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

by

Zahra Behfarshad

B.Sc., Islamic Azad University Tehran Central Branch, 2011

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

May 2014

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. 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.

Preface

This thesis presents an empirical study of client-side hidden-web content conducted by the author under the supervision of Dr. Ali Mesbah. I was responsible for implementing the tool, collecting URLs, running the experiments, evaluating and analyzing the results, and writing the manuscript. My supervisor guided me with the creation of the experimental methodology, the analysis of the results, as well as editing and writing portions of the paper.

The results of this study were published as a full paper in the Proceedings of the International Conference on Web Engineering (ICWE) [4] in 2013. We are also proud to announce that our paper received the **Best Paper Award** at ICWE 2013.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my advisor Dr. Ali Mesbah for giving me this opportunity to be a part of his lab. His knowledge, motivation, inspiration, support and kindness through out my years of studying always encouraged me to think more critically and focus on my work more professionally. During his honourable supervision I learnt many things and appreciate it. I am proud to say that these years were one of the best years of my life and the most memorable.

I would also like to thank my family specially my parents for their unlimited support and love. Without them pursuing this degree would have been very difficult and cumbersome. My brothers, Alireza and Mohammad, too helped me become happy and keep my hopes up.

Finally, I would like to appreciate from a very dear friend, Shabnam Mirshokraie, for all of her support, kindness and guidance. She always motivated me to follow my dreams and never give up.

Last but not least, I also like to thank Prof. Philippe Kruchten and Prof. Karthik Pattabirman for accepting to be apart of my defence committee.

Dedication

To my family and friends

Chapter 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, 29], 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 [5]. More recently, form-based hidden web content has been estimated at several millions of pages [3, 17].

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 extensive research on detecting [3, 20, 29] and measuring [5, 15] 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 [31], 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. In most Web 2.0 applications, JavaScript code extensively interacts with and incrementally updates the Document Object Model (DOM) at runtime in order to present new state changes seamlessly. 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.

1.1 Objective

Our goal is to measure the pervasiveness and characterize the nature of hidden-web content induced by client-side JavaScript in today's web applications. Therefore, the main focus of our study is to understand how client-side scripting languages, specially JavaScript , contributes to hidden-web content by measuring the percentage of hidden-web in websites. In addition, we attempt to identify any correlation between the hidden web content and the DOM size along with the JavaScript custom code size.

For simplicity, we will refer to this type of hidden-web content as **client-side hidden-web** throughout the thesis. To the best of our knowledge, we are the first to conduct an empirical study on this topic.

1.2 Contributions

Our study makes the following main contributions:

- A systematic methodology and tool, called JAVIS, to automatically analyze and measure client-side hidden-web content;
- An empirical study, conducted on 500 online websites, pointing to the ubiquity and pervasiveness of this type of hidden-web content in today's web-

sites. Our results show that **95%** of the websites we analyzed contain some degree of client-side hidden-web content; On those websites with client-side hidden-web, on average (1) **62%** of the crawled web states are hidden, (2) there is around **19 kilobytes** of DOM content that is hidden per hidden state whereas **0.6 kilobytes** are only content, (3) the `DIV` element is the most commonly used (**61%**) clickable type contributing to client-side hidden-web content;

- A discussion of the implications of our empirical findings. Our study indicates that there is a possible correlation between the size of the DOM tree and the hidden-web content, but surprisingly, no strong correlation exists between the amount of JavaScript code and client-side hidden-web.

The results of our study were published as a full paper [4] in the Proceedings of the International Conference on Web Engineering (ICWE) in 2013, which received the **Best Paper Award**.

1.3 Thesis Organization

In Chapter 2, we provide background information regarding web applications, particularly with respect to the use of client-side JavaScript , along with the motivation to conduct this study. Chapter 3 discusses the related work in this area of research. Chapter 4 describes in detail the experimental methodology used to measure the hidden-web content induced by JavaScript . In Chapter 5, the method used to pursue our research questions along with how we analyze the collected data to perform the evaluation is provided. Chapter 6 presents the results acquired from this study and answers the research questions. Chapter 7 discusses the implications our results have on web application programmers, testers, and tool developers, and some of the validity threats in our study. Finally, Chapter 8 concludes our work and presents future research directions.

Chapter 2

Background and Motivation

2.1 The Hidden-Web

With the rapid growth of data embedded inside web applications, the Internet has become a main source of information today. To retrieve relative information from billions of existing web applications, search engines require a means to crawl and index the data efficiently and effectively.

In reality, search engines employ what is called *crawlers* or *spiders*. These crawlers automatically inspect web pages and create a copy of the visited pages. These actions are repeated daily to index useful content within the web application or simply update the resources in order to facilitate future searching. Therefore, search engines only recall data within pages which were previously toured by crawlers.

A question regarding search engines and indexing is, how much of the Web content can be searched and retrieved by the current search engines? In order to answer this question, we first explain two divisions of the Web: The *Visible Web* and the *Hidden Web*.

The visible Web (also known as the surface/indexable Web) [5] is the portion of the Web where search engines are capable to index and retrieve the content. In contrast, the hidden Web (also known as the deep/invisible Web, deepnet, or

darknet) [5, 29] is the part in which conventional search engines are not able to index the content and thus, it is invisible to the users through search engines.

There are eight major causes for the deep Web: *dynamic content*, *unlinked content*, *private text*, *contextual Web*, *limited access content*, *scripted content*, *non-HTML/text content* and *text content using Gopher/FTP protocol*. Each of these causes are described as the following:

Dynamic Content. Dynamic content refers to dynamic generation of information which are returned in response to a submitted query or accessed only through a form.

This type of content is associated with forms, databases and the data store within the databases. Once the user completes a form and submits it, related information corresponding to the search query is retrieved from the databases and presented. Today many websites are using databases and forms to benefit from the advantages it provides for both users and the website owners.

A web application might be composed of one or many databases either entirely connected or independent. Either way, it is very difficult for crawlers to gain access to the databases and extract information. Thus, the information reserved inside databases are indeed aggregated to the hidden-web.

Unlinked Contents. Any information staged within web pages which are not linked to by other pages become hidden from the user. Clearly, if the web page is not linked to other web pages of a website, crawlers won't gain any access to them since there is no connection between the web pages to continue with the crawling.

Private Websites. Private websites are described as websites that require registration and login (password-protected resources). Today, many of the websites require their users to acquire a username and password to be able to obtain access after signing in. This can be seen in forums and social networks where people tend to communicate with another.

One of the goals of requesting registration and logging in is to protect the user's information and responses from adversaries. Although it is rationale, yet it prevents the crawlers from reaching the content within the websites and therefore, the data shift towards the hidden-web.

Contextual Web. Refers to pages with content varying by different access contexts. This means that based on the different access types, for example the location of the user or the IP address, different content is presented. Since the web page might contain and display a variety of details and data, search engines will be able to index a partial of the data and the rest grow inside the hidden-web.

Limited Access Content. Limited access content is sites that limit access to their pages in a technical way. What is meant by technical way, are any means that prohibit search engines from inspecting the web pages such as CAPTHCAs or using the robots exclusion standard.

CAPTCHAs are usually used to detect whether it is a real user or simply a robot. To pursue this, each time a random statement (either a questions is asked or a word) is generated by default which the user has to type. Since the statements are generated randomly, robots cannot reply and thus, the website will not be browsed. Regarding the exclusion standard, web developers have the ability to block out the crawlers from investing their web applications.

Scripted Content. Scripted content is any type of information that is produced by scripting languages such as the most popular scripting language, JavaScript . This content is usually generated dynamically and since the search engines do not execute JavaScript code they become hidden content.

Non-HTML/Text Content. Non-HTML/text content is textual contents encoded within multimedia. It is obvious, that any text embedded within videos,

audio and images cannot be searchable since they are not crawled in the beginning.

Although there are many reasons behind hidden-web content as mentioned above, yet only a few of them are studied. These studies [3, 11, 12, 20, 22, 29, 30]. mostly focus on the portion of hidden-web content behind forms and databases and other causes have gained little attention. In this study, we focus on the content dynamically generated by scripting languages and in particular by JavaScript . As mentioned before, our goal is to understand how JavaScript contributes to and correlates with hidden content on the web.

2.2 JavaScript Background

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.

JavaScript plays an important role when interacting with web applications. It is the language of the browser and by writing JavaScript code, we can simply generate dynamic code and thus, enhance the responsiveness and interactions of the web applications.

Although JavaScript might have many disadvantages, such as increasing vulnerability in web applications and browsers, yet the advantages overweight them. One of the most powerful abilities it provides is the capability of easily modifying the DOM tree.

