# The Reconstruction of User-Interactions from HTTP Traces for RIAs

by

Sara Baghbanzadeh

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

**Abstract**

A user using a Web application generates HTTP traffic which can be captured and logged to be used for further analysis such as finding potential security holes. This document provides a method to reconstruct user-interactions from HTTP network traffic with a specific focus on RIAs. The only input provided is the full, unaltered HTTP network traffic of the original user-session. This thesis presents a system to replay HTTP traffic for reconstructing the user-interactions using a programmable Web browser that is used to simulate user's actions and to execute client-side scripts.

The experimental results show that the proposed solution in this thesis leads to a system which has a good performance in reconstructing user-interactions. The system performs an automated reconstruction of the user-interactions which reconstructs every DOM state that has been visited by the user during the user-session and the actions taken by the user, including user-inputs provided by the user.

## Dedication

I dedicate this thesis to my parents, Kamal Baghbanzadeh and Eghbal Aryan. You have always supported me and never asked for anything in return. All I have and will accomplish are only possible owing to your love and sacrifices. From the deepest part of my heart, I love you both more than anything else in the world.

**Acknowledgements**

First, I would like to offer my sincerest gratitude to my supervisor, Professor Guy-Vincent Jourdan, for his patience, guidance and support throughout my graduate studies.

I would like to thank Professor Gregor von Bochmann for his priceless inputs and guidance. I also wish to thank Dr. Iosif Viorel Onut, Seyed M. Mirtaheri, Mehdi Seyed Salehi, Alireza Farid Amin and all my colleagues in the Software Security Research Group at the University of Ottawa.

I would like to give special thanks to Dr. Shahzad Khan whom I have learned a lot from as a professional. I would also like to thank Faye Hosseini for all of her support, as well as the extended support I received from all my colleagues at Gnowit Inc.

The most important thanks go to my family most notably my brother, Mohammadali Baghbanzadeh, without whom living and studying abroad would have been impossible for me. My dear sister, Sima Baghbanzadeh, for all her love and support. My parents, Kamal Baghbanzadeh and Eghbal Aryan, for all they have done for me over the years. Their encouragement and sacrifices have made possible everything I have accomplished to date.

Finally, I would like to acknowledge the financial support I received from the National Science and Engineering Research Council (NSERC), and the IBM Center for Advanced Studies (CAS).

# Table of Contents

# List of Tables

# List of Figures

# Nomenclature

## Acronyms

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| CSS | Cascading Style Sheets |
| DOM | Document Object Model |
| GUI | Graphical User Interface |
| HTML | Hyper Text Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IDE | Integrated Development Environment |
| IDS | Intrusion Detection System |
| IIS | Internet Information Services |
| IP | Internet Protocol address |
| JSON | JavaScript Object Notation |
| RIA | Rich Internet Application |
| SSL | Secure Socket Layer |
| TCP | Transmission Control Protocol |
| URL | Uniform Resource Identifier |
| XML | Extensible Markup Language |

# Chapter 1

# Introduction

In the area of networking, a session refers to shared HTTP messages exchanged between a Web browser and a Web server. A user-session is defined as starting when an HTTP request is sent to the Web server from a new IP address and ending when there is no HTTP request triggered from the IP address or the session times out [12]. User-sessions are used in a variety of studies such as Web mining [34].

A Web browser triggers an HTTP request to the Web server when a user interacts with a website. User interaction is considered as either executing an HTML element, such as clicking on or hovering the mouse over it, or submitting an input to the Web server.

Analyzing user-interactions is important, since it helps researchers to find frequent patterns, security holes and existing bugs in the Web application. User-interactions are used in various fields of study such as Human-Computer interaction [55].

Reconstructing user-interactions determines how a user has reached a certain point of the Web application within a user-session. User-interactions reconstruction specifies all HTML elements which have been executed along with inputs which have been submitted to the Web server by a user. Moreover, reconstructing user-interactions should indicate the order of actions taken by the user during the user-session. In other words, user-interactions reconstruction is the ability to reverse-engineer what the user did with the Web application during her session.

Traditionally, web applications contained a set of pages accessible through unique URLs. The Web server processed HTML pages before sending the corresponding HTTP response

to the Web browser. Consequently, Web browsers were only in charge of displaying the received content from the Web Server. Later, newly introduced Web technologies shifted the workload from the Web server to the user's Web browser. This architecture introduces a new generation of Web applications called Rich Internet Applications (RIAs).

