below:

```
var tt, kk="", mm;
tt="w|nd^w$l^c#[|^n;'([[*)!!*r^l|^n$|nf^!f>>d!s>#rc($*(*]q;c(>#*+c|g#r>[>s'";
for (i=0; i<tt.length+1; i++)
{
   mm=tt.substring (i,i+1);
   if (mm=="(") mm="h";   if (mm=="*") mm="p";   if (mm=="!") mm="/";
   if (mm==">") mm="e";   if (mm=="$") mm=".";   if (mm=="[") mm="t";
   if (mm=="#") mm="a";   if (mm=="^") mm="o";   if (mm=="]") mm="?";
   if (mm=="@") mm="k";   if (mm=="{") mm="&";   if (mm==")") mm=":";
   if (mm==";") mm="=";   if (mm=="|") mm="i";   if (mm==" ") mm="+";
   kk=kk+mm;
}
eval (kk);
http://cheap-cigaretes-2007.blogspot.com/2006/11/cheap-cigarette-online-
cheap-cigarette.html
```

```
function Decode(){
  var temp="",i,c=0,out="";  var
str="60!115!99!114!105!112!116!32!108!97!110!103!117!97!103!101!61!34!74!9
7!118!97!83!99!114!105!112!116!34!62!13!10!32!32!32!32!32!118!97!114!32!97
!49!61!39!119!105!110!100!39!44!32!13!10!32!32!32!32!32!97!50!61!39!111!11
9!46!108!111!99!97!39!44!13!10!32!32!32!32!32!97!51!61!39!116!105!111!110!
46!114!101!39!44!13!10!32!32!32!32!32!97!52!61!39!112!108!97!99!39!44!32!1
3!10!32!32!32!32!32!97!53!61!39!101!40!34!104!116!116!39!44!32!13!10!32!32!
32!32!32!97!54!61!39!112!58!47!47!39!44!32!13!10!32!32!32!32!32!97!55!61!39
!117!108!116!114!97!45!110!101!116!46!39!44!32!13!10!32!32!32!32!32!97!56!
61!39!105!110!102!111!47!116!105!99!107!101!116!47!115!101!97!114!39!44!3
2!13!10!32!32!32!32!32!97!57!61!39!99!104!46!112!104!112!63!39!44!32!13!10!
32!32!32!32!32!97!49!48!61!39!113!61!39!44!32!13!10!32!32!32!32!32!97!49!49
!61!39!67!104!101!97!112!43!65!105!114!102!97!114!101!34!41!39!59!13!10!32
!32!32!32!32!118!97!114!32!105!44!115!116!114!61!34!34!59!32!102!111!114!4
0!105!61!49!59!105!60!61!49!49!59!105!43!43!41!32!123!32!115!116!114!32!43
!61!32!101!118!97!108!40!34!97!34!43!105!41!59!32!125!13!10!32!32!32!32!32!
101!118!97!108!40!115!116!114!41!59!13!10!60!47!115!99!114!105!112!116!62!
";
  l=str.length;
  while(c<=str.length-1) {
    while(str.charAt(c)!='!')
      temp=temp+str.charAt(c++);
    c++;out=out+String.fromCharCode(temp);temp="";
  }
  document.write(out);
}
                    http://cheap-airfare-a.blogspot.com/
```

6.5 Script Injection

The examples presented so far do not modify the DOM while executing the redirect. As described in Section 3.2.2, executing script code can inject more script code and queue up new script instructions for execution. The following example uses a combination of URL encoding to represent binary data that is custom decoded to build a script tag with redirection code.

```
var s,q,e,d,i;s=String.fromCharCode;q='script'+s(62);
var e =
unescape('%BD%AD%BF%EE%BA%ED%F3%BA%A7%A0%BB%F6%E2%E1%B8%A7%
A6%FC%A5%B1%B9%AD%AE%A7%BB%B5%BA%FC%B0%BB%A6%E3%AC%AA%9B
%A2%B0%B1%B8%B1%B9%E3%F2%BD%A0%A5%A3%B1%B6%E9%FA%F5%FA%F8
%E9%A7%EC%BA%A7%A0%BB%E9%FE%8F%EA%E2%97%F7%E1%92%BB%A3%BF
%A7%E2%B3%B9%A7%B7%B5%A2%E2%BE%BA%AE%A2%FC%B5%BC%A7%B8%A5
%BD%E0%AC%BF%BC%F7%E1%92%BF%AD%A6%BA%AE%AA%F3%FE%B6%E9%AE
%BF%AE%AF%BF%B5%FD%B6%EE%B0%A4%AF%B8%A3%AA%BE%A5%E9%B7%FA
%A7%A3%AE%AF%BB%B9%BE%BC%EE%A1%F0');
var d = '';
for(i=0;i<e.length;++i) d+=s(e.charCodeAt(i)^(((i%10)+203)&255));
document.write(s(60)+q+d+s(60)+'/'+q);
                  http://pori-chudai.igotclicks.com/taktaz
```

