Hidden-Web Induced by Client-Side Scripting: An Empirical Study

Zahra Behfarshad and Ali Mesbah

University of British Columbia,
Vancouver, BC, Canada
{janab,amesbah}@ece.ubc.ca

Abstract. Client-side JavaScript is increasingly used for enhancing web application functionality, interactivity, and responsiveness. Through the execution of JavaScript code in browsers, the DOM tree representing a webpage at runtime, can be incrementally updated without requiring a URL change. This dynamically updated content is hidden from general search engines. In this paper, we present the first empirical study on measuring and characterizing the hidden-web induced as a result of client-side JavaScript execution. Our study reveals that this type of hidden-web content is prevalent in online web applications today: from the 500 websites we analyzed, 95% contain client-side hidden-web content; On those websites that contain client-side hidden-web content, (1) on average, 62% of the web states are hidden, (2) per hidden state, there is an average of 19 kilobytes of data that is hidden from which 0.6 kilobytes contain textual content, (3) the `DIV` element is the most common clickable element used (61%) to initiate this type of hidden-web state transition, and (4) on average 25 minutes is required to dynamically crawl 50 DOM states. Further, our study indicates that there is a correlation between DOM tree size and hidden-web content, but no correlation exists between the amount of JavaScript code and client-side hidden-web.

1 Introduction

General web search engines cover only a portion of the web, called the visible or indexable web, which consists of the set of web pages reachable purely by following URL-based links. There is, however, a large body of valuable web content that is not accessible by simply following hypertext links. Well-known examples include dynamic server-side content behind web forms [3,19] reachable through application-specific queries. This portion of the web, not reachable through search engines, is generally referred to as the invisible or hidden web, which, in 2001, was estimated to be 500 times larger than the visible web [4]. More recently, form-based hidden web content has been estimated at several millions of pages [3,12].

With the wide adoption of client-side programming languages such as JavaScript and Ajax techniques to create responsive web applications, there is a new type of hidden-web that is growing rapidly. Although there has been

F. Daniel, P. Dolog, and Q. Li (Eds.): ICWE 2013, LNCS 7977, pp. 52–67, 2013.

extensive research on detecting [3,14,19] and measuring [4,11] hidden-web content behind web forms, hidden-web induced as a result of client-side scripting has gained limited attention so far.

JavaScript is the dominant language for implementing dynamic web applications. Today, as many as 97 of the top 100 most visited websites [1], have client-side JavaScript [20], often consisting of thousands of lines of code per application. JavaScript is increasingly used for offloading core functionality to the client-side and achieving rich web interfaces. JavaScript code interacts with and incrementally updates the Document Object Model (DOM) in an event-based style. Changes made dynamically to the structure, contents or styles of the DOM elements are directly manifested in the browser's display. This event-based style of interaction is substantially different from the traditional URL-based page transitions through hyperlinks, where the entire DOM is repopulated with a new HTML page from the server for every user-initiated state change.

The goal of our paper is to measure the pervasiveness and characterize the nature of *hidden-web content induced by client-side JavaScript* in today's web applications. For simplicity, we will refer to this type of hidden-web content as **client-side hidden-web** throughout this paper. To the best of our knowledge, we are the first to conduct an empirical study on this topic. Our empirical data shows that as high as 95% of the 500 websites we analyzed contain hidden-web content, and on average 62% of the 50 states we crawled for each website are hidden due to client-side scripting.

2 Background and Motivation

Client-Side Hidden-Web Content. Client-side scripting empowers achieving dynamic and responsive web interfaces in today's web applications. Through JavaScript, the client-side runtime DOM tree of asear web application can be dynamically updated with new structure and content. These updates are commonly initiated through event-listeners, AJAX callbacks, and timeouts. The new content, either originated from the server-side or created on the client-side, is then injected into the DOM through JavaScript to represent the new state of the application.

Although DOM manipulation through JavaScript increases responsiveness of web applications, these dynamically mutated states end up in the hidden-web portion of the web. The main reason is that crawling such dynamic content is fundamentally more challenging and costly than crawling classical multi-page web applications, where states are explicit and correspond to pages that have a unique URL assigned to them.

