2012 IEEE/ACIS 11th International Conference on Computer and Information Science

An Analysis of URLs Generated from JavaScript Code

Jingyu Zhou
School of Software
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240, P.R. China
zhou-jy@cs.sjtu.edu.cn

Yu Ding
School of Software Engineering
University of Science and Technology of China
166 Ren'ai Road, Suzhou 215123, Jiangsu, P.R. China
sea10548@mail.ustc.edu.cn

***Abstract*—Search engines use a crawling system to recursively download web pages, analyze HTML pages, and generate a new list of URLs to crawl. As web pages are becoming more dynamic than before, JavaScript is heavily used, which poses a great challenge for the crawling system, because now many URLs are embedded in the JavaScript code and are invisible to the crawler. Worse, there is no study on the usage patterns of these URLs and the impact of JavaScript-generated URLs is unknown.**

We propose a browser emulation method to study the usage of URLs from JavaScript code. In order to find these URLs, we instrument a browser core to output all URLs inside a web page, including those generated from JavaScript. Then we classify these URLs into a number of types and study reasons that web developers put them in JavaScript. We analyze top Internet sites and popular web pages. The results show that more than half of them contain URLs generated from JavaScript, which accounts for about 6-19% of total URLs. Among them, 26-41% refer to potential important contents that should be indexed by search engine crawlers, and advertising URLs are about 26-35%.

***Keywords*-JavaScript, URL, browser emulation, web crawler**

I. Introduction

In 2007, Greg Bulmash [2] tested whether Google will index contents generated by JavaScript. By putting JavaScript code on a page and later checking the Google search results, Bulmash was able to determine that Google didn't index the JavaScript-generated contents. In fact, Google [7] indicates that "links within JavaScript will likely be inaccessible to" search engines and those links should be kept outside JavaScript or be replicated in a `noscript` tag.

As web pages become more dynamic, the usage of JavaScript has been more prevalent. A recent study found that all top 100 sites and 89% of top 10,000 sites use some JavaScript [13]. A natural question is how crawler-friendly are the current web pages. Because if web developers write JavaScript code without such considerations, URLs embedded in the JavaScript code cannot be indexed by search engines.

Previous work has studied finding more URLs for crawlers from toolbar logs of browsers [1], from social bookmarking sites [8], and from the hidden web [12, 11, 10]. However, none has studied the problem of extracting URLs from JavaScript codes.

In this paper, we try to identify the usage of URLs embedded in the JavaScript from popular web sites and popular web pages. Specifically, we construct a browser emulation engine that can automatically visit web pages. We run this program twice: one with JavaScript disabled and the other with JavaScript enabled. After the web page finishes loading, the program traverses DOM tree and discovers all URLs within the page. By comparing the two result sets, the program produces the additional URLs that are induced by JavaScript. We further studied usage patterns of URLs from JavaScript with two datasets. This paper has made the following contributions:

- We have designed and implemented a browser emulation scheme to extract URLs generated by JavaScript and have tested our tool on top Internet sites and popular web pages.
- We provide the first analysis on the usage patterns URLs from JavaScript. Our results show that web developers has used this technique for displaying volatile contents, providing better user interface, and hiding information. Experimental results also show that advertisement, recommendations, and navigation bars are the most frequently used patterns.

The rest of the paper is organized as follows. Section II discusses the background and challenges in analyzing URLs embedded in web pages. Section III presents our approach for finding URLs in JavaScript code and our analysis of these URLs. Section IV evaluates the distributions of JavaScript-generated URLs from top Internet sites and popular web pages. Section V discusses related work. Finally, Section VI concludes the paper with future work.

II. Background and Challenges

JavaScript is a widely used web programming language for every major browsers and is standardized in ECMA specifications [5]. Most JavaScript code operates in an HTML page of a browser context with access to HTML DOM and browser objects, which causes significant challenges to the analysis on the control flow and data flow [15, 9].

978-0-7695-4694-0/12 $26.00 © 2012 IEEE
DOI 10.1109/ICIS.2012.28

There are two common ways for JavaScript to dynamically generate new content on a web page. One is using `document.write()` method, which dynamically inserts an HTML segment to the place where the function is called. Alternatively, one could directly set the `innterHTML` of a DOM element.