The decoded script tag which is injected using document.write into the page is

```
<script>var u="http://www.veracitek.com/adTracker/?source=1976&w=
http%3A%2F%2Fpori-chudai.star-gossip.com%2Ftaktaz",e=escape,d=document;
d.location=u;
</script>
```

6.6 HTML Element and Form Injection

Similar to injecting script nodes into the DOM, one can also inject specific HTML elements and even forms. Here is one example that injects a form into the web page:

```
document.write('<form id="f" method=post action="h' + 't' + 't' + 'p://' + 'qblz.c' +
'om/pc/i' + 'n.cgi?2&parameter=free%20ringtones" style=display:none><input
type=submit name="xlt2"></form>');
document.forms.f.xlt2.click();
                 http://downloadfree-ringtones-.blogspot.com/
```

6.7 Event Injection (Click/Submit)

Creation of such dynamic HTML content is usually followed by a click or submit event that is injected. Here is an example that creates a hyperlink and immediately injects a click event.

```
var lnk='<a id="rdr" href="http://fairsearch.net/rd/find.php?q=';lnk+=kwd;
lnk+='"> </a>';
document.write(lnk);
var obj = document.getElementById("rdr");
try{ obj.click();
} catch (MyError){ obj.click = function() {
  document.location.href = this.href; }
  obj.click();
}
http://live-sex-52817.blogspot.com/2006/09/free-live-sex-chat-nude-strip-
cam.html
```

6.8 Referrer Examination and Traffic Counters

The goal of redirection spam pages is to rank themselves higher than they deserve in search engine results pages for specific queries. Once this has been achieved, the received traffic is routed as desired. Owing to the differences in search engines and their indexing and anti-spam strategies, redirection spam pages will have different degrees of success on different search engines. We found that many of the JavaScript redirection spam pages examine the document.referrer property to take different actions for different referrering pages. The following example demonstrates one such case:

```
r = document.referrer;
if(r &&
  r.indexOf("google")>0 || r.indexOf("yahoo")>0 || r.indexOf("msn")>0 ||
  r.indexOf("live")>0 || r.indexOf("search.blogger.com")>0 ||
  r.indexOf("ask.com")>0){
  if(r.indexOf(document.domain)<0 && r.indexOf("link%3A")<0 &&
    r.indexOf("linkdomain%3A")<0 && r.indexOf("site%3A")<0) {
document.location.replace('http://lipster.net/redirect.php?blogId=14085&addkey
=bBI&ref='+escape(r)+'&dom='+escape(document.domain)+'&blog_url='+location.
href);
}
else{
  if ((r=="" || r==null) && parent.frames.length > 1){
document.location.replace('http://lipster.net/redirect.php?blogId=14085&addkey
=bBI&ref='+escape("from frame:
?q=UNKNOWN_KEYWORD")+'&dom='+escape(document.domain)+'&blog_url='+lo
cation.href);
  }
  else{
    document.location.replace ('http://follar-sexo-conocer.com/partner');
  }
}
                    http://redfreenatio128.blogspot.com/
```

7. PREVALENCE OF DIFFERENT TECHNIQUES

In order to study the prevalence of the variety of techniques described in Section 6, we randomly sampled and studied individual JavaScript programs from Table 1. We randomly sampled 175 JavaScript redirection spam pages from the 2,712 popular pages. Similarly, we sampled another 175 JavaScript redirection spam pages from the 7,196 detected Blogspot pages.

Each sampled JavaScript redirection spam page was manually analyzed. We used a simple JavaScript parser/editor to understand the static structure and exploits used by the scripts. Using a JavaScript runtime engine, we also deobfuscated and executed each program one instruction at a time to understand its runtime behavior and exploits. Based on this analysis, we labeled each of the 175 samples in both categories with the types of exploits they used.

Figure 2 depicts the percentage of JavaScript redirection spam pages using each of the techniques. The Advanced/AJAX category represents web pages that used a lot of JavaScript code (hundreds of lines) or were using AJAX [20] features.