Client-side state is determined dynamically through changes in the DOM that are only visible after executing the corresponding JavaScript code. The major search giants have currently little or no support for dynamic analysis of JavaScript code due to scalability and security issues. They merely extract hypertext links and index the resulting HTML code recursively.

```
1 $(document).ready(function() {
2   $('div.update').click(function() {
3     var updateID = $(this).attr('rel');
4     $.get('/news/', { ref:updateID },
5      function(data) {
6        $(updateID+'Container').append(data); });  }) });
```

Fig. 1. JavaScript code for updating the DOM after a click event

```
<body><h1>Sports News</h1>
 <p><span id="sportsContainer"></span></p>
 <div class="update" rel="sports">Update!</div>
</body>
```

Fig. 2. The initial DOM state

Hidden-Content Example. We present a simple example of how JavaScript code can induce hidden-web content by dynamically changing the DOM tree. Figure 1 depicts a JavaScript code snippet using the popular jQuery library.[1] Figure 2 illustrates the initial state of the DOM before any modification has occurred. Once the page is loaded (line 1 in Figure 1), the JavaScript code attaches an `onclick` event-listener to the `DIV` DOM element with class attribute `'update'` (line 2). When a user clicks on this `DIV` element, the anonymous function associated with the event-listener is executed (lines 2–8). The function then sends an asynchronous call to the server (line 4), passing a parameter read from the `DIV` element (i.e., `'sports'`) (line 3). On the callback, the response content from the server is injected into the DOM element with ID `'sportsContainer'` (line 6). The resulting updated DOM state is shown in Figure 3. All the data retrieved and injected into the DOM this way will be hidden content as it is not indexed by search engines. Although the effect of client-side scripting on the hidden-web is clear, there is currently a lack of comprehensive investigation and empirical data in this area.

3 Related Work

Crawling the Hidden-Web. Crawling techniques have been studied since the advent of the Web itself. Web crawlers find and index millions of HTML pages daily by searching for hyperlinks. Yet a large amount of data is hidden behind web queries and therefore, extensive research has been conducted towards finding and analyzing the hidden-web – also called deep-web – behind web forms [3,7,8,14,15,19]. The main focus in this line of research is on exploring ways of detecting query interfaces and accessing the content in online databases, which is usually behind HTML forms. This line of research is merely concerned with server-side hidden-web content (i.e., in databases).

On the contrary, exploring the hidden-web induced as a result of client-side scripting has gained very little attention so far. Alvarez et al. [2] discussed the

[1] `http://jquery.com`

```
<body>
 <h1>Sports News</h1>
 <p><span id="sportsContainer">
   <h3>US GP: Vettel fastest in Austin second practice</h3>
   <p>Vettel produced an ominous performance</p></span></p>
 <div class="update" rel="sports">Update!</div>
</body>
```

Fig. 3. The updated DOM tree after clicking on 'Update!'

importance and challenges of crawling client-side hidden-web. Mesbah et al. [17] proposed an automated crawler, called CRAWLJAX, for AJAX-based web applications. Duda et al. [9] presented how DOM states can be indexed. The authors proposed a crawling and indexing algorithm for client-side state changes.

Measuring the Hidden-Web. Researchers have reported their results of measuring the hidden-web behind forms. In 2001, Bergman [4] reported a study indicating that the hidden-web was about 500 times larger than the visible web. In 2004, Chang et al. [5] measured hidden-web content in online databases using a random IP-sampling approach, and found that the majority of the data in such databases is structured. In 2007, He et al. [11] conducted a study using an overlap analysis technique between some of the most common search engines such as Yahoo!, Google, and MSN and discovered that 43,000-96,000 deep websites existed. They presented an informal estimate of 7,500 terabytes of hidden data, which was 500 times larger than the visible web, which supported the earlier results by Bergman.

All this related work focuses on measuring server-side hidden-web behind forms. To the best of our knowledge, we are the first to study and measure client-side hidden-web.

4 Methodology

Our main objective is to gain an understanding of how much dynamic client-side content is unsearchable for end-users on the Web. To that end, we conduct a quantitative empirical study to measure the pervasiveness and characterize the nature of hidden-web content induced by client-side scripting. Our research questions are formulated as follows:

RQ1 How pervasive is client-side hidden-web in today's web applications?
RQ2 How much content is typically hidden due to client-side scripting?
RQ3 Which clickable elements contribute most to client-side hidden-web content?
RQ4 Are there any correlations between the degree of client-side hidden-web and a web application's characteristics?

4.1 Experimental Objects

In this study, we analyze 500 unique websites in total. To obtain a representative pool of websites, similar to other researchers [13,20], we select 400 unique

websites from Alexa's Top Sites [1] (henceforth referred to as ALEXA). For multiple instances of the same domain on Alexa's top list (e.g., *www.google.com*, *www.google.fr*), we only include and count one instance in our 400 objects list. In addition, we gather another 100 random websites using Yahoo! random link generator (henceforth referred to as RANDOM), which is also used in other studies [6,16]. All the 500 websites (henceforth referred to as TOTAL) were crawled and analyzed throughout February-March 2013.