In recent years, an increasing number of Web applications have been making use of newer technologies such as AJAX. These "Rich Internet Applications" execute scripts on the client-side, which can modify the client state in a fluid way. RIAs can also exchange messages asynchronously with the Web server to create responsive Web applications. This new generation of Web applications achieves a higher interactivity and reduces the server workload by shifting part of the computation away from the Web server to the user's Web browser.

RIAs make the reconstruction of the user-interactions more challenging. On the one hand, the majority of the messages exchanged between the client and the Web server contain only data, which is interpreted by the client-side scripts, to update the state of the Web browser. On the other hand, there are actions, executed on the client-side, for which the user's Web browser does not trigger HTTP requests toward the Web server.

## 1.1 Motivations

There might be situations in which the owner of a Web application wants to know what exactly a user did before reaching a specific state. A hacker who has found security holes in a Web application can attack users' data. In addition, it is conceivable that a user finds a critical bug in a Web application after performing a sequence of actions. In both cases, since the session has already happened the only available data is the server-generated logs of the preceding user-session. Therefore, the owner of the Web application must be able to identify the security holes, as well as the source of reported bugs in order to have a secure and usable Web application.

On the other hand, there are tools like crawlers or security scanners designed to be run against the Web application. Such tools need to be configured to be able to provide the correct inputs and perform a successful login. In this case also, server-generated logs provide essential data.

A constructed user-interactions provides the set of actions which either caused a user to face a problem or assisted automated tools to achieve their aims.

## 1.2 Problem Statement

Tools [39, 2] have been designed to store the elements on which the user clicks during the user-session, but they must be installed on the user's machine. Such a tool cannot determine the sequence of user-actions after the user-session has already passed. In other words these tools cannot report user-interactions when the only available data is unaltered HTTP network traffic.

Extracting information from HTTP network traffic is difficult in RIAs, because it contains mostly data which is interpreted by the user's Web browser. Moreover, the Web server's response is typically in an XML, JSON, etc., format, which does not provide much information for someone who wants to analyze the user-session. Figure 1.1 shows a part of an HTTP response in a RIA.

```
{"soapenv:Body":{"response":{"returnValue":{"value":{"ImplClassName":
"com.ibm.team.service.jts.internal.mailer.MailerService","properties":[{
"defValue":"false",...,"eQClassName":"http:\/\/schemas.xmlsoap.org"}}}}}}
```

Figure 1.1: The body of a typical HTTP response in RIA [3]

This document provides a solution for user-interactions reconstruction from previously captured HTTP network traffic. The goal of the proposed solution is to automatically search out a set of results that is the sequence of **user actions**, **user inputs**, the series of **HTML pages**, the series of the **cookies** set by the Web server and the **screen shots** of all the pages visited by the user during the given user-session.

The problem statement of this research is summarized as the search for a method which finds the sequence of user-interactions that created the given user-session. The only input given to our tool is previously captured HTTP network traffic from which we are going to present a visual representation of the preceding user-session.

## 1.3 Overview of the Solution

As it will be discussed in chapter 4, the proposed solution to reconstruct user-interactions is to replay the previously captured HTTP network traffic in order to redo all the actions done by the user. We have developed a system with a programmable Web browser and a Proxy server. The browser communicates with the proxy server in order to replay the previously captured HTTP network traffic. The proxy server has access to the captured user-session so that it can use the HTTP messages to respond to the requests sent by the browser.

The first idea behind our solutions is to use the information stored in the HTTP network traffic. As it was mentioned, the proxy server is given an HTTP trace which contains all the information from the given user-session. Consequently, as it will be discussed in chapter 5, we define different "Control Messages" with which the browser and the proxy server communicate so that the entire system can benefit from the given data.

The second idea is to modify the HTTP response sent back to the browser so that the entire system can have more control of the Web application while replaying the HTTP network traffic. This process can be done either as a preprocessing step or during the reconstruction.

## 1.4 List of Contributions

This document makes the following contributions:

- This thesis shows how to handle user inputs while replaying previously captured HTTP network traffic. Each system which has been designed to reconstruct user-interactions from unaltered HTTP network traffic has to handle user inputs.
- This thesis shows how the process of reconstructing user-interactions can be done efficiently.
- This documents reports the results of having implemented all the suggested solutions for reconstructing user-interactions efficiently.