By using a JavaScript function, the developer can simply gain access to any DOM element within the web page. Once the element is obtained, based on the purpose any sort of alteration can be carried out. These modifications can be of any sort such as removing or appending new nodes to an element, modifying the textual content of the element, altering the attributes of the element or their values and so on. A clear example of DOM modification is provided in section 2.4.

The JavaScript engine interprets and executes the relative code and the DOM

tree will be updated. Once the update is complete, the browser interprets the new DOM tree and the web page is reconstructed containing new data.

2.3 Document Object Model (DOM)

The fundamental and main part of a web application is the HTML elements. These elements are the crucial components for a web application to exist. There is no limitation on how and where to use the elements and it only varies in terms of the design and functionality of the web page.

The DOM, is a model representing the structure of the HTML elements. It describes how the elements are connected with each other, and how you can gain access to them.

By calling an element within a web page with its appropriate name, the developer can yield knowledge of its attributes, their values and the text it holds. In the DOM tree, each element is referred to as a *node*. A node can be either of these two cases: be both a parent node and a child node or only be a child node. The DOM tree will also present the attribute nodes. In other words, not only the elements and text are added to the DOM tree, but also the attributes of the elements can be appended.

One of the powerful advantages the DOM tree provides for the developer is the ability to walk the DOM tree, identify the element she desires and modify it. This action can be pursued easily by simply obtaining a node. However, in order to alter the DOM tree, specific API (Application Programming Interface) are required. These API are used in scripting languages such as JavaScript , for example "getElementById('news')" in the following example.

2.4 Hidden-content Example

We present a simple example of how JavaScript code can induce hidden-web content by dynamically changing the DOM tree. Figure 2.1 depicts a JavaScript

```
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); });   })↩
           });
```

Figure 2.1: 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>
```

Figure 2.2: The initial DOM state.

code snippet using the popular jQuery library.[1] Figure 2.2 illustrates the initial state of the DOM before any modification has occurred.

Once the page is loaded (line 1 in Figure 2.1), 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 2.3. 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.

[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>
```

Figure 2.3: The updated DOM tree after clicking on 'Update!'.

2.5 Client-Side Hidden-Web Content

Client-side scripting empowers achieving dynamic and responsive web interfaces in today's web applications. The most widely used language for client-side scripting is JavaScript , which is an event-driven dynamic and loosely typed interpreted programming language.

JavaScript is supported by all modern web browsers across multiple platforms including desktops, game consoles, tablets, and smartphones. Through JavaScript , the client-side runtime DOM tree of a 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-based manipulation through JavaScript increases responsiveness of web applications, these dynamically injected contents 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. Moreover it is a basis for AJAX-based web applications, which has become very popular in the past decade.

AJAX (Asynchronous JavaScript and XML) is an umbrella term for some

technologies which allow asynchronous server communication without requesting a completely new version of the page. Some well-known representative examples include Gmail and Google Docs.

In modern web applications, however, the 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 basically crawl and extract hypertext links and index the resulting HTML code recursively.

Chapter 3

Related Work

In order to discuss previous work in more details, we classify the related work into two major categories: (1) crawling the hidden-web, and (2) measuring the hidden-web.

3.1 Crawling the Hidden-Web

Crawling techniques have been studied since the advent of the Web itself [6, 7, 9, 16, 28]. 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, 11, 12, 20, 22, 29, 30].

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. HiWE [29] is one of the very first proposed hidden-web crawlers which tries a combination of different query values for HTML forms.

Ntoulas et al. [27] focus on an effective hidden-web crawler that discovers and downloads pages by simply generating queries based on pre-defined search policies. Barbosa et al. [3] propose an adaptive crawling strategy to locate online

databases by searching extensively through web forms, avoiding crawling any unrelated pages.

Carvalho et al. [12] present a method, called SmartCrawl, for retrieving information behind HTML forms by automatically generating agents that fill out forms. Similarly, Palmieri et al. [20] automatically generate fetching agents to identify hidden-web pages by filling out HTML forms.

Liakos and Ntoulas [21] have recently proposed a topic-sensitive hidden-web crawling approach. Khare et al. [18] provide a comprehensive survey of the research work in search interfaces and keywords.

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 importance and challenges of crawling client-side hidden-web. Mesbah et al. [24, 25] were among the first to propose an automated crawler, called CRAWLJAX, for eAJAX-based web applications. CRAWLJAX automates client-side hidden-web crawling by firing events and analyzing DOM changes to recursively detect a new state. Duda et al. [13] presented how DOM states can be indexed. The authors proposed a crawling and indexing algorithm for client-side state changes.

3.2 Measuring the Hidden-Web

Researchers have reported their results of measuring the hidden-web behind forms. In 2001, Bergman [5] reported a study indicating that the hidden-web was about 500 times larger than the visible web.

In 2004, Chang et al. [8] 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. [15] 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.

Chapter 4

Methodology

In this chapter, the research questions and the fundamentals of the methodology considered for this study are each explained separately in the following sections. It should be noted, the approach used for analyzing the subjects of the experiments are explained in full description in the next chapter.

4.1 Research Questions

The main goal of our empirical study is to measure the pervasiveness and characterize the nature of hidden-web content induced by client-side scripting. By understanding these characteristics we gain knowledge of how much information is hidden, whether it is possible to reduce this amount of hidden content and transfer them towards visible content. It should be noted that we only focus on the onclick event type in this study since it is most widely used among web application.

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?

4.2 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 those websites that does contain hidden-web content, we measure what percentage of the crawled states is hidden.

To measure the amount of content that is hidden (**RQ2**), we compute the total and average of hidden content in terms of *differences* between each hidden state and its previous state regardless of whether it is hidden or visible. It should be noted, we consider two cases, hidden content only containing the textual differences and hidden content comprising of both textual and DOM differences.

Regarding **RQ3**, we analyze the distribution of the elements used among all web applications at first. Furthermore, we classify the clickable elements that exercising them results in hidden states in our analysis to address this research question. In other words, 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 correlations between the client-side hidden-web content and the average DOM size along with custom JavaScript code of each website examined. In the next chapter, technical details of our analysis approach are explained clearly.

4.3 Experimental Objects

In this study, we analyze 500 unique websites in total. To obtain a representative pool of websites, similar to other researchers [19, 31], we select 500 unique websites from Alexa's Top Sites [1] (henceforth referred to as ALEXA). However,

ALEXA contains websites that are exactly the same (same domain) but hosted in different countries. Therefore, 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. This leads to a total number of 400 objects in our list.

Since the 400 websites are all selected from ALEXA based on popularity, we gather another 100 random websites additionally using Yahoo! random link generator (henceforth referred to as RANDOM), which is also used in other studies [10, 23]. The purpose of taking this action is to gain a more generalizable conclusion after evaluating the results.

It should be noted all the 500 websites (henceforth referred to as TOTAL) were crawled and analyzed throughout February 2013.

Chapter 5

Client-Side Hidden-Web Analysis

Figure 5.1 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

5.1.1 State Exploration.

To automate the crawling step, we use and extend our AJAX crawler, called CRAWLJAX [25]. CRAWLJAX is capable of automatically exploring JavaScript -induced DOM state changes through a dynamic event-based crawling technique. Our approach for automatically exploring a web application's state space is based on our CRAWLJAX [25] 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 2.0 AJAX-based web applications. By inserting random or user-specified data and firing events on the web elements and

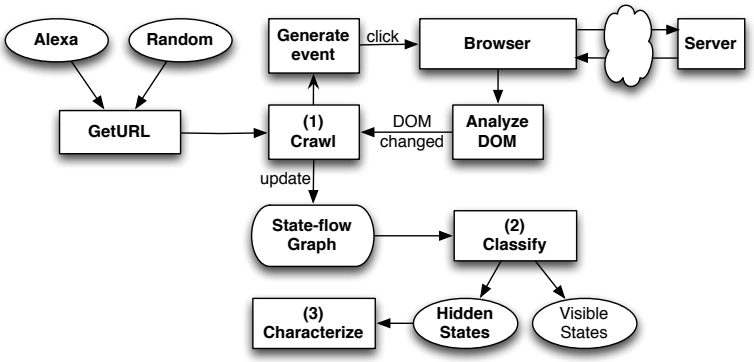

Figure 5.1: Overview of our client-side hidden-web analysis.

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) [25] capturing the client-side states and possible event-based transitions between them. This state-flow graph is defined as follows:

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

$\boldsymbol{r}$ is the root node (called Index) representing the initial state when **A** has been fully loaded into the browser.

$\boldsymbol{V}$ is a set of vertices representing the states. Each $\boldsymbol{v} \in \boldsymbol{V}$ represents a runtime DOM state in **A**.

$\boldsymbol{E}$ 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 $\boldsymbol{v}_1$.

$\mathscr{L}$ is a labelling function that assigns a label, from a set of event types and DOM element properties, to each edge.