JavaScript codes may be included inline (i.e., inside HTML) or separately in another file, which can be statically specified in HTML or be dynamically specified by JavaScript code. Figure 1 illustrates an example of insertion of dynamic content downloaded from web servers, where an Ajax request retrieves content during runtime. More complex scenarios involve JavaScript injects a `script` tag into DOM to trigger the download of a remote JavaScript file, which then inserts new content when being executed.

```
$J.ajax({
  type: 'POST',
  dataType: 'json',
  url: '/home/json_home',
  success: function(content) {
    $J('#home_left').html(content.lenses.html);
  }
});
```

Figure 1: An example of dynamic insertion of HTML content by making an AJAX request, which utilizes jQuery for constructing the request.

We summarize the challenges for analyzing URLs embedded in JavaScript as follows:

- The dynamic nature of JavaScript makes static analysis difficult. As shown in Figure 1, real contents can only be found after the execution of JavaScript code and downloading from remote servers. This process may be recursively executed several times before the real content is revealed.
- Web pages are frequently updated. For instance, news, blogs, tweets are often updated in terms of seconds. Another example is rotating ads displayed on web pages. As a result, the analysis is always shooting at a moving target.
- The usage of JavaScript libraries complicates the analysis. There are a number of JavaScript libraries commonly used by web applications, such as jQuery [1] and Prototype [2]. A recent study [14] found that 44 of 100 top sites used a public-available library, and 7 sites used more than one simultaneously. As Jensen et al. [9] pointed out, we need better techniques to handle commonly used JavaScript libraries.
- URLs can be in the form of a JavaScript function. The hyper-reference of an HTML anchor can be a JavaScript function, which may or may not cause a browser to fetch a web content when clicked, e.g., modifying DOM in certain ways.

[1] http://jquery.com/
[2] http://www.prototypejs.org/

Previous work [15, 9] has been focusing on statically analyzing JavaScript, but static analysis of JavaScript is difficult for several reasons. First, the JavaScript language (actually the ECMAScript) contains a standard library of 161 functions and many objects that all need to be modeled by any static tools. Second, functions such as `eval`, `setTimeout`, and the `Function` construct allow dynamic code construction from text strings, which often complicates static analysis. Third, static analyzer needs to model HTML DOM and browser API correctly. As a reference, TAJS [9] implements 250 abstract objects with 500 properties and 200 transfer functions, which is still less than what a real browser does.

In this paper, we choose a dynamic approach for extracting URLs embedded in JavaScript. By executing JavaScript code in a browser context, our approach can leverage browser's implementation of JavaScript interpreter and objects, thus generating precise results. The details of our approach are described in the next section.

III. Method

Figure 2 illustrates our method for uncovering URLs embedded in JavaScript code. Given a URL input, we load the page into a browser engine with JavaScript enabled and a second time with JavaScript disabled. Then URLs in the page are output and compared, thus generating the set of URLs from JavaScript code. Finally, these URLs are classified manually to different types, such as advertising and navigational URLs.

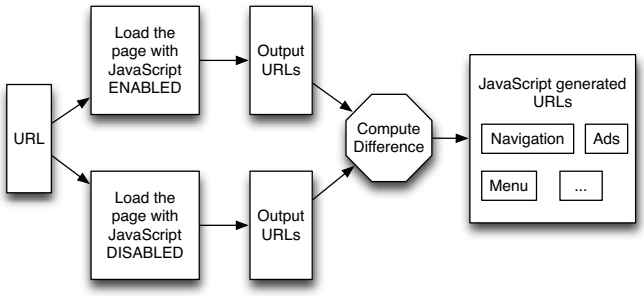

Figure 2: Finding URLs generated by JavaScript.

Our dynamic approach addresses the challenges listed in the previous section. We choose to run two instances of the tool simultaneously in locksteps, one with JavaScript enabled and the other disabled. Both instances synchronize first and then load the same web page, thus minimizing the impact of frequently changing page contents. Our tool takes the advantage that the browser engine can readily handle all JavaScript libraries. For the HTML anchor in the form of JavaScript code, the tool directly emulates user clicks on the

anchor. Then from the browser engine's outgoing channel, the tool detects the real URL if any.

A. Browser Emulation

We build a tool based on WebKit browser engine [3]. Given a set of URLs as input, our tool automatically loads each web page. Once the tool detects that the page has been loaded completely, the traversal of the DOM tree of the page is performed. Finally, the tool outputs all URLs found within the page. When running, the tool can either enable or disable the JavaScript engine.