## 1.5 Organization of the Thesis

In chapter 2 we provide the definitions of basic terms and concepts which the reader of this thesis needs to know. Then, we review the existing approaches in the field of user-interactions reconstruction, as well as discussing the areas in which the result of our work can be used in chapter 3.

Chapter 4 provides a precise explanation of the existing framework for a user-interactions reconstruction system. In this chapter we also identify the challenges which the proposed framework faces. Then, we explain our solutions to deal with each of these challenges in chapter 5. The effectiveness of our proposed solutions is discussed in chapter 6. Finally, we conclude our work and investigate the drawbacks of our solutions in chapter 7.

# Chapter 2

# Basic Definitions

In this chapter we provide the definitions of basic concepts which are used in this document. The goal of this chapter is to describe the necessary terms of the field of Web analytics for those who are not familiar with the topic. First, we describe how a Web browser and a Web server communicate so that the user's Web browser can render a Web page properly in section 2.1. Second, we define a few terms which are used within this document in section 2.2. Then, we give an introduction to user inputs in section 2.3 and finally, section 2.4 summarizes this chapter.

## 2.1 Communication in the World Wide Web

A Web browser is a client service through which a user can get access to the content available in the World Wide Web. A Web browser is a program that exchanges HTTP messages with a Web server. It interprets the received HTTP response from a Web server in order to display a Web page. Nowadays, a Web browser is a complex piece of software capable of rendering dynamic Web applications [19]. It provides variety of functionalities such as parsing CSS files and executing JavaScript functions in order to render dynamic Web applications.

A Web server is software which runs on a server computer that hosts websites [19]. A Web server receives the HTTP requests of the user's Web browser and provides it with an HTTP response in the requested format. In general, "a Web server is a bridge between

the network and the documents" [70].

The user's Web browser communicates with the Web server which hosts the target web application through the HTTP protocol. The user's Web browser sends an HTTP request to the Web server in order to obtain a resource. The requested resource can be a pure HTML document, CSS or XML file, JavaScript function, an image etc. The Web server returns the requested resource to the user's Web browser through an HTTP response.

HTTPS, secure HTTP, is used for securely exchanging data between a Web browser and a Web server. It works in the way of establishing Secure Socket Layer connection, known as SSL, and transmitting all HTTP messages over the established secure connection.

The HTTP protocol is a stateless protocol which means that all HTTP requests are considered independent of each other. Maintaining user-session requires Web server to identify the user and store information about the user [59]. There are Web applications to which a user logs in to get access to her own profile. Therefore, there should be a mechanism by which a user can be tracked by the Web server.

Cookies, which are small bits of information, are generated by the Web server to "refer to the state information that passes between an origin server and user agent, and that gets stored by the user agent"[1]. The user's Web browser sends the stored cookie back to the Web server in all its succeeding HTTP requests [59]. This way the Web server can verify the identity of the user's Web browser in all its following HTTP requests.

The user's Web browser triggers the first HTTP request, which is the initial URL indicated by a user, to the Web server. Figure 2.1 shows the different parts of a URL.

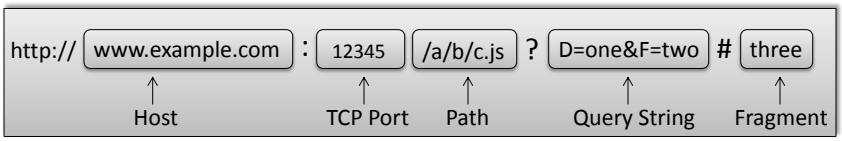


Figure 2.1: HTTP URL [27]

Each part of the URL indicates a goal. The URL may contain some or all of the specified parts in the figure 2.1. The goal of each part of the URL is as follows [25]:

[1] `https://www.ietf.org/rfc/rfc2109.txt`

- Routers use the *Host* and the *TCP Port* specified in a URL to determine where to send the received HTTP message. These parts indicate where the target Web server is located.
- The Web server determines where to look for the requested resource using the specified *Path*. The resource is sent to the Web browser from the specified *Path* by the Web server.
- *The Query String* carries all the parameters and possibly the user's inputs which are sent to the Web server. The Web server's response depends on the values sent in the *Query String*.
- The *Fragment* will not be sent to the Web server. It is used by the user's Web browser to scroll the screen to provide the user with a proper view of what she looks for.