4.2 Experimental Design

To investigate the pervasiveness of hidden content due to client-side scripting (**RQ1**), we examine all the 500 websites and count the percentage of websites that exhibit client-side hidden-web content. In addition, for each of the websites that contains hidden-web content, we measure what portion of the crawled (50) states is client-side hidden-web. To measure the amount of content that is hidden (**RQ2**), we compute the total and average in terms of textual *differences* between each hidden state and its previous state. To address **RQ3**, we classify the type of clickable elements, which clicking them results in a hidden state in our analysis. We assess what type of DOM elements are commonly used in practice by web developers that induce this type of dynamic JavaScript-driven state change. In order to answer **RQ4**, we analyze possible correlations between the client-side hidden-web content and the average DOM size and custom JavaScript code of each website examined, for 100 websites randomly chose from our pool of 500 websites. In the next section, technical details of our analysis approach are presented.

5 Client-Side Hidden-Web Analysis

We have implemented our client-side hidden-web analysis approach in a tool called JAVIS, which is available for download, along with all our empirical data.[2]

Figure 4 depicts our client-side hidden-web content analysis technique which is composed of three main steps: (1) dynamically crawling each given website, (2) classifying the detected state changes into visible and hidden categories, and (3) conducting characterization analyses of the hidden states. Each step is described in the subsequent subsections.

5.1 Event-Driven Dynamic Crawling

State Exploration. Our approach for automatically exploring a web application's state space is based on our CRAWLJAX [17] work. CRAWLJAX is a crawler capable of automatically exploring JavaScript-induced DOM state changes through an event-driven dynamic crawling technique. It exercises client-side code, detects and executes clickables that lead to various dynamic states of Web

[2] `http://salt.ece.ubc.ca/content/javis/`

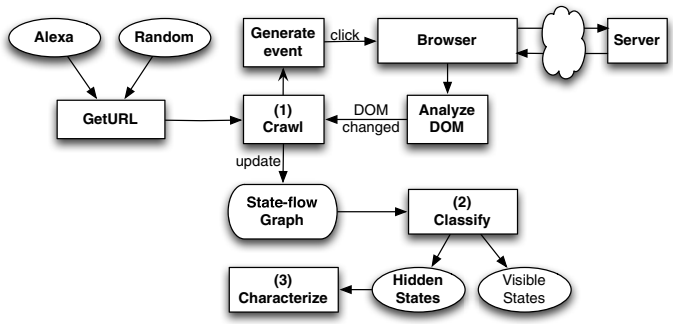


Fig. 4. Overview of our client-side hidden-web analysis

2.0 AJAX-based web applications. By firing events on the web elements and analyzing the effects on the dynamic DOM tree in a real browser before and after the event, the crawler incrementally builds a *state-flow graph* (SFG) capturing the client-side states and possible event-based transitions between them. This state-flow graph is defined as follows:

Definition 1. *A **state-flow graph** SFG for an* AJAX*-based website* **A** *is a labeled, directed graph, denoted by a 4 tuple* $< \boldsymbol{r}, \boldsymbol{V}, \boldsymbol{E}, \mathcal{L} >$ *where:*

1. $\boldsymbol{r}$ *is the root node (called Index) representing the initial state when* **A** *has been fully loaded into the browser.*
2. $\boldsymbol{V}$ *is a set of vertices representing the states. Each* $\boldsymbol{v} \in \boldsymbol{V}$ *represents a runtime DOM state in* **A**.
3. $\boldsymbol{E}$ *is a set of (directed) edges between vertices. Each* $(\boldsymbol{v}_1, \boldsymbol{v}_2) \in \boldsymbol{E}$ *represents a clickable* $\boldsymbol{c}$ *connecting two states if and only if state* $\boldsymbol{v}_2$ *is reached by executing* $\boldsymbol{c}$ *in state* $\boldsymbol{v}_1$.
4. $\mathcal{L}$ *is a labelling function that assigns a label, from a set of event types and DOM element properties, to each edge.*
5. *SFG can have multi-edges and be cyclic.*

CRAWLJAX is also capable of crawling traditional URL-based websites. It is fully configurable in terms of the type of elements that should be examined or ignored during the crawling process. For more details about the architecture, algorithms or capabilities of CRAWLJAX the interested reader is referred to [17,18].[3]

Crawling Configuration. We have extended, modified, and configured CRAWLJAX for this study as follows:

Maximum states. Dynamic crawling is quite expensive and time consuming. To constrain the state space and still acquire a representative sample for our analysis in a timely manner, we define an upper limit on the number of states to dynamically crawl for each website, namely, 50 unique DOM states.

[3] `http://crawljax.com`

Crawling depth. Similar to other studies [11], we set the maximum crawling depth to 3 levels.

Candidate clickables. Traditionally, forms and anchor tags pointing to valid URLs were the only clickables capable of changing the state (i.e., by retrieving a new HTML page from the server after the click). However, in modern websites, web developers can potentially make any HTML element to act as a *clickable* by attaching an event-listener (e.g., `onclick`) to that element. Such clickables are capable of initiating DOM mutations through JavaScript code. In our analysis, we include the most commonly used clickable elements, namely: `A`, `DIV`, `SPAN`, `IMG`, `INPUT` and `BUTTON`.