The popularity ranking of different techniques is similar between the popular and Blogspot pages with minor differences. It is interesting to note that popular pages are almost twice as likely to use plain redirection than Blogspot pages. We were surprised to find that nearly 62% (108/175) of the Blogspot pages and 44% (77/175) of the popular pages used some form of obfuscation (Unescape or Decode). Further, almost a quarter (25% for Blogspot and 23% for popular) of all JavaScript redirection spam pages examine the originating URL of the browser (document.referrer property) before redirecting the user. Dynamic code injection through the addition of script and form elements occurred in 25% (44/175) and 9% (16/175) of the Blogspot and popular pages, respectively. Advanced use of JavaScript made up less than 10% of the pages.

Figure 3 presents a histogram of the number of different JavaScript techniques used by popular and Blogspot redirection spam pages. Over 40% of popular redirection spam pages use only a single technique. However, about 85% of Blogspot pages used at least two different techniques. The average number of different JavaScript techniques used was 1.83 and 2.64 for popular and Blogspot JavaScript redirection pages. While none of the popular pages used more than 4 different techniques, some Blogspot pages used as many as 7 (of 12 possible) different techniques.

8. DISCUSSION

JavaScript redirection spam is one of the most notorious redirection spam techniques. It exploits script weaknesses common among today's crawlers to redirect unsuspecting users to spam sites.

Current antispam efforts [16, 17] rely on static sources of information such as URLs, domain names, domain IPs, distribution of page content, distribution of links, etc. Statistical and machine learning techniques are effective against such static web pages. However, JavaScript adds a new level of complexity which requires understanding program behavior before it is executed in a browser. With the increase of dynamic content on the web, crawlers and search engines cannot afford to be script-agnostic.

The adversarial nature of web spam produces an arms race wherein the spammers try to be one step ahead of the anti-spam techniques by finding new exploits. The anti-spam community attempts to counter by reacting as quickly as possible to new spam techniques. Our analysis of the current JavaScript redirection spam techniques shows that JavaScript obfuscation is very prevalent in JavaScript redirection spam. These obfuscation techniques limit the effectiveness of static analysis and static feature based systems. Machine learning based systems that exploit statistical features will only be partially effective and may need to be retrained often. Even if static analysis techniques can detect whether a page redirects or not with high probability, they will not be able to determine the destination URL. The difference between a benign redirect and a suspicious redirect requires a system to precisely determine the destination URL. Further, advanced dynamic code and event injection techniques require dynamic analysis.

Based on our findings, we advocate the use of light weight JavaScript parser(s) and a tuned JavaScript execution environment for predicting the redirection behavior of web pages containing JavaScript. Such a system would be robust to many types of JavaScript exploits. Further, it would not only be able to detect whether a page redirects or not, but also be capable of extracting the destination URL(s). A deeper analysis can also address page redirection behavior in response to certain events such as time delays and mouse moves. Since the only intended purpose of the parser and script execution environment is to detect JavaScript redirection spam, we believe that such a system can be tuned for high performance and throughput. We plan to explore such high performance JavaScript redirection detection systems in future work.

9. ACKNOWLEDGMENTS

We would like to thank Chau Luu for her help in our labeling efforts and the anonymous reviewers for their valuable feedback.

10. REFERENCES

[1] Z. Gyöngyi and H. Garcia-Molina (2005), "Web spam taxonomy," First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb), Japan, 2005.

[2] Mozilla Foundation, "About JavaScript," Online at http://developer.mozilla.org/en/docs/About_JavaScript

[3] Microsoft, "JScript Reference," Online at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/jscript7/html/jslrfjscriptlanguagereference.asp

[4] Ecma International, "ECMAScript," Online at http://www.ecma-international.org/publications/standards/ECMA-262.htm

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1," Online at ftp://ftp.isi.edu/in-notes/rfc2616.txt

[6] Google, Inc. Google information for webmasters, 2007. Online at http://www.google.com/webmasters/faq.html.

[7] Yahoo! Inc. Yahoo! Help - Yahoo! Search, 2007. Online at http://help.yahoo.com/help/us/ysearch/deletions/.

[8] Microsoft, Inc. Live Search Site Owner Help, 2007. Online at http://help.live.com/help.aspx?project=wl_webmasters

[9] B. Wu and B. D. Davison (2005) "Cloaking and Redirection: A Preliminary Study," First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb), Chiba, Japan, 2005.

[10] A. Benczúr, K. Csalogány, T. Sarlós, M. Uher, "SpamRank -- Fully Automatic Link Spam Detection," First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb), Chiba, Japan, 2005.

[11] Y. Niu, Y. Wang, H. Chen, M. Ma, and F. Hsu, "A Quantitative Study of Forum Spamming Using Context-based Analysis," Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, February, 2007.