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 [25, 26].[1]

5.1.2 Crawling Configuration.

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

Maximum states. To constrain the state space and still acquire a representative sample for our analysis, we define an upper limit on the number of states to crawl for each website, namely, 50 unique DOM states.

Setting the maximum number of DOM states to 50, is based on a pilot study conducted beforehand. Since it was only a pilot study, 10 random websites with different rankings were selected and crawled for four rounds. At the first round the maximum state crawling configuration was set to 25 states and in each round afterwards, 25 more states were added. Therefore, each website was crawled with 25, 50, 75 and 100 states. However, it should be noted that not all websites resulted to 75 or 100 unique DOM states. In those cases that did include 100 distinctive states, the hidden-web state percentage was no different from the hidden-web state percentage of the crawls with 50 states. Therefore, by analyzing the hidden-web state percentage and the number of states of all four rounds, the conclusion was to set the maximum states to 50.

Crawling depth. Similar to other studies [15], we set the crawling depth to 3.

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

[1] http://crawljax.com

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.

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.

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

5.2 Classification

As shown in Figure 5.1, 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 search engines and, therefore, it is visible;

2. A non-URL-based state transition is not crawled nor indexed by search engines and thus, it ends up in the hidden-web; For instance, the DOM update presented in Figure 2.3, as a result of clicking on the `DIV` element of Figure 2.2, is hidden.

To classify the crawled states, we traverse the inferred state-flow graph of each website. For each state, we analyze all the incoming edges (i.e., clickables). If none of the incoming edges is a valid URL-based transition, we consider that state to be a hidden state. Otherwise, it is visible.

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

We will analyze the following characteristics:

5.3.1 Hidden-Web Quantity

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 the *unix difference*. To measure the amount of data that can be hidden, the differentiation 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.

5.3.2 Clickable Types

Clickable types: To investigate which clickable type (i.e., `A`, `DIV`, `SPAN`, `IMG`, `INPUT` and `BUTTON`) contribute most to 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.

5.3.3 Correlations

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 [14] to calculate the non-parametric Spearman correlation coefficients (r) as well as the p-values (p), and plot the graphs. We present combinations that indicate a possible correlation.

Before going into details regarding each each aspect, a brief description of how we intend to evaluate them is described. We use the *Pearson correlation* which is a widely used measure of linear dependency between two variables.

The Pearson correlation coefficient is obtained by dividing the covariance of the two variables by the product of their standard deviations. The more it is close to +1, the more positive (increasing) linear relationship exists, and the more it is close to -1, the more negative (decreasing) linear relationship exists. As it approaches zero there is less of a relationship (closer to uncorrelated). We should

note that a high correlation does not necessarily infer a causal relationship between the variable, though it can indicate the potential existence of causal relations.

DOM Size: To find the dependence and effectiveness of DOM elements (Href, Div, Span, Img, Input and Button) on the percentage of invisibility, we study the statistical relationship between the DOM elements as random variables and the hidden-web rate. It should be noted that a web site containing a huge amount of elements that directs to large DOM size does not necessarily contribute to higher invisibility rate and vice versa. The DOM size obtained for each web application is an average of the total DOM size and is presented in terms of kilo bytes.

JavaScript code size: In order to find the correlation between JavaScript code size and hidden-web, a tool is used with the specific purpose of measuring this parameter. Although many tools exist that can aid us in this regard yet we chose the *Web Developer* tool which is an open source tool which easily provides us with the information required.

With the aid of this tool, the JavaScript code size is can be obtained as compressed or uncompressed size in kilo bytes based on the users opinion and therefore, in our case we use the compressed size. Note that the JavaScript code size provided by Web Developer includes all the script files which can be constituted of any JavaScript library used for that web application along with the custom JavaScript files written by the developer.

Although we obtain Pearson correlation coefficient between this aspect and the hidden-web, yet we should keep in mind that the result might not be so remarkable. This does not mean that if code size is small than it leads to more hidden content and vice versa.

Categories: An advantage that ALEXA provides us with is the opportunity to simply classify the collected web sites into the already specified categories defined by them.

ALEXA provides three different options to choose from the top best web site title which are: 1) Top best web sites distributed *Globally* , 2) Top best web sites based on *Country* and 3) Top best web sites based on *Category*. Since our goal is to obtain comprehensive data, we selected the first option in order to present us with the universal web sites. However, in order to understand the correlation between the categories and hidden web, we sort the obtained web sites into the pre-defined categories too.

There are 17 categories in general which can be found at Alexa's web site, yet for our purpose we manually grouped them into the different subjects and a total of 15 categories were concluded. In our findings, we noticed that based on Alexa's assortment, one web site can be considered to be grouped into multiple categories. It should also be noted that this action is not applied for RANDOM web sites.

5.4 Tool Implementation

We have implemented our client-side hidden-web analysis approach in a tool called JAVIS, which is open source and available for download along with all our empirical data from: http://salt.ece.ubc.ca/content/javis/

JAVIS is implemented in Java, and is built as a plugin for CRAWLJAX [25].[2] All our experiments were conducted on a Debian-based machine running Firefox as the embedded browser for CRAWLJAX.

[2] crawljax.com

Chapter 6

Results

In this section, the results of our empirical study is provided. We present our results in the following subsections, each corresponding to a research question as formulated in Chapter 4, Methodology.

Table 6.2 depicts 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 RANDOM. It should be noted that the Total column in this table refers to both the hidden DOM structure and the textual content whereas the Total Content only refers to the hidden textual content. The Average column also refers to the average hidden content and DOM structure per state. The complete set of our empirical data containing all information in the table is available for download.[1]

6.1 Pervasiveness (RQ1)

To investigate the pervasiveness of client-side hidden-web, we crawled all the 500 websites and analyzed the findings. The results are obtained distinctively for each website and as a whole for both set of websites: ALEXA and RANDOM.

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

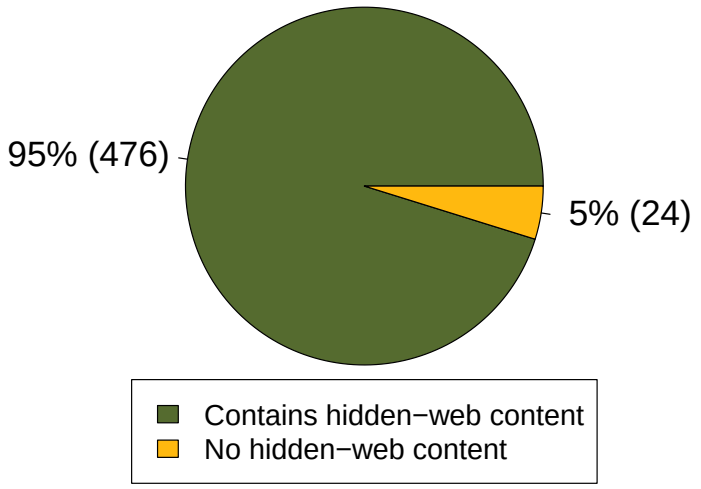

Figure 6.1: Pie chart representing the percentage of websites exhibiting client-side hidden-web content from all the 500 websites.

The pie chart in Figure 6.1 depicts the percentage of websites that fully or partially exhibit client-side hidden-web content. This means, we consider a website totally *visible* that does not contain any hidden content. In other words, if none of the crawled states are hidden, the website is referred to as a *visible* website. Unfortunately, only 5% of the websites can be crawled by search engines and thus, visible to the user. This fraction in comparison to the billions of websites that exist today in the real world, is extremely small. In contrast, 95% (476/500) of the websites analyzed exhibit some degree of client-side hidden-web content. What we mean by some degree of hidden-web content, is that they have at least one or more client-side hidden states.

In addition, we wanted to gain knowledge of what percentage of these 476 websites that lie in the portion of hidden websites, actually contain hidden-web content. To pursue this task, we separated the websites based on the resources collected and analyzed the results individually. Figure 6.2 presents three box plots illustrating the hidden-web state percentages for the two resources ALEXA, RAN-

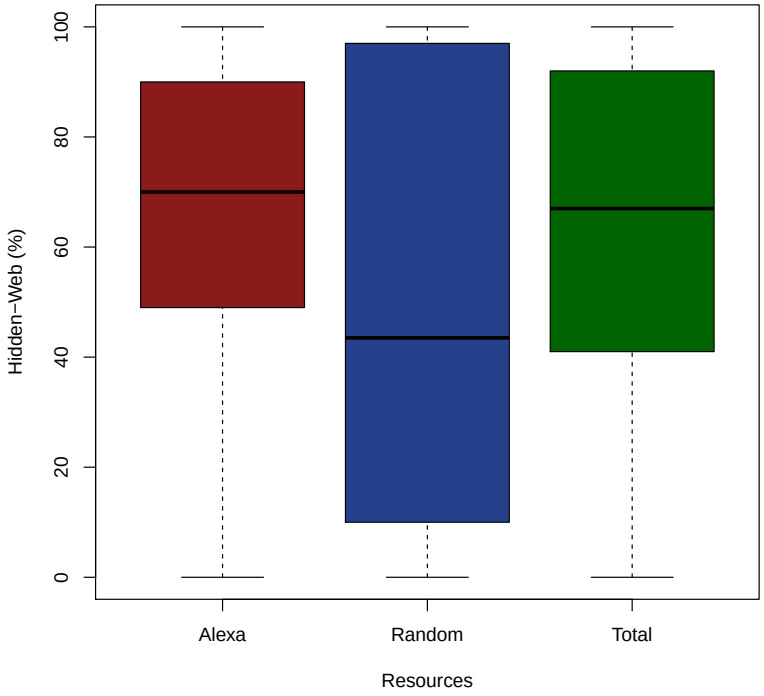

Figure 6.2: Box plots of the percentage of client-side hidden-web states in ALEXA, RANDOM, and TOTAL.