Event type. We specify the event type to be `click`. This means the crawler will generate click events on DOM elements that are spotted as candidate clickables, i.e., elements potentially capable of changing the DOM state. Note that there are other types of events (e.g., `onmouseover`) that can generate hidden-web content. Our study is currently targeted towards the click event, which is the mostly commonly used event-type in web applications.

Randomized crawling. In order to get a simple random sample, we randomize the crawling behaviour in terms of selecting the next candidate clickable for exploration. Hence, the crawler clicks on any of the defined candidate clickable types (e.g., `DIV`, `A`, etc) randomly while crawling.

Once the tool is configured, we automatically select and crawl each website, and save the resulting state-flow graph containing the detected states (DOM trees) and transitional edges (clickables).

5.2 Classification

As shown in Figure 4, for each website crawled, we classify the detected states into two categories: visible and hidden. Our client-side hidden-web analysis is largely based on the following two assumptions:

1. A valid URL-based state transition can be crawled and indexed by general search engines and, therefore, it is visible;
2. A non-URL-based state transition is not crawled nor indexed by general search engines and thus, it ends up in the hidden-web; For instance, the DOM update presented in Figure 3, as a result of clicking on the `DIV` element of Figure 2, is hidden.

To classify the crawled states into the visible or hidden group, we traverse the inferred state-flow graph of each website. For each state, we analyze all the incoming edges (i.e., clickables). If the incoming edges is a valid URL-based transition, we consider that state to be visible, otherwise it is hidden.

Each edge contains information about the type of clickable element that caused a state change. Our classification uses that information to decide which resulting states are hidden as follows:

Anchor tag (A). The anchor tag can produce both visible and hidden states, depending on the presence and URL validity of the value of its `HREF` attribute. For instance, clicking on `<A HREF='www.eg.com/news/'>` results in a visible state, whereas `<A HREF='#' onclick='updateNews();'>` can produce a hidden state.

IMG. The image tag is also interesting since it can result in a visible state when embodied in an anchor tag with a valid URL; For every edge of `IMG` type, we retrieve the parent element from the corresponding DOM state. If the parent element is an anchor tag with a valid URL, then we categorize the resulting state as visible, otherwise the state is hidden.

Other element types. Per definition, `DIV`, `SPAN`, `INPUT`, and `BUTTON` do not have attributes that can point to URLs, and thus, the resulting state changes are all categorized as hidden.

5.3 Characterization Analysis

Hidden-Web Quantity. Once the explored states are categorized, we annotate the hidden states on the state-flow graph to measure the amount of hidden-web data in those states. We traverse the annotated state-flow graph, starting from Index, and for each annotated hidden state, we compute the differences between the previous state (which could be a visible or hidden state) and the annotated hidden state using a differencing engine. To measure the amount of data that can be hidden, the differencing method computes merely the *additions* in the target (hidden) state. For each website, Javis saves all the differences in a file and measures the total size in bytes. We also compute the pure textual content in the total differences.

Clickable Types. To investigate which clickable type (i.e., `A`, `DIV`, `SPAN`, `IMG`, `INPUT` and `BUTTON`) contributes most to inducing hidden-web content in practice, Javis examines the annotated state-flow graph and gathers the edges that result in hidden states. It then calculates, for each element type, the mean of its contribution portion to the hidden-web percentage.

Correlations. Further, we measure the average DOM string size as well as the custom JavaScript code (excluding common libraries such as jQuery, Dojo, Ext, etc) of each website. To examine the relationship between these measurements and the client-side hidden-web content, we use R [10] to calculate the non-parametric Spearman correlation coefficients (r) as well as the p-values (p), and plot the graphs.

6 Results

Table 1 provides a representative small sample (20 websites) of the kind of websites we have crawled and the type of data we have gathered, measured, and analyzed in this study. These websites are randomly selected from our total pool of 500 websites. The first 10 are taken from Alexa and the second 10 from

Table 1. Hidden-web Analysis Results. The first 10 are from ALEXA, and the remaining 10 from RANDOM.