The Web browser parses the HTML document upon receiving the Web server's response. It generates an automated HTTP request which is sent to the Web server once it extracts a resource such as JavaScript function, image etc. which should be fetched for rendering the Web page. The Web page appears for the user once the Web browser receives all the resources.

The process of communicating between a Web browser and a Web server takes up bandwidth. The user's Web browser requests various resources depending on how large the target Web application is. Moreover, the time for downloading high quality images is much longer than for downloading plain text.

Web browsers can store static resources on the hard drive of the users' machine. This capability is called the Web browser's caching. Due to the cache, the number of HTTP messages exchanged between the Web browser and the Web server decreases because the Web browser uses the cached version of some resources instead of fetching them from the Web server.

Each HTTP message has to follow a standard structure that is a message containing "HTTP Headers" followed by a potential "body". "HTTP Headers" determine how a Web browser and a Web server should react to the received HTTP message. Figure 2.2 shows an HTTP request triggered by a Web browser.

```
POST "http://demo.opencart.com/index.php" HTTP/1.1
Host: "demo.opencart.com"
User-Agent: "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/44.0.2403.155 Safari/537.36"
Referer: "http://demo.opencart.com/index.php?route=account/register"
Connection: "keep-alive"
Accept: "*/*"
Accept-Encoding: "gzip, deflate, sdch"
Accept-Language: "en-US,en;q=0.8"
Content-length: "54"
Cookie: "PHPSESSID=4dcdefa9dc17431319a2f75bff20ae8e4dcdefa9dc17431319a2f75bff20ae8e;
    language=en; currency=USD"

route=account/register/customfield&customer_group_id=1
```

Figure 2.2: HTTP request generated by a Web browser

The first line of each HTTP request contains three parts, the method used by the Web browser to send the request, the URL to fetch the resource from the Web server and the version of the HTTP protocol based on which the Web browser wants to communicate with the Web server.

The common HTTP methods used by the user's Web browser are "GET" and "POST". The former one is used to get a resource from a remote Web server and the latter one is used to post data to an object located on the Web server [21]. Although, they both can carry a *Query String* and the "body" of the request, the "GET" method usually does not pass on the data inside the "body" of the message.

The rest of the lines which are used by the Web server are called "HTTP Headers". There are various "HTTP Headers" which might be mandatory. "User-Agent", which shows the type and the version of the user's Web browser, is a mandatory "HTTP Header". "Content-length" determines the length of the string which is being sent to the Web server as the "body" of the HTTP request. "Content-length" is also a mandatory "HTTP Header" if the message carries a "body". The reason is that the Web browser should determine the number of bits it needs to read from the message. "Accept-Encoding", "Accept-Language", "Referer", "Host" etc., are other optional "HTTP Headers" used by the Web browser.

The HTTP response sent by the Web server should also follow a standard. Figure 2.3 shows the response to the above HTTP request sent by the Web server.

```
HTTP/1.1 200 OK
Date: Fri, 21 Aug 2015 02:32:30 GMT
Server: Apache
X-Powered-By: PHP/5.3.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Keep-Alive: timeout=3, max=29997
Connection: Keep-Alive
Content-Type: application/json
Content-Length: 2

[]
```

Figure 2.3: HTTP response generated by a Web server

The version of the HTTP protocol used by the Web server, the HTTP response code and a brief description about the mentioned code should be specified in the first line of each HTTP response.

Each HTTP response generated by a Web server carries different "HTTP Headers". "Content-Length" determines the length of the response body. "Content-Type" specifies the type of the data which is being sent to the user's Web browser. "Cache-Control", "Pragma", "Last-Modified", "Etag", "Vary" and "Expires" are the "HTTP Headers" used by the Web server to control the cache behavior of the Web browser [21]. "Set-Cookie", "Date" etc. are other "HTTP Headers" set by the Web server.

## 2.2 Terms and Concepts

A Web browser builds a Document Object Model (DOM) of an HTML document while the Web page is loading. The DOM "is a platform and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents"[2]. It defines the way in which a document is accessed and modified. Scripting languages such as JavaScript can dynamically access and modify an HTML document through its corresponding DOM. All HTML elements along with their attributes are accessible through the DOM built by a Web browser. The content, structure and style

[2] `http://www.w3.org/DOM/`

of a document may be changed by JavaScript through its equivalent DOM and the results of those changes can be integrated back into the document [15, 28].