Table 6.1: Descriptive statistics of the percentage of client-side hidden-web states in ALEXA, RANDOM, and TOTAL.

Resource	Min	1st Qua.	Median	Mean	3rd Qua.	Max
ALEXA	0	49	70	65.63	90	100
RANDOM	0	10	44	50.6	98	100
TOTAL	0	41	67	62.52	92	100

DOM, and in general as the TOTAL. For more clarity and exact percentages regarding the hidden content embedded within the web applications, Table 6.1 is provided which presents the min, max, median, mean, and the 1st and 3rd quartiles of the results per resource and in total.

Table 6.2: 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

ALEXA: For the 400 websites obtained 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. These websites are among the top most visited sites in the world. It is clear that the developers of many of these websites use the latest Web 2.0 technologies, such as JavaScript , DOM, Ajax, and HTML5, to provide high quality features that come with rich interaction and responsiveness.

As can be seen in Table 6.1, the minimum and maximum of hidden-web are 0% and 100% respectively. It is obvious that some websites considered using only hypertext links whereas others preferred using only the latest technologies such as JavaScript to dynamically edit content. Therefore, the websites are either totally visible (0% hidden content) or completely hidden (100% hidden content). The first quartile, median and the 3rd quartiles are 49%, 70% and 90% respectively. Compared to the RANDOM websites, the first quartile and the median are much higher and this can be due to extensive use of new technologies to increase popularity and be rated as top best websites.

As we have discussed in Section 2, Background, these Web 2.0 techniques contribute enormously to the creation of client-side hidden-web.

RANDOM: Similarly, an average 50.6% of the states from the RANDOM websites constitute hidden states. These websites were purely randomly chosen on the Web with no previous background information about them. In other words, we do not have any idea about their rankings nor the functionality they provide. We were expecting to witness a lower percentage here, because (1) many websites on the Web might still be classical in nature, without using any modern Web 2.0 techniques; (2) the developers prefer to use more URL-based links for state transitions, rather than using JavaScript to avoid ending up in the hidden-web share. We believe that 50.6% is still a high number, pointing to the pervasiveness of client-side hidden-web on

the Web.

As discussed in the previous part and seen in Table 6.1, RANDOM in contrast to ALEXA has lesser percentage of hidden-web in the mean, median and the first quartile. However, the third quartile of RANDOM differs in 8%. Although 8% might not be significant now yet it should be noted that (1) we only analyzed 100 random websites and if more random websites were to be observed this can lead to a higher percentage and a greater difference; (2) these websites do not have any sort of rankings. One possible reasoning for this difference can be explained such that if the web application is using JavaScript and modern technologies, then it seems more calls to the JavaScript code is made and thus, the DOM is modified more extensively. However this might not be true for all websites.

TOTAL: When the results of ALEXA and RANDOM are combined in TOTAL, we witness a total hidden-web state percentage of 62.52%. Interestingly, the average of the TOTAL is very close to the average of ALEXA. It should be noted that on average, 25 minutes is required to crawl all the 50 states of a website while it takes 211 hours to crawl and classify all the 500 websites.

It is obvious that crawling more websites with a higher number of states and different types of clickable elements can certainly lead to higher percentage of hidden-web and of course doing so will be very time consuming and expensive.

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 6.3 shows the minimum, mean, and maximum amount of client-side hidden-web content for all of the crawled hidden-web states, and per hidden-web state in two cases: (1) includes both the DOM structure and the content, and (2)

Table 6.3: 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

only the content is considered. For the first case, not only do we attempt to measure the pure content, but we also consider extracting the different elements or attributes that either have been added, omitted or simply edited.

All hidden-web states crawled. For all the states crawled, we measured an average of 870 kilobytes of client-side hidden-web content including both DOM and textual content while the textual content only is 27.6 KB. The minimum and maximum for the first case (including DOM and textual content) are 0, 13170 kilobytes and 0 and 536 kilobytes for the latter case (only textual content).

It is visible from the table that the average size of all the hidden content is more than 30 times larger than the average of the textual content only. This indicates that an extensive amount of the HTML structure of the website is also altered in addition to the content change. It should be noted that, a web application containing many hidden states does not necessarily result to a huge textual difference since one web application that is comprised of a few hidden state may conclude to the same or even a larger amount of hidden content. We simply, obtain the content differences and do not compare.

Per hidden-web state. Per hidden-web state, on average 19 kilobytes of DOM and textual content exist while 0.6 was only textual content. Table 6.3 shows the minimum and maximum for this case.

Similarly, the mean size of all the hidden content per state is more than 30

times larger than the size of only the textual content. It is clear that every state does not lead to the same amount of hidden content, i.e., one state can only contribute to a few minor DOM and textual changes such as appending a new child while another state can alter (append, delete or edit) many elements, attributes and their text values.

We also attempted to analyze the textual content as well to have a better understanding of what sort of content are being altered. The textual content gathered and observed are either a singular word, numbers, short messages or a whole sentence.

The words are either (1) names of the countries so that the website can be used worldwide and users can easily change the language of the website based on their region, (2) help instructions to aid users in pursuing a task, (3) feedback or review about an object or a subject, and (4) names of items, specially items that were being sold in the website such as amazon or ebay.

The numbers range from one digit to ten digits long. These numbers act as IDs and are usually seen in websites that sell items as discussed above. The short messages indicate the limits of either the (1) user, like how many times he can listen to the songs; (2) the website like the expiration time of the website. The sentences are either normal sentences describing a subject such as the value of team work or they are questions. Either way, they are mainly providing information as news in every aspect, e.g. health issues, science, animals, videos and images, actors and stars, sports and so on.

6.3 Induction (RQ3)

To better understand what types of clickable elements web developers use in today's web applications that induce state changes in the browser, we analyzed how much each clickable type contributes to the measured hidden-web state percentage. As a reminder, these elements can be either `DIV`, `SPAN`, `INPUT`,

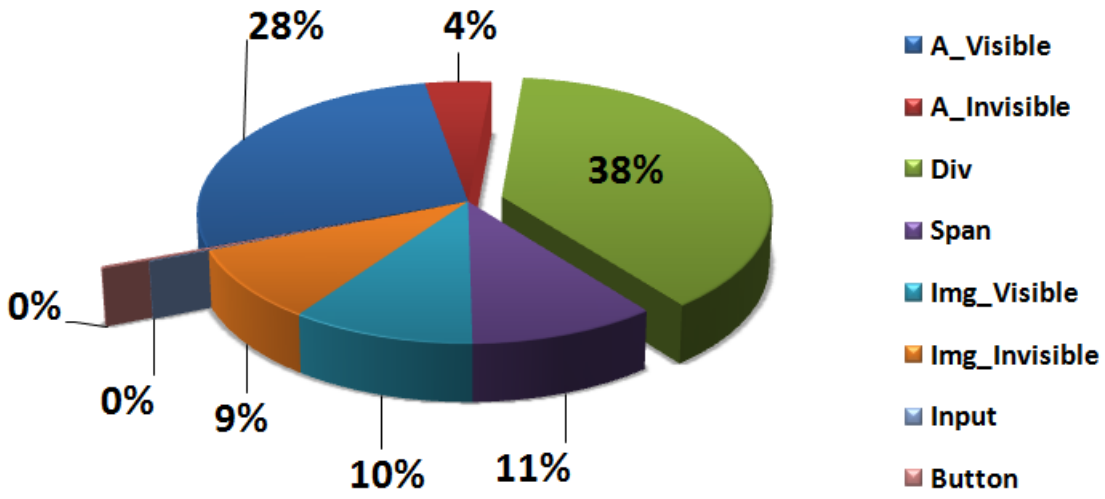

Figure 6.3: Pie chart displaying the use of different clickables throughout the 500 websites.