			Clickables			Clickable Types								Size		Hidden				
ID	Site Name	States (#)	Total	Visible	Hidden	A (Visible)	A (Hidden)	Div	Span	Img (Visible)	Img (Hidden)	Input	Button	JavaScript (KB)	DOM (KB)	States (%)	Total (KB)	Total Content (KB)	Average (KB)	Time Elapsed (S)
1	Google	50	49	3	46	3	0	29	16	0	1	0	0	329	210	94	906	13	18	228
2	ESPN	50	49	12	37	6	0	26	2	6	9	0	0	161	196	75	4358	120	89	7565
3	AOL	50	49	8	41	5	1	18	22	3	0	0	0	203	170	82	4626	140	64	4727
4	Youtube	50	49	7	42	7	0	7	17	0	7	0	17	286	153	84	4230	153	86	530
5	Aweber	50	49	24	25	16	1	20	0	8	4	0	0	41	31	65	38	0	0.78	740
6	Samsung	50	49	3	46	2	0	42	3	1	0	0	1	96	267	92	1381	21	28	1274
7	USPS	50	49	8	41	5	1	33	7	3	0	0	0	200	258	82	563	6	11.5	317
8	BBC	50	49	41	8	25	0	3	3	16	2	0	0	142	112	16	293	6	6	794
9	Alipay	50	49	2	47	2	7	33	7	0	0	0	0	200	72	94	77	0	1.5	828
10	Renren	50	49	0	49	0	0	49	0	0	0	0	0	100	47	100	1613	3	33	152
11	EdwardRobertson	50	49	1	48	1	2	45	1	0	0	0	0	120	64	98	154	7	3.14	161
12	Rayzist	50	49	31	18	31	1	16	0	0	1	0	0	329	54	37	257	38	5.2	976
13	Metmuseum	50	49	3	46	3	0	2	0	0	44	0	0	54	87	94	935	68	19	364
14	JiveDesign	50	49	0	49	0	0	49	0	0	0	0	0	241	202	100	369	0	7.5	322
15	MTV	50	49	0	49	0	0	19	0	0	30	0	0	242	200	100	530	14	10.8	417
16	Challengeair	50	49	0	49	0	0	49	0	0	0	0	0	176	28	100	22	0	0.45	145
17	Mouchel	50	52	52	0	51	0	0	0	1	0	0	0	20	60	0	0	0	0	535
18	Sacklunch	50	49	45	4	3	0	3	0	42	1	0	0	121	83	8	166	6	3.39	236
19	Pongo	50	49	3	46	3	0	46	0	0	0	0	0	61	463	94	4229	58	83.3	713
20	MuppetCentral	50	49	17	32	8	0	32	0	9	0	0	0	254	224	65	4807	272	98.1	966

RANDOM. The complete set of our empirical data is available for download.[4] It should be noted that Total column in the table refers to both the hidden DOM structure and the textual content while the Total Content only refers to the hidden textual content. The Average is the average hidden content and DOM structure per state.

6.1 Pervasiveness (RQ1)

95% (476/500) of the websites we analyzed exhibit some degree of client-side hidden-web content, i.e., they have at least one or more client-side hidden states.

To gain deeper knowledge of what percentage of the 50 states crawled from each of these websites actually contain hidden-web content, each web application is analyzed individually. Figure 5 presents three box plots illustrating the hidden-web state percentages for ALEXA, RANDOM, and TOTAL.

Alexa. For the 400 websites from ALEXA, on average 65.63% of the 50 states we analyzed were client-side hidden-web. This high number can be explained by the nature of such websites perhaps. They are among the top most visited sites in the world. As such, developers of many of these websites use the latest Web

[4] `http://salt.ece.ubc.ca/content/javis/`

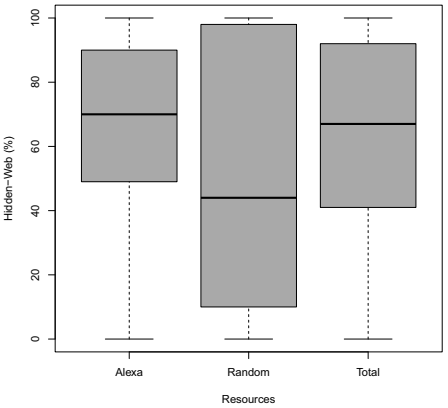


Fig. 5. Box plots of the percentage of client-side hidden-web states in ALEXA, RANDOM, and TOTAL. 50 states (pages) are crawled from each website.

Table 2. Descriptive statistics of the average hidden-web content for all states and per state

	Textual hidden content (KB)			All hidden content (KB)		
Hidden-Web	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
Per State	0	0.60	11.65	0	18.91	286.4
All States	0	27.6	536	0	869.7	13170

2.0 technologies, such as JavaScript, DOM, Ajax, and HTML5, to provide high quality features that come with rich interaction and responsiveness. As we have discussed in Section 2, these Web 2.0 techniques contribute enormously to the creation of client-side hidden-web.

Random. An average of 50.6% of the states from the RANDOM websites constitute hidden states. These websites were purely randomly chosen on the web. In other words, we do not know about their rankings nor their popularity among end users. The lower percentage is perhaps due to the fact that many websites on the web might still are quite classical in nature, meaning they use more URL-based links for state transitions, rather than using JavaScript. However, although the percentage is not as high as the websites on ALEXA, the rather high 50.6% in the wild still points to the pervasiveness of client-side hidden-web on the web.