We augment the WebKit engine in the following ways. First, we define a slot for `loadFinished()` signal to be notified when a page finishes loading. In the slot, the tool first traverses the whole DOM tree to find all anchors. For embedded frames in a web page, our tool recursively traverses each frame and outputs URLs in the frame. Then the hyper-references of these anchors are normalized into full URLs, where duplicates are removed. If JavaScript is enabled, the tool emulated user clicks on anchors, that have JavaScript code as hyper-references, by actually running the code. Any subsequent URL requests made by the engine are recorded in the output URL set. To reduce delays due to DNS lookups, the tool prefetches for the next ten domains in the input URL list.

There are a few limitations of our approach. First, some JavaScript code is triggered by user events, such as a mouse move-over event. Currently, out tool cannot cover such code and may miss some URLs. Second, many browser extensions dynamically change the page that a user is visiting. For instance, the AdBlock Plus [4] for Firefox allows a user to subscribe to a filter list so that any content origin matching the list will be removed. As a result, URLs we found may not appear in a user's browser. We could incorporate such browser extensions into our tool. However, there is no way to tell which extensions users are using. Thus, determining if the URLs found in JavaScript are displayed in a user's browser is beyond our work.

B. Classifying URLs

We use a semi-automatic method to classify URLs generated from JavaScript into eight different types. Briefly speaking, we first use regular expressions as rules for classification. For the remaining URLs, manual classification is performed. We use filters from the AdBlock Plus subscription list [5] as our rules for identifying advertisement. For the other types, we derive rules via manual inspections.

C. Categories of URLs in JavaScript

URLs are classified into eight categories. According to the content types of URL pages, the categories include advertisement, recommendations, realtime information, multimedia addons, and emails. The other three (i.e., navigation bars, sharing, and menus) are related to the functionality of URLs.

1) Advertisement: A large number of URLs in JavaScript are used to show advertisement, either from the site itself, or more often from an advertising agency. Figure 3 illustrates an example of using Google AdSense.

```
<script type="text/javascript"><!--
    google_ad_client = "pub-0855602874499281";
    /* 728x90, created 8/14/09 */
    google_ad_slot = "3395622907";
    google_ad_width = 728;
    google_ad_height = 90;
//-->
</script>
<script type="text/javascript"
src="http://pagead2.googlesyndication.com/pagead/show_ads.
    js">
</script>
```

Figure 3: A typical example of advertisement using Google AdSense.

2) Recommendations: An interesting feature of many sites is to recommend popular pages for users visiting the sites. Figure 4 illustrates such an example from Squidoo.com, where recommendations are delivered as an AJAX response of the code shown in Figure 1.

Figure 4: JavaScript generated recommendation URLs from http://squidoo.com.

3) Realtime Information: Stock prices, weather forecast, and game scores are highly volatile. As a result, sites often employ JavaScript to dynamically retrieve such data from publishing sites. Figure 5 illustrates such an example of stock market data.

Figure 5: JavaScript generated realtime URLs from http://nytimes.com.

[3] http://trac.webkit.org/wiki/QtWebKit
[4] https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/
[5] http://adblockplus.org/en/subscriptions

4) Multimedia Addons: Some sites that offer multimedia contents, especially videos, often need to install special browser plugins, e.g., flash players. These sites use JavaScript to detect if the required plugin is installed and to prompt users for installation. Figure 6 is an example JavaScript code for installing a movie player addon.

```
function clickplgbg58r(){
   window.parent.location = 'http://install.securesoft
.info/installer/zcdownload/948b13b3e2835c6b89039e9a2ea
964d02989b81ed38f1f236a2ef977770c97ac334e71823c:bb8b3c
bd3f75b56aac41ffd10ca2db75/?lp=http%3A%2F%2Fgalleries.
securesoft.info%2F874191c1bc%2F854099c2be10';
   closeplgbg58r();
   return void(0);
}
function initializeplgbg58r(){
   document.body.style.marginTop='30px';
   document.write('<div id="plgbg58r"><div id="plgbg58r2
        "><a href="javascript:clickplgbg58r()">Additional
        HD Player - install this to view and download
        movies in HD Quality</a></div><div id="plgbg58r3
        "><a href="javascript:clickplgbg58r()" title="
        Install HD Player Plugin for FREE"><span>Install
        This Player...</span></a></div><div id="plgbg58r4
        ">  <a href="javascript:closeplgbg58r()
        ">X</a></div></div>');
}
initializeplgbg58r();
```

Figure 6: An example code of installing a movie player addon from http://10starmovies.com/Watch-Movies-Online/Apocalypse_Island_2010/.