`BUTTON` along with the `IMG` and `A` tags.

The usage distribution of different clickables can be seen in Figure 6.3. As can be seen, approximately 40% of the websites use the `DIV` element (10327/27261), followed by the `A` tag with 32% (8734/27261) regardless of being visible or hidden. The third most used element is the `IMG` tag with 19% considering both visible and hidden states. The last three rated are `SPAN`, `BUTTON` and `INPUT` with 11%, 0.3% and 0.1% whereas 0.3% and 0.1% are being rounded down to zero.

As discussed in Section 5.3, the anchor tag (`A`) and the image element (`IMG`) can induce both visible and hidden states. Just to remember, an anchor tag is considered as an invisible element if it contributes to a new hidden state change but without owning any valid URLs. Regarding the `IMG` tag, we assess the parent tag of this element that directs to a new state change since we have observed that it is set inside other tags. If the parent tag is a visible element, in other words a visible anchor tag, the `IMG` tag is also considered as a visible element and thus, the resulting state is a visible state. Otherwise the element itself is added to the set of invisible elements and the state change is concluded as a hidden state transition. However, to answer the third research question (RQ3), we only consider the elements that cause hidden-states in our analysis.

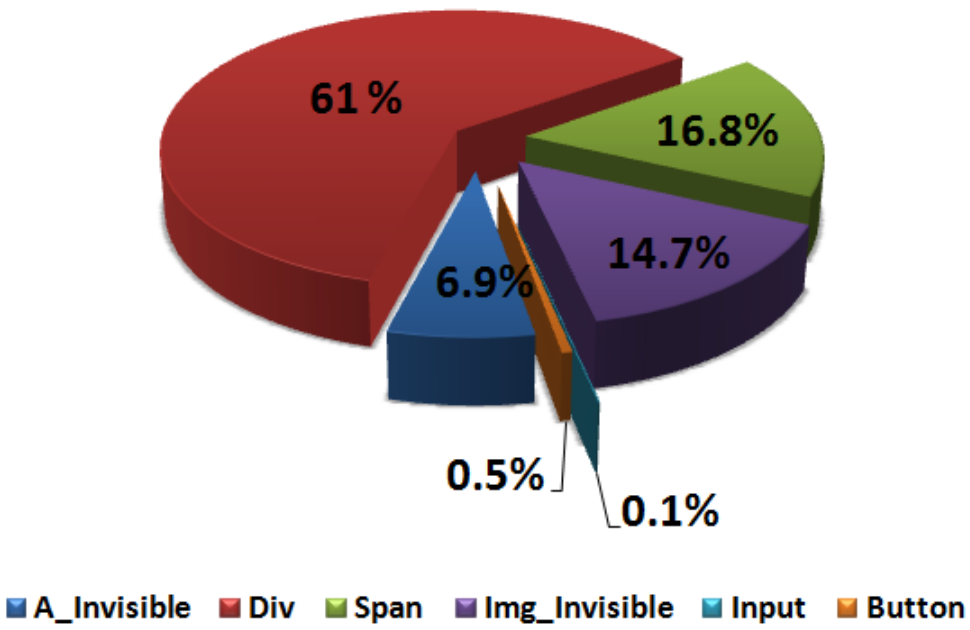

Figure 6.4: Pie chart showing hidden-web percentage behind different types of clickables. 'A_INVIS' represents anchor tags without a (valid) URL. 'MG_INVIS' represents `IMG` elements not embedded in an anchor tag with a (valid) URL.

Figure 6.4 depicts another pie chart specifically related to the different invisible clickable types associated to hidden-web state percentage. We can see that the `DIV` element 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 with `INPUT` being the least.

In final, we learn from the results that the `DIV` element, in comparison to the other elements, is more widely used in practice as a clickable.

6.4 Correlations (RQ4)

As a part of our study, we intended to investigate whether there are any correlations between the hidden-web and other characteristics of the web applications. Therefore, we considered three aspects of the websites which are explained in full details below. These aspects are ALEXA categories, DOM size and JavaScript custom code size.

Figure 6.5 displays the relationship between the ALEXA categories with the hidden-web state percentage and Figures 6-9 are scatter plots of DOM and JavaScript size against the hidden-web state percentage and content.

Alexa Rank and Categories

For the websites obtained from ALEXA, we examined their rankings and categories (e.g., Business, Computers, Games, Health, etc)[2], to learn whether any correlation with respect to the degree of hidden-web content exist.

Figure 6.5 presents the contribution of each Alexa category toward the percentage of hidden-web. As can be seen, there are 15 categories in general. Websites in the categories of *Computers* and *Regional* seem to contain the most client-side hidden-web content. Websites in the *Kids/Teen* category have the least correlation. The hidden-web percentage of other categories are in the range between 5 and 30 %. Note that each website can be a member of multiple categories on Alexa, and therefore, the distribution shown in Figure 6.5 is merely an indication. We did not witness any noticeable correlations with Alexa rank.

DOM Size and Hidden-Web State Percentage

We analyzed the DOM size corresponding to the web applications for both ALEXA, RANDOM and as a TOTAL. As can be seen in Table 6.4, the minimum, maximum and average of DOM size present in ALEXA web sites are 11, 826 and 146.2 kilobytes respectively. However, these amounts are less in websites

[2] http://www.alexa.com/topsites/category

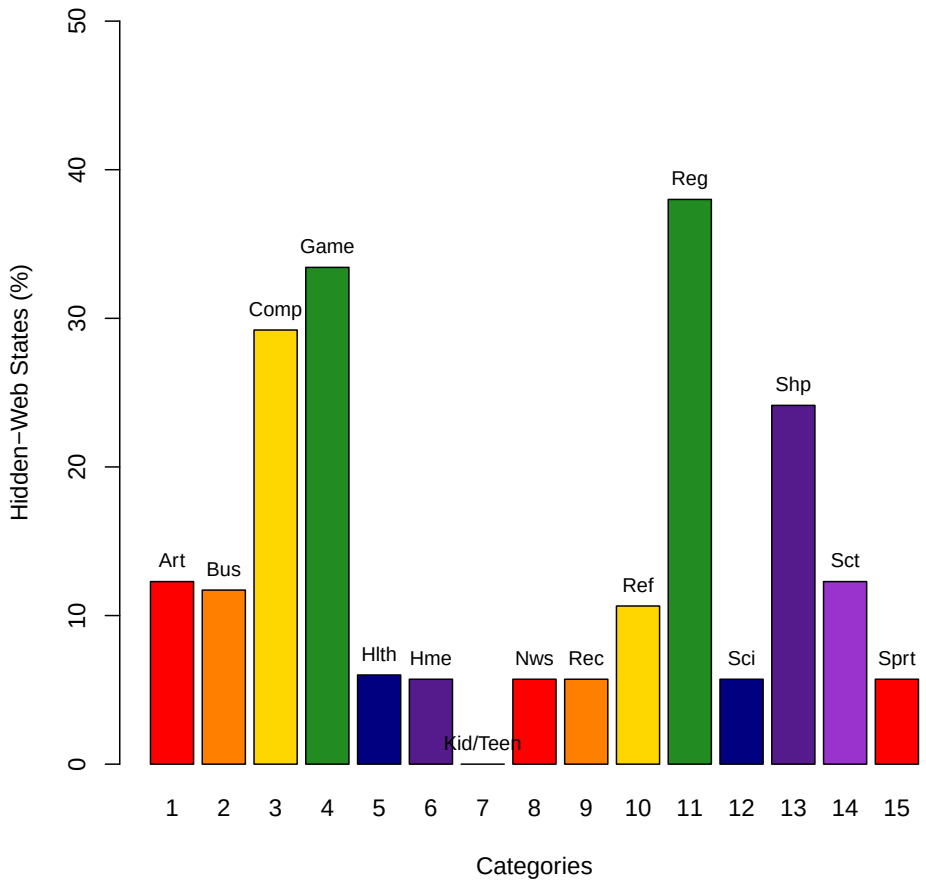

Figure 6.5: Bar chart of Alexa categories versus hidden-web state percentage.

obtained from RANDOM which are 7, 426 and 90.6 kilobytes respectively. It is obvious that websites collected from ALEXA are based on popularity and ranking and thus, contribute to more DOM size whereas RANDOM web sites are normal web leading to less DOM size.

We also conducted a correlation analysis of the degree of hidden-web state percentage with respect to the average DOM size, taken over all the crawled states.

Figure 6.6 depicts a scatter plot of the DOM size against the hidden-web state percentage. In this figure, the DOM size has a weak correlation ($r = 0.4$) with the hidden-web state percentage. A website that contributes to more hidden states does not necessarily require to be composed of an enormous DOM tree. For example, one website with 57% hidden-web state percentage has the highest DOM

Table 6.4: Descriptive statistics of the DOM size in ALEXA, RANDOM, and TOTAL.