Total. When the results of ALEXA and RANDOM are combined in TOTAL, the total hidden-web state percentage is 62.52%. It should be noted that both ALEXA and RANDOM contain websites that have as low as 0% and as high as 100% hidden-web states, regardless of any rankings.

On average, per website 25 minutes was required to dynamically crawl 50 states. It took JAVIS 211 hours ($\approx$ 8.8 days) to crawl and classify all the 500 websites (each with 50 states).

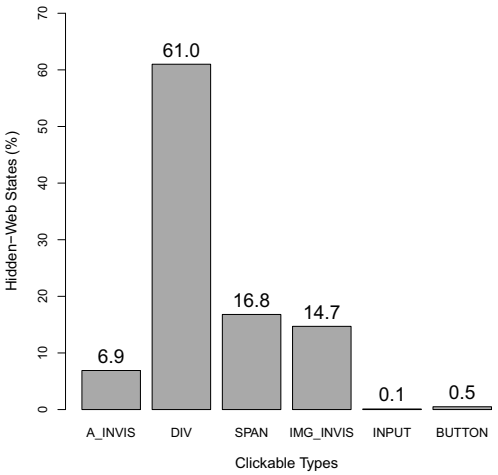


Fig. 6. Barplot of hidden-web percentage behind different types of clickables. 'A_INVIS' represents anchor tags without a (valid) URL. 'MG_INVIS' represents `IMG` elements not embedded in an anchor tag with a (valid) URL.

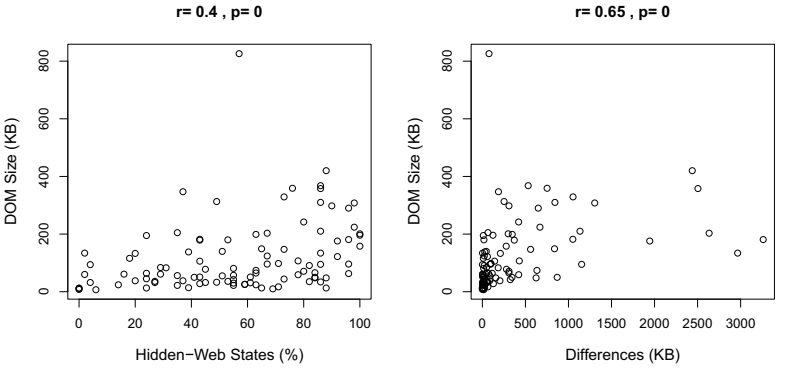
6.2 Quantity (RQ2)

In order to gain an understanding of the quantity of content in the client-side hidden-web states, we measured the amount of hidden data as described in Section 5.3. Table 2 shows the amount of client-side hidden-web content for all of the crawled hidden-web states, and per hidden-web state. It shows descriptive statistics for all the hidden content including DOM structures and textual content as well as only textual, natural language content, extracted from the DOM elements.

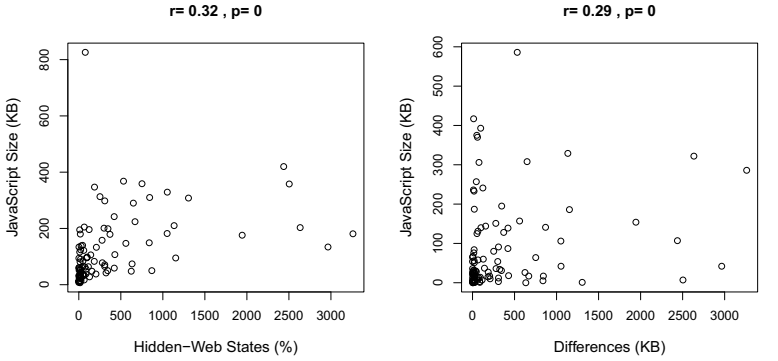
Per Hidden-Web State. Per hidden-web state, on average 19 kilobytes of DOM and textual content exist while 0.6 kilobytes was only textual content. Some states have as high as 286 kilobytes of hidden content.

All Hidden-Web States. For all the states crawled together, we measured an average of 870 kilobytes of client-side hidden-web content including both DOM and textual content while the textual content was around 27.6 KB. The minimum and maximum are 0, 13170 kilobytes, and 0 and 536 kilobytes respectively.

We manually examined some of the hidden textual content to understand why type of information would be hidden to end-users. The nature of the hidden textual content is a combination of singular words, numbers, short messages or whole sentences. The short messages are mostly informative descriptions of the websites or advertisements. The larger sentences range from descriptions of a particular subject, questions/answers, news items, and discussions in various domains such as health, science, animals, videos and images, actors and stars, sports and so on.



(a) DOM Size.



(b) JavaScript Size.

Fig. 7. Scatter plots of the average DOM and JavaScript size versus the hidden-web state percentage and content. r represents the Spearman correlation coefficient and p is the p-value.