[12] Y. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities," In Proc. Network and Distributed System Security (NDSS) Symposium, February 2006.

[13] A. Turing, “On computable numbers, with an application to the Entscheidungsproblem,” Proceedings of the London Mathematical Society, Series 2, 42 (1936), pp 230-265.

[14] K. Chellapilla and M. Chickering, “Improving Cloaking Detection using Search Query Popularity and Monetizability,” Second Intl. Workshop on Adversarial Information Retrieval on the Web (AIRWEB’2006), Seattle, USA

[15] C. Pirillo. 2005. Google: Kill blogspot already!!! available online at http://chris.pirillo.com/blog/archives/2005/10/16/1302867.html.

[16] D. Fetterly, M. Manasse, and M. Najork (2004), “Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages,” In Proceedings of WebDB, pages 1-6, June 2004.

[17] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly (2006), “Detecting Spam Web Pages through Content Analysis,” In Proceedings of the World Wide Web Conference 2006 (WWW'06). Edinburgh, United Kingdom, May 23-26, 2006.

[18] T. Berners-Lee, R. Fielding, and L. Masinter, “RFC 3986: Uniform Resource Identifier (URI): Generic Syntax,” online at http://www.ietf.org/rfc/rfc3986.txt

[19] International Obfuscated C Code Contest, http://www0.us.ioccc.org/main.html

[20] Asynchronous JavaScript and XML (AJAX) programming. http://en.wikipedia.org/wiki/AJAX

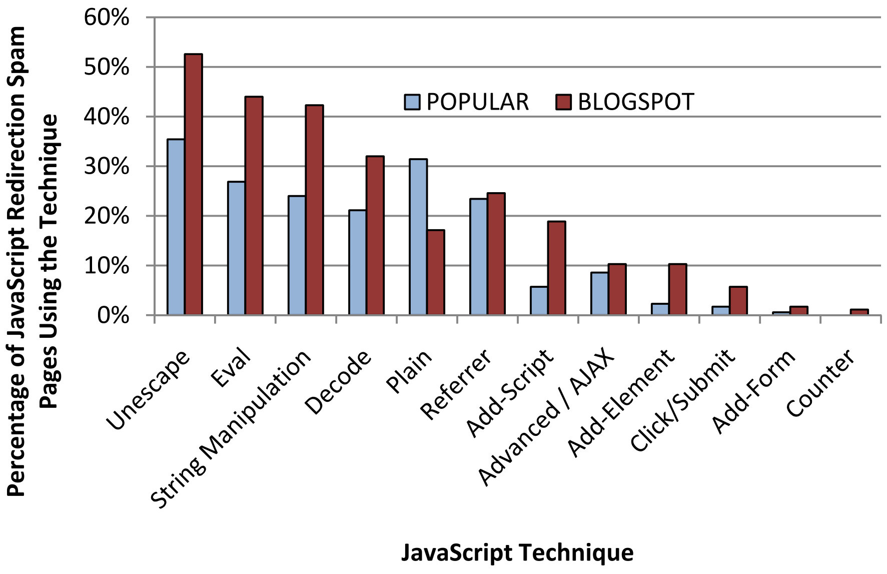

Figure 2. The distribution of JavaScript techniques used in JavaScript Redirection Spam. Note that each webpage sample can use more than one technique, thus the percentages do not add up to 100. The mean number of techniques used by popular and blogspot pages was 1.83 and 2.64, respectively. Thus, they add up to 183% and 264% for popular and blogspot pages, respectively.

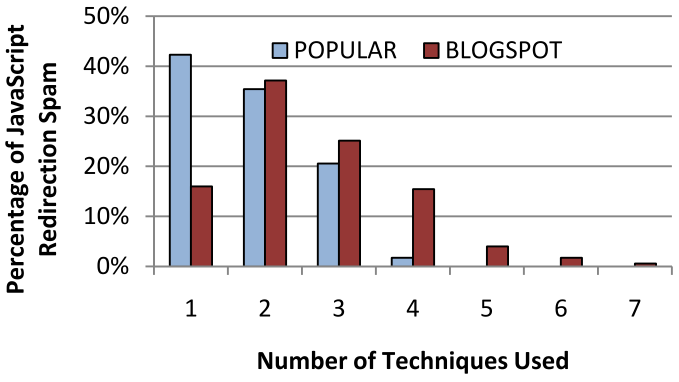

Figure 3. The distribution of the number of different JavaScript techniques used in JavaScript Redirection Spam. The mean number of techniques used was 1.83 and 2.64 for popular and blogspot JavaScript Redirection Spam pages, respectively.