	DOM SIZE (KB)		
Resources	**Min**	**Mean**	**Max**
ALEXA	11	146.2	826
RANDOM	7	90.62	420
TOTAL	7	118.41	826

Table 6.5: Descriptive statistics of the JavaScript custom code size in ALEXA, RANDOM, and TOTAL.

	JAVASCRIPT SIZE (KB)		
Resources	**Min**	**Mean**	**Max**
ALEXA	1	116.44	586
RANDOM	0	57.26	417
TOTAL	0	86.85	586

size (more than 800 KB), whereas the DOM size of websites with higher hidden-web percentage (90%) are less than 210 KB.

DOM Size and Hidden-Web Content

We also pursued to analyze the correlation between the degree of hidden-web content with respect to the average DOM size, again considering all the crawled states.

Figure 6.7 depicts a strong correlation ($r = 0.65$) between the DOM size with the amount of hidden-web content. As can be seen in the plot, around 80% of the websites with less than 200 KB of DOM size contribute to less than 500 KB of hidden content. And as the DOM size grew larger, the hidden content also expanded.

The correlation between these two parameters comes as no surprise, because

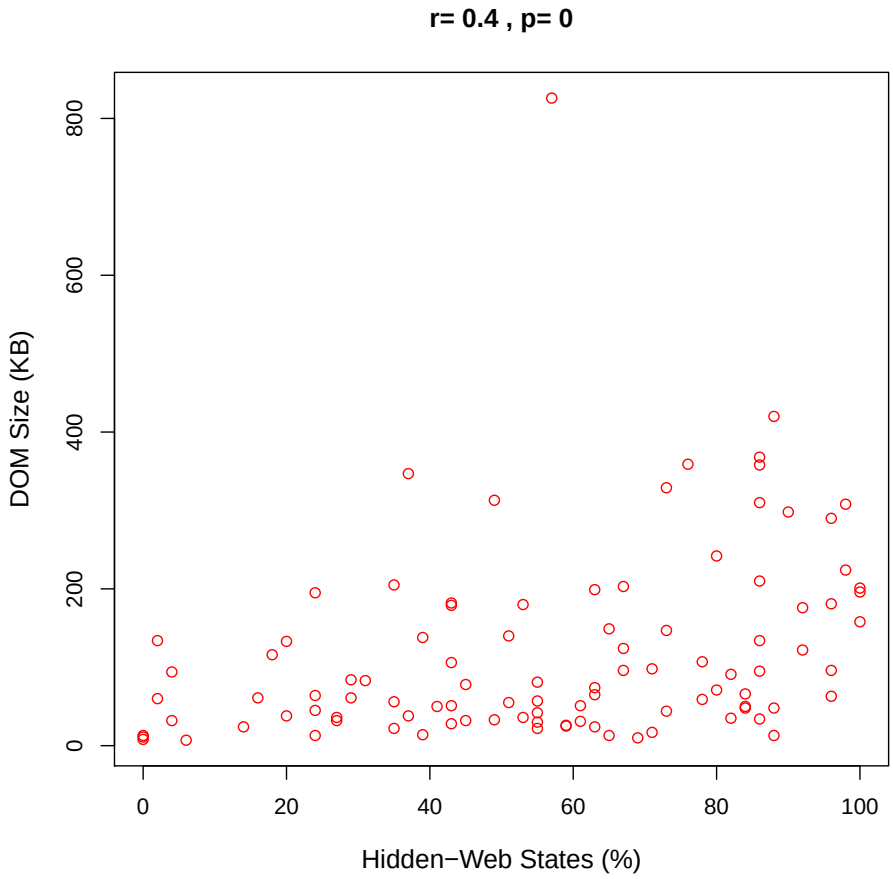

Figure 6.6: Scatter plots of the DOM size versus hidden-web state percentage. 'r' represents the Spearman correlation coefficient and 'p' is the p-value.

the larger the DOM tree, the more content there will be in a website. However, for a web application with a large DOM tree, it is not clear which type of content embedded within the application is more, the hidden content or visible content.

JavaScript Size and Hidden-Web State Percentage

First we analyzed the JavaScript custom code size corresponding to the web applications for both ALEXA, RANDOM and as a TOTAL. As can be seen in Table 6.5, the minimum, maximum and average of JavaScript code size present in ALEXA websites are 1, 116.44 and 586 kilobytes respectively. Although there might be web applications that use JavaScript often, yet there are web applications which contain zero or very little JavaScript code, as can be seen in RANDOM web

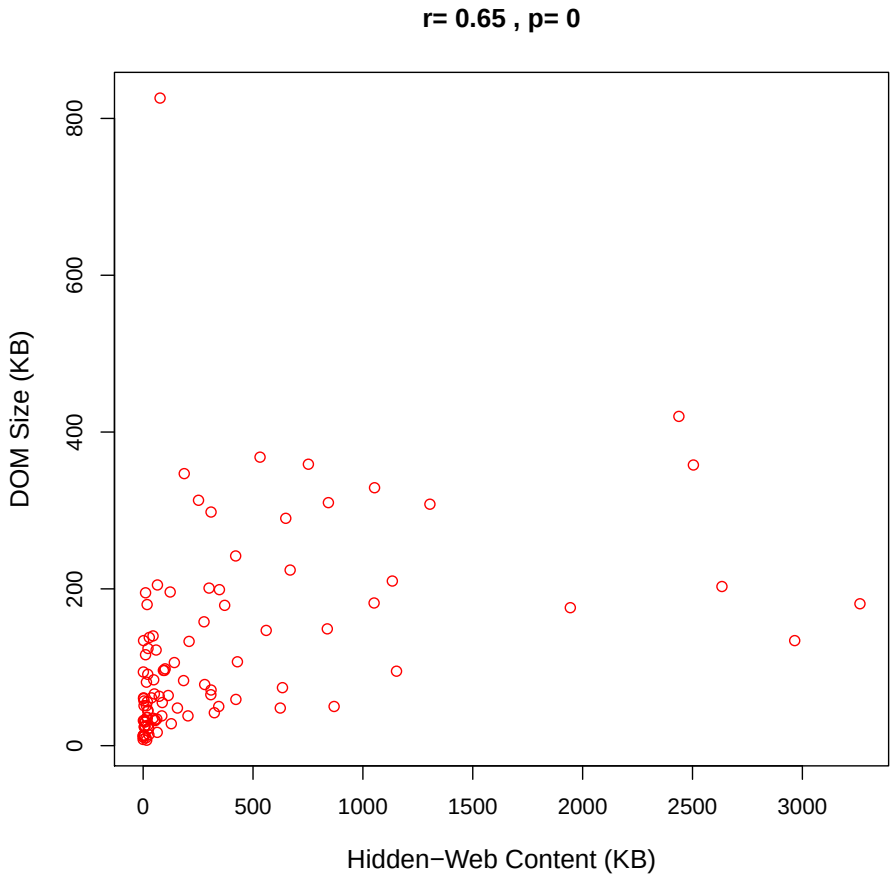

Figure 6.7: Scatter plots of the DOM size versus hidden-web content. 'r' represents the Spearman correlation coefficient and 'p' is the p-value.

applications. Some web application prefer generating dynamic content by using JavaScript , while other web developers tend to provide simple web application without the aid of JavaScript .

In Figure 6.8, around 80% of the websites have less than 100 KB of JavaScript code while the hidden-web percentage related to those websites range from 0 to 100%. The JavaScript code size of the remaining 20%, vary between 100 and 586 KB.