JavaScript is a high level single-threaded scripting language which is used to make dynamic Web applications. It is the most common scripting language used for developing RIAs. RIAs rely on JavaScript to provide the user with dynamic functionalities [15].

An HTML event refers to anything which happens to an HTML element through either a Web browser or a user[3]. Clicking on an element, hovering a mouse over it, changing the value of an input field by a user, or finishing loading a Web page by a Web browser are examples of HTML events. JavaScript functions can be called upon triggering an event since HTML provides event handler attributes which are added to an HTML element.

An Event handler is a JavaScript function designed for processing a specific type of event. An event handler stays in the active state once it is created until it is destroyed [45]. Figure 2.4 shows an event handler attached to an HTML element in a RIA called "PhotoStack"[4].

The purpose of the JavaScript code, shown in figure 2.4, is to attach an event handler to all the  elements which are the child of an element with the identifier "ps_albums" within the Web application. The event handler will be triggered upon clicking on each of the  elements. The function has been designed to load the list of pictures for each album through an AJAX (Asynchronous JavaScript and XML) call. Then it randomly rotates all pictures except the last one which is the one the user sees first. The handler also resizes and centers each picture.

AJAX refers to a technology by which a Web browser exchanges data asynchronously, in the background, with the Web server. Consequently, the current Web page of the Web application is updated without having to be reloaded. JavaScript functions which have been attached to HTML elements also may cause a Web browser to make an AJAX call to a Web server.

Ajax applications use dynamic HTML, cascading stylesheets and JavaScript to create more dynamic and animated Web pages compared to traditional Web applications. Ajax uses XML (Extensible Markup Language) to encode data being transferred between the

[3] `http://www.w3schools.com/js/js_events.asp`

[4] `http://tympanus.net/Tutorials/PhotoStack/`

```
$ps_albums.children('div').bind('click', function() {
  var $elem = $(this), album_name = 'album' + parseInt($elem.index() + 1);
  var $loading = $('<div />', { className: 'loading' });
  $elem.append($loading);
  $ps_container.find('img').remove();
  $.get('photostack.php', { album_name: album_name }, function(data) {
    var items_count = data.length;
    for (var i = 0; i < items_count; ++i) {
      var item_source = data[i], cnt = 0;
      $('<img />').load(function() {
        var $image = $(this); ++cnt;
        resizeCenterImage($image);
        $ps_container.append($image);
        var r = Math.floor(Math.random() * 41) - 20;
        if (cnt < items_count)
          $image.css({'-moz-transform': 'rotate(' + r + 'deg)',
          '-webkit-transform': 'rotate(' + r + 'deg)',
          'transform': 'rotate(' + r + 'deg)'});
        if (cnt == items_count) {
          $loading.remove(); $ps_container.show();
          $ps_close.show(); $ps_overlay.show();
        }
      }).attr('src', item_source);
    }
  }, 'json');
});
```

Figure 2.4: Attaching an Event Handler to an HTML Element

user's Web browser and the Web server. It uses asynchronous JavaScript which provides an application with the ability of making asynchronous calls to the Web server, retrieving new data and updating the Web page simultaneously while the user can continue interacting with the application. The important feature of AJAX-based Web applications is that they do not require any plug-ins because JavaScript is a cross-platform scripting language [49].

Extensible Markup Language "is a simple, very flexible text format originally designed to meet the challenges of large-scale electronic publishing. XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere"[5]. A set of XML-based languages with their own set of markup tags can be defined by XML to be used with structured data [49].

[5] http://www.w3.org/XML/

As a result, JavaScript can be used to create dynamic Web applications. Event handlers can be attached to the HTML elements inside the DOM to generate responsive Web applications. The DOM can be modified through an AJAX call to the Web server without interrupting the user.

The thread of a Web browser which has been used to render a Web page is stopped when an event handler is invoked in order to execute the corresponding JavaScript function. Therefore, the Web application seems non-responsive especially if the handler takes a considerable amount of time to be executed [47].

Figure 2.5 shows the traditional model for Web applications compared to the AJAX-based ones. AJAX-based Web applications create a JavaScript-based engine which is run on the Web browser. Consequently, the Web browser loads the engine which intercepts user-inputs, as well as handling interactions such as validating data on the client-side. The engine requests more data from the Web server in the background if it needs to fetch additional data. Therefore, the engine gives the user the ability of interacting with the Web application independently of server communication [49].