5) Emails: There are some email addresses generated by JavaScript. One reason for doing this is apparently hiding email addresses from spammers, because extracting emails from JavaScript is significantly more difficult than parsing HTML documents. Figure 7 illustrates such an example for hiding the email address.

```
var prefix = '&#109;a' + 'i&#108;' + '&#116;o';
var path = 'hr' + 'ef' + '=';
var addy80941 = 'M&#117;r&#97;t.Ah&#97;t' + '&#64;';
addy80941 = addy80941 + '&#101;t&#117;' + '&#46;' +
        '&#101;ph&#101;' + '&#46;' +
        's&#111;rb&#111;nn&#101;' + '&#46;' + 'fr';
document.write( '<a ' + path + '\'' + prefix + ':' +
        addy80941 + '\'>' );
document.write( addy80941 );
document.write( '<\/a>' );
```

Figure 7: The JavaScript code of hiding email addresses. The produced HTML code is: `<a href="mailto:Murat.Ahat@etu.ephe.sorbonne.fr">Murat.Ahat@etu.ephe.sorbonne.fr</a>`

6) Navigation Bars: Many pages contain navigational bars to facilitate users browsing through the site. For instance, Figure 8 is a sample navigation bar generated by JavaScript code.

7) Sharing: Many sites use JavaScript to generate links for sharing page content on social networking sites, such as Twitter, Facebook, and Google Plus, as shown in Figure 9.

Figure 8: JavaScript generated navigational URLs.

Figure 9: JavaScript generated sharing URLs.

8) Menus: Different from navigation bars, context menu is another type of user interface for display directories, e.g., as shown in Figure 10.

D. Why putting URLs in JavaScript?

We summarize the reasons for putting URLs in JavaScript as follows:

- **Displaying volatile contents.** Some information is updated constantly in a page, such as advertisement, recommendations, and realtime information. As a result, sites often adopt JavaScript to dynamically retrieve relevant information.
- **Providing better user interface.** Navigation bars, menus, and sharing links provide better user experiences for web pages. For example, menus allow users to easily view contents organized in directories; and sharing links facilitates users to quickly post contents to other sites.
- **Hiding information.** To prevent easy access to certain content, web developers have encoded URLs in JavaScript code instead of revealing them in HTML. Such examples include hiding email addresses from spammers, and concealing real URLs by software download sites, e.g., http://www.skycn.com.

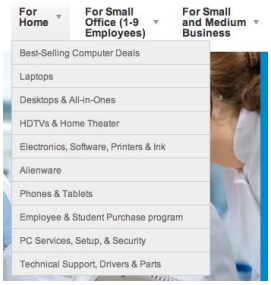

Figure 10: JavaScript generated menu URLs from http://dell.com.

IV. Evaluation

A. Datasets

We use two datasets to assess the usage patterns of JavaScript-generated URLs. Inspired by popularity based sampling [3], we first find most popular Google queries from Google trends [6]. By feeding these queries to Google, we then obtained a set of popular pages on the web. Among them, we randomly selected 888 pages in this study.

Another dataset is top sites from Alexa [7], which tracks traffic for visiting different web sites. We use the top 1,000 sites in this study. Among them, there are sites blocked by the Great Chinese Firewall [6] so that we cannot visit. Some other sites display different pages for each visit. Still others redirect to different pages for browsers with JavaScript enabled or disabled. Excluding these sites, we obtained URLs from 814 sites and our analysis was performed on this dataset.

B. Overall Results

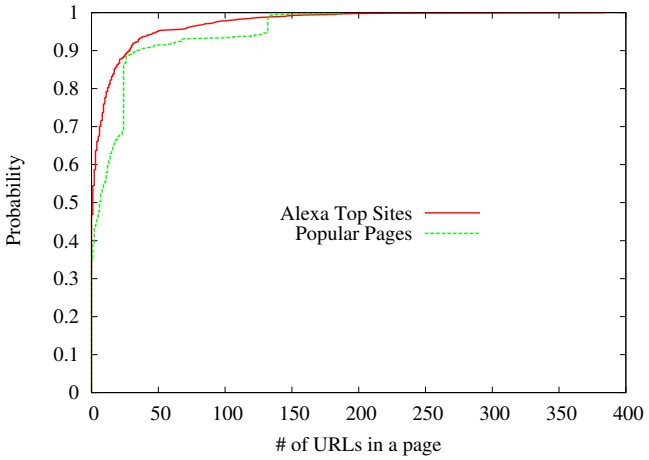

Figure 11: Cumulative Distribution Function (CDF) of JavaScript-generated URLs in web pages.