Although we assumed if the client-side hidden-web state percentage is high, the JavaScript code size will also be high, yet Figure 6.8 rejects our assumption by presenting the weak correlation between these two parameters. A web application does not fundamentally need to use and call many of the JavaScript code to induce hidden states. One JavaScript function that modifies the DOM tree can

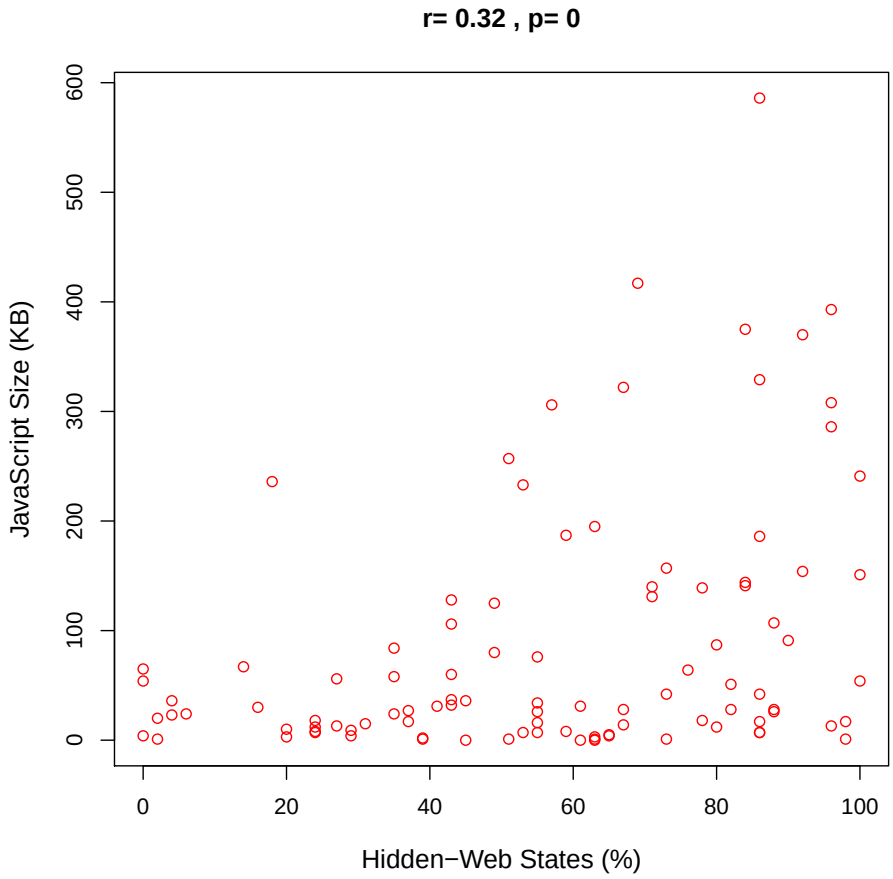

Figure 6.8: Scatter plots of the JavaScript size versus hidden-web state percentage. 'r' represents the Spearman correlation coefficient and 'p' is the p-value.

simply be executed many times by different elements and therefore, the results will be more hidden states.

JavaScript Size and Hidden-Web Content

Figure 6.9 also pinpoints a weak monotonic correlation between the JavaScript code with the hidden-web content. 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. One less likely reason to the low correlation can be due to the exclusion of popular JavaScript libraries that were used within the web application. However, this weak correlation can also be explained from another perspective.

As mentioned before, many websites use JavaScript today, yet they do not

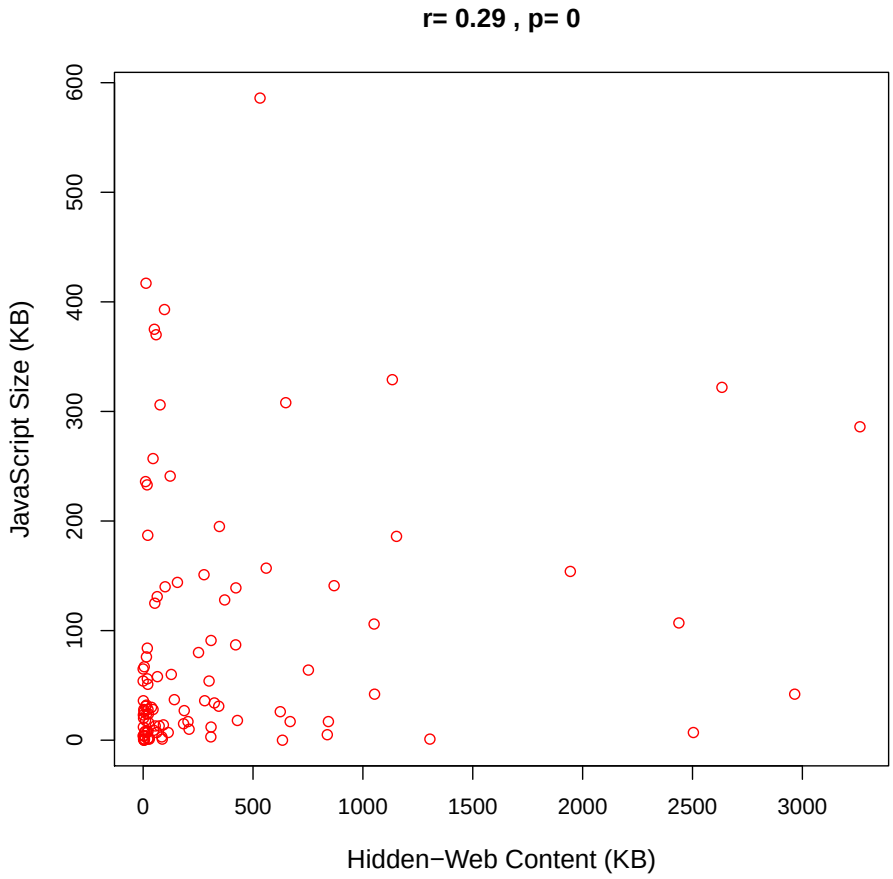

Figure 6.9: Scatter plots of the JavaScript size versus hidden-web content. 'r' represents the Spearman correlation coefficient and 'p' is the p-value.

necessarily require to have a lot of JavaScript code in order to modify the DOM tree and contribute to more hidden content. This behaviour can be explained using a simple example as the one used in Chapter 2, Background and Motivation. Figure 2.1 is a piece of JavaScript code that can causes many hidden-web states and thus, increases the hidden-web content although the amount of the 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.

Although we have an idea about the correlation between the JavaScript and DOM

size with the hidden-web content and state, yet more investigation is required in this regard. Maybe by examining more websites and inspecting their JavaScript and DOM, a more significant correlation can be met.

Chapter 7

Discussion

In previous chapters, the approach proposed for this study, the hidden-web analysis technique and the evaluation for 500 websites are explained clearly. We now have an insight on how hidden-web is induced by client-side scripting languages such as JavaScript from the results. However, as other studies, our study may not be fully accurate and risk free. There might be some aspects in our study that may be seen as validity threats which can influence the evaluation of our results and affect our conclusions. These aspects are explained in the following sections.

7.1 Threats to Validity

There are four validity threats: Internal Validity, External Validity, Construct Validity, and finally Conclusion Validity. We discuss the four validity threats in the following subsections.

7.1.1 Internal Validity

Threats to internal validity concern any issues corresponding to how the subjects are selected and treated during the experiment.

One of the factors that impacts internal validity in our study is sampling. If the websites were selected manually by the author, the final results would have

been non-exclusive and questionable. Thus, to reduce bias from our sampling, we decided to select websites from ALEXA and RANDOM, where ALEXA provides us with websites that are already ranked throughout the world and RANDOM provides us random websites without our participation.

In addition, we restricted our analysis to the 400 websites gathered from ALEXA based on *popularity*. Since the results may not hold for websites that are not popular, we considered analyzing website that are not ranked by gathering 100 websites from RANDOM.

Another internal threat is sequentially selecting and crawling the clickable elements from the candidate elements list. Although this issue might not seem problematic, yet it indeed effects the percentage of hidden-web. Thus, to mitigate this kind of threat, the list of candidate elements are shuffled before clicking. In other words, we randomize the candidate clickable selection while crawling, to make the state exploration of each website unbiased.

One way to mitigate the threats related to sampling is to select samples from different aspects such as the countries they are hosted, categories they represent, popularity and so on.

7.1.2 External Validity

External validity refers to the generalization of the study, whether the final result can be generalized outside of the scope of the study.

In order to present generalizable results, we not only analyze websites obtained from ALEXA, but also websites gathered from another resource, RANDOM. In terms of *representativeness*, we collected data for a sample of 500 websites from ALEXA where these websites are ranked as the most popular websites in the world.

There are millions of web application that exist today and compared to them we only examined a fraction. Therefore, the number of websites obtained for evaluation (500) is limited and considered as another external threat.

In addition, there are other variables that are used in the setup of CRAWL-

JAX that can affect the evaluation phase. Each of these variables are discussed separately below:

Clickable Types: Through JavaScript event-driven programming any HTML element can potentially become a clickable item. In this study we include 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 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.

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.

In-code URLs: We make the assumption that if a transition is made through JavaScript , then it is hidden. However, giant search engines such as Google, parse the website's static JavaScript code in search of valid URL's in the code to crawl. In our study, we do not explicitly take valid URLs in the code into account.

Number of states examined: To be able to have a fair analysis in a reasonable amount of time, we constrained the maximum number of states to crawl for each website to fifty. There were a few websites that did not have that many states to crawl. In these cases, we analyzed the websites according to the number of crawled states. 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).

To alleviate the generalizability issue, we can analyze more websites along with more states. Although increasing the maximum number of states and depth

may not necessarily contribute to more hidden-web content, yet it is required to gain more generalizable conclusions.