6.3 Induction (RQ3)

To better understand what type of clickable elements web developers use in today's websites that induce state changes in the browser, we analyzed how much each clickable type contributes to the measured hidden-web state percentage.

As discussed in Section 5.3, the anchor tag (`A`) and the image element (`IMG`) can induce both visible and hidden states; For this part, we only consider the ones that cause hidden-states in our analysis. Figure 6 depicts a barplot of the different clickable types versus the associated hidden-web state percentage. We can see that the `DIV` has the highest contribution to the hidden-web state percentage (61%), followed by `SPAN` (16.8%). Interestingly, the `IMG` and `A` element types are also used quite often to induce client-side hidden content, with 14.7% and 6.9% each, respectively. Finally, `BUTTON` and `INPUT` contribute to less than one percent of the hidden-web states. This shows that while crawling, it is not sufficient to simply focus on the anchor tags of a website, any longer.

6.4 Correlations (RQ4)

DOM and JavaScript Size. We conducted a correlation analysis of the degree of hidden-web with respect to (1) average DOM size, taken over all the crawled states, and (2) JavaScript custom code size. Figure 7 depicts the scatter plots of these two measurements against the hidden-web state percentage and content.

For the DOM size, Figure 7(a)-Left indicates a weak correlation ($r = 0.4$) with the hidden-web state percentage while 7(a)-Right shows a strong correlation ($r = 0.65$) with the amount of hidden-web content. This comes as no surprise, because the larger the DOM tree, the more visible and hidden content there will be in a website.

For the JavaScript size, both Figures of 7(b) indicate a weak monotonic correlation with the percentage and amount of client-side hidden-web. We expected to see a stronger correlation, because after all, it is JavaScript code that is the root cause of client-side hidden-web content. However, this behaviour can be explained using a simple example as the one used in Section 2: Figure 1 is a piece of JavaScript code that can cause many hidden-web states and much hidden-web content, although the amount of code is relatively small; In this simple example all the state updates are retrieved in small HTML deltas from the server, and injected into the DOM tree through a small piece of JavaScript code. In fact, we have witnessed this kind of behaviour in many of the examined websites that have client-side hidden-web characteristics.

7 Discussion

In this section, we discuss some of the threats to validity, limitations, and implications of our findings.

Client-Side Scripting. Plugin-based Rich Internet Applications (RIA) such as Adobe Flash and Silverlight have their own client-side scripting languages that induce hidden-web content. The main focus of our work, however, was standard-based technologies and therefore we limited our study to only JavaScript initiated client-side hidden-web content.

Clickable Types. Through JavaScript event-driven programming any HTML element can potentially become a clickable item. In this study, we included six of the most common HTML elements used as clickables. We made our selection based on a small pilot study we conducted on ten Alexa websites. Other clickable types (e.g., `P`, `TD`) could also potentially induce client-side hidden-web content, which we have not analyzed. The inclusion of other clickable types can probably marginally increase the hidden-web percentage.

Crawler. We extended and used Crawljax [17] to crawl client-side hidden-web content. Using a different crawler could result in different outcomes. However, to the best of our knowledge, Crawljax is currently the only available open source tool capable of crawling JavaScript-based applications.

Event Types. Our study is constrained to the `click` event type. We believe this is the most commonly used event type in practice for making event-driven

transitions in Web 2.0 apps. However, the DOM event model has many other event types, e.g., *mouseover*, *drag and drop*, which can potentially lead to hidden-web states. This is part of our future work.

Number of States Examined. To be able to have a fair analysis in a timely manner, we constrained the maximum number of states to crawl for each website to 50. There were a few websites that did not have that many states to crawl. In those cases, we analyzed the websites according to the number of states available. Choosing a different maximum number could theoretically impact our evaluation results, although we do not have any evidence that that would be the case (because of the randomization).

Representativeness. We have collected data for 500 websites. To obtain a representative sample and minimize selection bias, we collected 400 URLs from Alexa and 100 randomly from the wild. For the same reason, we randomized the candidate clickable selection while crawling, to make the state exploration of each website unbiased.

Reproducibility. Our tool implementation, Javis, the list of all websites used in our study, as well as all the empirical data are available for download, making the study fully replicable.

Implications. Our study shows that there is a considerable amount of data that is *hidden* due to client-side scripting. The hidden content is increasing rapidly as more developers adopt modern Web 2.0 techniques to implement their web applications. We believe more research is needed to support better understanding, analysis, crawling, indexing, and searching this new type of hidden-web content. In addition, web developers need to realize that by using modern techniques (e.g., JavaScript, Ajax, HTML5), a large portion of their content becomes hidden, and thus unsearchable for their potential users on the web.