For the Alexa dataset, there are a total of 147,114 and 137,327 URLs for browser with JavaScript enabled and disabled, respectively. The number of JavaScript-generated new URLs (i.e., those that do not appear in static HTML code) is 8,796, which accounts for 5.98% of total URLs in web pages.

For popular pages, there are a total of 90,208 and 75,005 URLs for browser with JavaScript enabled and disabled, respectively. The number of JavaScript-generated new URLs is 16,765, or 18.58% of the total URLs.

Figure 11 illustrates the Cumulative Distribution Function (CDF) of URLs generated from JavaScript code from web pages. 46.8% of Alexa sites and 34.7% of popular pages do not have JavaScript-generated URLs. 90% of pages have less than 28 and 35 JavaScript generated URLs for Alexa sites and popular pages, respectively. The distribution exhibits the heavy tail property. The maximum number of URLs from JavaScript code in a page is 385.

[6]http://www.google.com/trends
[7]http://www.alexa.com/

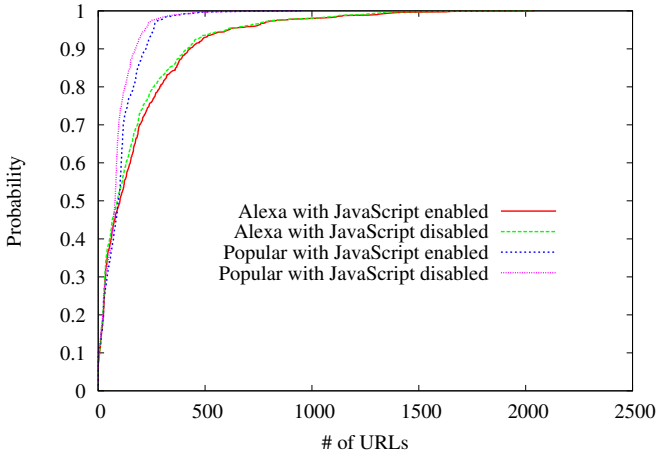

Figure 12: Cumulative Distribution Function (CDF) of total URLs in web pages.

Figure 12 illustrates the distribution of total URLs in web pages. We can observe that these two curves are very close to each other, indicating that the majority of URLs is already in the HTML pages. In fact, URLs generated from JavaScript accounts for 5.98-18.58% of total URLs. For Alexa top sites, the 90-percentile of pages have less than 421 URLs with JavaScript disabled, and less than 431 URLs with JavaScript enabled. The maximum number of URLs exceeds 2000 for both cases. For popular pages, the 90-percentile of pages have less than 177 URLs with JavaScript disabled, and less than 224 URLs with JavaScript enabled. The maximum number of URLs is 949 for both.

C. JavaScript URL Categories

We classify all URLs generated from JavaScript into a number of categories. The two datasets exhibit different characteristics. As shown in Table I, for Alexa top sites advertisement accounts for 34.9%, followed by 23.45% navigation bars and 18.07% recommendations, and all other types have less than 10% share. In contrast, for popular pages sharing links accounts for 38.35%, followed by 26.04% of advertisement and 16.87% of recommendations, and the rest has less than 10% share.

The higher percentage of sharing links in popular pages indicates people are more likely to share these pages, which may in turn increase their rankings in search engine index, thus becoming even more popular.

For top sites, the intention is more focused on helping visitors browse through other site contents, so navigation

Table I: Categorization of JavaScript-generated URLs.

Types	Ads	Menu	Email	Recommendation	Realtime info	Nav. bar	Addons	Sharing	Others
# of URLs (Alexa)	3070	159	6	1589	355	2063	19	821	712
Percentage	34.90	1.81	0.07	18.07	4.04	23.45	0.22	9.33	8.09
# of URLs (Popular)	4366	195	4	2828	795	1557	43	6429	548
Percentage	26.04	1.16	0.02	16.87	4.74	9.29	0.26	38.35	3.27

bars and recommendations occupy a strong share (41.5% total). Even in popular pages, these two types have 26.2% share. Because navigation bars and recommendations refer to other potentially important pages, they are definitely interesting to search engine crawlers.