7.1.3 Construct Validity

As mentioned before the aim is to measure how often content is added to the portion of hidden-web due to the client-side scripting languages used by web developers. In step 2 of Figure 5.1, the states in the state flow graph are classified into two categories: visible-web lead to by hypertext links, and hidden-web lead to by non-hypertext links. However, there might be some cases where the hidden state is reached by external parties such as advertisers or adversaries. Today many advertisement companies embed their own JavaScript code into web applications with or without any permission and the crawler regardless of knowing where the clickable element is obtained from, clicks on it and if state transition is seen, a new state is added to the state flow graph.

In order to mitigate the construct validity, all advertisements or external code not relating to the web application itself should be removed. Although this is required yet it is a sensitive task and requires manual inspection.

7.1.4 Conclusion Validity

Conclusion validity is the degree to which the conclusions we reach about the relationships between our data are reasonable. In this study based on the frequency of the usage of client-side scripting languages, specifically JavaScript , a conclusion that similar frequency is anticipated for the entire web is reached. The reason behind this conclusion is that the web is constructed by web developers. A threat regarding conclusion validity can be the fact that we did not take automated code generation tools into consideration. However, in order to minimize this threat, our JAVIS tool is open-source and available for download as well as all the empirical data. Therefore, in terms of *reproducibility*, we provide all the necessity to fully replicate our study.

To detract the conclusion validity we should take other sources of web con-

struction such as automated code generation tools into consideration.

7.2 Implications

Our study shows that there is a considerable amount of data that is hidden due to client-side scripting. It should also be noted that many of today's advertisements produce dynamic content which can lead to hidden content too. 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.

Chapter 8

Conclusions and Future Work

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.

We present 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 62% of the states were hidden, and per hidden state, we measured an average of 19 kilobytes of hidden content whereas 0.6 kilobytes are 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 including the examination of such elements in modern crawling engines and going beyond link analysis in anchor tags.

As future work, our goal is to complement this study to gain more insight on the size and growth of the hidden-web content. In order to achieve this goal, a few

enhancements have been considered which are discussed below separately.

One of the improvements is to expand the amount of experimental objects obtained for this research so that the final results become more generalizable. Therefore, instead of only considering 100 RANDOM web sites to analyze the hidden content, we will increase this number to 400 random web sites generated automatically using a single script.

Another modification is to increase the maximum number of states examined per web site. As discussed in the previous chapters, we set the maximum number of states to 50 whereas in future we will set this to 70 states. Although we are not fully certain whether this will definitely increase the hidden web content, yet we wish to analyze the results by considering this circumstance.

Another extension is to enhance our tool to automatically obtain the custom JavaScript code within the web application. To pursue this task, we will attempt to use a proxy to easily gain access to the custom JavaScript code within the web page and record all the existing interactions, specially the custom JavaScript code.

Bibliography

[1] Alexa top sites. http://www.alexa.com/topsites/. → pages 1, 16

[2] M. Alvarez, A. Pan, J. Raposo, and A. Vina. Client-side deep web data extraction. In *Proc. of the Int. Conf. on E-Commerce Technology for Dynamic E-Business*, pages 158–161. IEEE Computer Society, 2004. → pages 13

[3] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proc. of the 16th Int. Conf. on World Wide Web (WWW)*, pages 441–450. ACM, 2007. ISBN 978-1-59593-654-7. doi:http://doi.acm.org/10.1145/1242572.1242632. → pages 1, 7, 12

[4] Z. Behfarshad and A. Mesbah. Hidden-web induced by client-side scripting: An empirical study. In *Proceedings of the International Conference on Web Engineering (ICWE)*, volume 7977 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2013. → pages iii, 3

[5] M. Bergman. White paper: the deep web: surfacing hidden value. *Journal of Electronic Publishing*, 7(1), 2001. → pages 1, 4, 5, 13

[6] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998. ISSN 0169-7552. doi:http://dx.doi.org/10.1016/S0169-7552(98)00110-X. → pages 12

[7] M. Burner. Crawling towards eternity: Building an archive of the world wide web. *Web Techniques Magazine*, 2(5):37–40, 1997. → pages 12

[8] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: observations and implications. *SIGMOD Rec.*, 33(3):61–70, 2004. → pages 13

[9] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1-7):161–172, 1998. ISSN 0169-7552. → pages 12

[10] S. R. Choudhary, H. Versee, and A. Orso. WebDiff: Automated identification of cross-browser issues in web applications. In *Proc. of the 26th IEEE Int. Conf. on Softw. Maintenance (ICSM'10)*, pages 1–10, 2010. → pages 17

[11] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *Proc. of the Int. Conf. on World Wide Web (WWW)*, pages 421–430. ACM, 2007. ISBN 978-1-59593-654-7. doi:http://doi.acm.org/10.1145/1242572.1242630. → pages 7, 12

[12] A. F. de Carvalho and F. S. Silva. Smartcrawl: a new strategy for the exploration of the hidden web. In *Proc.s of the ACM Int. Workshop on Web information and Data Management*, pages 9–15. ACM, 2004. ISBN 1-58113-978-0. doi:http://doi.acm.org/10.1145/1031453.1031457. → pages 7, 12, 13

[13] C. Duda, G. Frey, D. Kossmann, R. Matter, and C. Zhou. Ajax crawl: making Ajax applications searchable. In *Proc. Int. Conf. on Data Engineering (ICDE'09)*, pages 78–89, 2009. → pages 13

[14] R. Gentleman and R. Ihaka. The R project for statistical computing. http://www.r-project.org. → pages 23

[15] B. He, M. Patel, Z. Zhang, and K. Chang. Accessing the deep web. *Communications of the ACM*, 50(5):94–101, 2007. → pages 1, 13, 20

[16] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999. ISSN 1386-145X. → pages 12

[17] W. Hsieh, J. Madhavan, and R. Pike. Data management projects at Google. In *Proc. of the Int. Conf. on Management of Data (SIGMOD)*, pages 725–726, 2006. → pages 1

[18] R. Khare, Y. An, and I.-Y. Song. Understanding deep web search interfaces: a survey. *SIGMOD Rec.*, 39(1):33–40, 2010. → pages 13

[19] B. Krishnamurthy and C. Wills. Cat and mouse: content delivery tradeoffs in web access. In *Proc. of WWW*, pages 337–346. ACM, 2006. → pages 16

[20] J. P. Lage, A. S. da Silva, P. B. Golgher, and A. H. F. Laender. Automatic generation of agents for collecting hidden web pages for data extraction. *Data Knowl. Eng.*, 49(2):177–196, 2004. ISSN 0169-023X. doi:http://dx.doi.org/10.1016/j.datak.2003.10.003. → pages 1, 7, 12, 13

[21] P. Liakos and A. Ntoulas. Topic-sensitive hidden-web crawling. *Proc. of the Int. Conf. on Web Information Systems Engineering (WISE)*, pages 538–551, 2012. → pages 13

[22] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, 2008. doi:http://doi.acm.org/10.1145/1454159.1454163. → pages 7, 12

[23] A. Mesbah and S. Mirshokraie. Automated analysis of CSS rules to support style maintenance. In *Proc. of the 34th ACM/IEEE Int. Conf. on Softw Eng. (ICSE)*, pages 408–418. IEEE Computer Society, 2012. → pages 17

[24] A. Mesbah, E. Bozdag, and A. van Deursen. Crawling Ajax by inferring user interface state changes. In *Proceedings of the International Conference on Web Engineering (ICWE)*, pages 122–134. IEEE Computer Society, 2008. → pages 13

[25] A. Mesbah, A. van Deursen, and S. Lenselink. 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. → pages 13, 18, 19, 20, 25

[26] A. Mesbah, A. van Deursen, and D. Roest. Invariant-based automatic testing of modern web applications. *IEEE Trans. on Softw Eng. (TSE)*, 38 (1):35–53, 2012. → pages 20

[27] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *Proc. of the ACM/IEEE-CS joint conference on Digital libraries*, pages 100–109. ACM, 2005. ISBN 1-58113-876-8. doi:http://doi.acm.org/10.1145/1065385.1065407. → pages 12

[28] B. Pinkerton. Finding what people want: Experiences with the web crawler. In *Proc. of the Int. World Wide Web Conf. (WWW)*, volume 94, pages 17–20, 1994. → pages 12

[29] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 129–138, 2001. ISBN 1-55860-804-4. → pages 1, 5, 7, 12

[30] W. Wu, A. Doan, C. Yu, and W. Meng. Modeling and extracting deep-web query interfaces. In *Advances in Information and Intelligent Systems*, volume 251, pages 65–90. Springer, 2009. → pages 7, 12

[31] C. Yue and H. Wang. Characterizing insecure JavaScript practices on the web. In *Proc. of the Int. World Wide Web Conf. (WWW)*, pages 961–970. ACM, 2009. ISBN 978-1-60558-487-4. doi:http://doi.acm.org/10.1145/1526709.1526838. → pages 1, 16