8 Conclusion

With the advent of Web 2.0 technologies, an increasing amount of the web application state is being offloaded to the client-side browser to improve responsiveness and user interaction. Through the execution of JavaScript code in the browser, the DOM tree representing a webpage at runtime, is incrementally mutated without requiring a URL change. This dynamically updated content is inaccessible through general search engines, and as a results it becomes part of the hidden-web portion of the Web.

In this paper, we presented the first empirical study on measuring and characterizing the hidden-web induced as a result of client-side scripting. Our study shows that client-side hidden-web is omnipresent on the web. From the 500 websites we analyzed, 476 (95%) contained some degree of hidden-web content. In those websites, on average 63% of the states were hidden, and per hidden state, we measured an average of 19 kilobytes of hidden content from which 0.60 kilobytes is pure textual content. The `DIV` element is the most commonly used clickable to induce client-side hidden-web content, followed by the `SPAN` element.

This points to the importance of including the examination of such elements in modern crawling engines and going beyond link analysis in anchor tags.

In future work, we will expand the list of websites in our analysis. We also intend to study the effects of other event-types (e.g., mouseover) and HTML5 (e.g., canvas) on the amount of client-side hidden-web content.

References

1. Alexa top sites, `http://www.alexa.com/topsites/`
2. Alvarez, M., Pan, A., Raposo, J., Vina, A.: Client-side deep web data extraction. In: Proc. of the Int. Conf. on E-Commerce Technology for Dynamic E-Business, pp. 158–161. IEEE Computer Society (2004)
3. Barbosa, L., Freire, J.: An adaptive crawler for locating hidden-web entry points. In: Proc. of the 16th Int. Conf. on World Wide Web (WWW), pp. 441–450. ACM (2007)
4. Bergman, M.: White paper: the deep web: surfacing hidden value. Journal of Electronic Publishing 7(1) (2001)
5. Chang, K.C.-C., He, B., Li, C., Patel, M., Zhang, Z.: Structured databases on the web: observations and implications. SIGMOD Rec. 33(3), 61–70 (2004)
6. Choudhary, S.R., Versee, H., Orso, A.: WebDiff: Automated identification of cross-browser issues in web applications. In: Proc. of the 26th IEEE Int. Conf. on Softw. Maintenance (ICSM 2010), pp. 1–10 (2010)
7. Dasgupta, A., Ghosh, A., Kumar, R., Olston, C., Pandey, S., Tomkins, A.: The discoverability of the web. In: Proc. of the Int. Conf. on World Wide Web (WWW), pp. 421–430. ACM (2007)
8. de Carvalho, A.F., Silva, F.S.: Smartcrawl: a new strategy for the exploration of the hidden web. In: Procs. of the ACM Int. Workshop on Web information and Data Management, pp. 9–15. ACM (2004)
9. Duda, C., Frey, G., Kossmann, D., Matter, R., Zhou, C.: Ajax crawl: making Ajax applications searchable. In: Proc. Int. Conf. on Data Engineering (ICDE 2009), pp. 78–89 (2009)
10. Gentleman, R., Ihaka, R.: The R project for statistical computing, `http://www.r-project.org`
11. He, B., Patel, M., Zhang, Z., Chang, K.: Accessing the deep web. Communications of the ACM 50(5), 94–101 (2007)
12. Hsieh, W., Madhavan, J., Pike, R.: Data management projects at Google. In: Proc. of the Int. Conf. on Management of Data (SIGMOD), pp. 725–726 (2006)
13. Krishnamurthy, B., Wills, C.: Cat and mouse: content delivery tradeoffs in web access. In: Proc. of WWW, pp. 337–346. ACM (2006)
14. Lage, J.P., da Silva, A.S., Golgher, P.B., Laender, A.H.F.: Automatic generation of agents for collecting hidden web pages for data extraction. Data Knowl. Eng. 49(2), 177–196 (2004)
15. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's deep web crawl. Proc. VLDB Endow. 1(2), 1241–1252 (2008)
16. Mesbah, A., Mirshokraie, S.: Automated analysis of CSS rules to support style maintenance. In: Proc. of the 34th ACM/IEEE Int. Conf. on Softw. Eng. (ICSE), pp. 408–418. IEEE Computer Society (2012)

17. Mesbah, A., van Deursen, A., Lenselink, S.: Crawling Ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web (TWEB) 6(1), 3:1–3:30 (2012)
18. Mesbah, A., van Deursen, A., Roest, D.: Invariant-based automatic testing of modern web applications. IEEE Trans. on Softw. Eng. (TSE) 38(1), 35–53 (2012)
19. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: Proc. of the Int. Conf. on Very Large Data Bases (VLDB), pp. 129–138 (2001)
20. Yue, C., Wang, H.: Characterizing insecure JavaScript practices on the web. In: Proc. of the Int. World Wide Web Conf (WWW), pp. 961–970. ACM (2009)