For both datasets, advertisement takes 26.0-34.9%, indicating strong motivations for profit from contents.

V. Related Work

Previous work [4] has studied redirection spam using JavaScript. Redirections were found by loading URLs into a browser with JavaScript initially disabled and then enabled. If the destination URLs are different, then a redirection is found. Our work focuses on finding JavaScript-generated URLs, which is a different goal.

A study on passive URL discovery [1] collects URLs from toolbar logs of users' browsers and finds that 80% URLs in toolbar logs were not discovered by the crawler. Heymann et al. [8] propose using URLs posted on a social bookmarking site, i.e., del.icio.us, to improve crawler's coverage, as approximately 25% URLs posted are new and unknown to search engines. Previous work has identified the hidden web [12], where web resources can only be discovered by properly filling web forms [11, 10]. Our work complements these existing studies on finding more URLs for crawlers, but differs in the sense that we actively extracting URLs from JavaScript codes.

There has been a number of tools for statically analyzing JavaScript. Kudzu [15] is a symbolic execution framework for exploring client-side code injection vulnerabilities. TAJS [9] is a static analysis tool that can detect type-related and dataflow-related programming errors. Our approach takes a different method by dynamically executing JavaScript code.

VI. Conclusions and Future Work

We studied the usage of JavaScript-generated URLs in web pages. Our approach is to compare the difference of a browser engine with JavaScript enabled and disabled. We found that more than half pages have some forms of JavaScript-generated URLs (about 5.98-18.58% of total URLs). Among them, advertisement accounts for about one third to one fourth. Navigation bars and recommendations share 26.2-41.5%, which are links to meaningful site contents and should be indexed by search engine crawlers. For popular pages, we find the majority is sharing links.

In this work, we focus on the URLs found for WebKit engine as a first attempt. Our ongoing work is to employ different browser engines and to compare the difference among them.

Acknowledgment

We thank anonymous referees for their helpful comments on this paper. This work was supported in part by NSFC 61003012.

References

[1] X. Bai, B. B. Cambazoglu, and F. P. Junqueira. Discovering urls through user feedback. In *CIKM '11*, Glasgow, Scotland, UK, 2011.

[2] G. Bulmash. Does google index dynamic javascripted content? http://www.brainhandles.com/2007/03/11/does-google-index-dynamic-javascripted-content/, 2007.

[3] K. Chellapilla and M. Chickering. Improving cloaking detection using search query popularity and monetizability. In *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2006.

[4] K. Chellapilla and A. Maykov. A taxonomy of javascript redirection spam. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pages 81–88, 2007.

[5] ECMA International. ECMAScript language specification. stardard ECMA-262, 3rd edition. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, 1999.

[6] Global Internet Freedom Consortium. The great firewall revealed. http://www.internetfreedom.org/files/WhitePaper/ChinaGreatFirewallRevealed.pdf, 2002.

[7] Google. Cloaking, sneaky javascript redirects, and doorway pages. http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=66355, 2011.

[8] P. Heymann, G. Koutrika, and H. Garcia-Molina. Can social bookmarking improve web search? In *Proc. 1st Int'l Conf. on Web Search and Data Mining*, pages 195–206, 2008.

[9] S. H. Jensen, M. Madsen, and A. Møller. Modeling the HTML DOM and browser API in static analysis of JavaScript web applications. In *Proc. 8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, September 2011.

[10] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *Proc. VLDB Endowment*, 1(2):1241–1252, 2008.

[11] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *Proc. 5th ACM/IEEE-CS Joint Conf. on Digital Libraries*, pages 100–109, 2005.

[12] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proc. 27th Int'l Conf. on Very Large Data Bases*, pages 129–138, 2001.

[13] G. Richards, C. Hammer, B. Burg, and J. Vitek. The eval that men do - a large-scale study of the use of eval in javascript applications. In *ECOOP*, pages 52–78, 2011.

[14] G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of javascript programs. In *PLDI*, pages 1–12, June 2010.

[15] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A symbolic execution framework for javascript. In *Proc. of the 31st IEEE Symposium on Security and Privacy*, pages 513–528, Oakland, CA, USA, 2010.