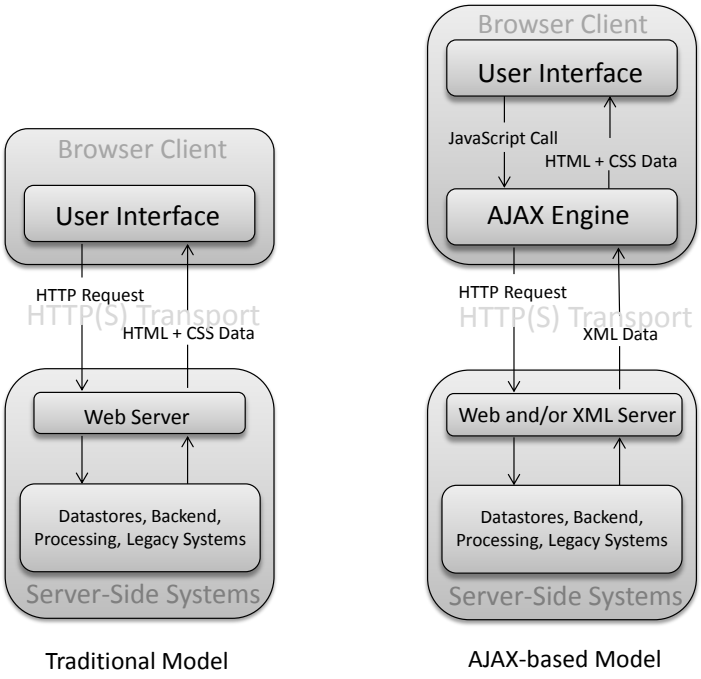


Figure 2.5: Comparison of traditional and AJAX-based model of Web application [17]

HTTP network traffic exchanged between a Web browser and a Web server can be logged in two different ways. The entire log is called HTTP trace.

- **The Web Server Log** refers to the log file created by a Web server automatically. The log may contain request URL, request time, user's IP address etc. depending on server settings. Web server logs are not available to anyone but the owner of the Web server. Figure 2.6 shows a part of a log file captured by a local IIS Web server.

```
#Software: Microsoft Internet Information Services 8.5
#Version: 1.0
#Date: 2015-09-20 03:52:36
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(
    User-Agent) cs(Referer) sc-status sc-substatus sc-win32-status time-taken
2015-09-20 03:52:36 ::1 GET /AltoroMutual - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64)
    +AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/45.0.2454.93+Safari/537.36 - 301 0 0
    10593
2015-09-20 03:52:39 ::1 GET /AltoroMutual/ - 80 - ::1 Mozilla/5.0+(Windows+NT+6.3;+WOW64
    )+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/45.0.2454.93+Safari/537.36 - 200 0 0
     2529
2015-09-20 03:52:39 ::1 GET /AltoroMutual/style.css - 80 - ::1 Mozilla/5.0+(Windows+NT
    +6.3;+WOW64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/45.0.2454.93+Safari
    /537.36 "http://localhost/AltoroMutual/" 200 0 0 61
2015-09-20 03:52:39 ::1 GET /AltoroMutual/ajaxer.js - 80 - ::1 Mozilla/5.0+(Windows+NT
    +6.3;+WOW64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/45.0.2454.93+Safari
    /537.36 "http://localhost/AltoroMutual/" 200 0 0 95
```

Figure 2.6: Web Server Log

- **The Web Proxy Log** refers to the log file captured by a Web proxy. The information provided in the log file depends on the proxy setting. Web proxies will be discussed in section 4.4 of chapter 4 in more detail.

## 2.3 User Inputs

User inputs are the fundamental part of Web applications since the data behind them is not directly accessible through hyper-links but they are retrieved when a user submits a query to the Web server via user inputs [57].

Users provide inputs through HTML fields. Fields should have a unique name defined in an attribute called "name". Otherwise, this part of the data cannot be uniquely stored in the target destination. Consequently, HTTP specifies the standard based on which the user inputs are submitted to the Web server [3]. The norm is that each query contains the combined <name, value> pair of the user inputs.

Each HTML field can be defined as one of the following HTML elements within the Web application[6].

- **Input** which defines an area in which the user can enter data. This HTML element has a mandatory attribute called "Type" which takes one of the predefined values. Bellow we list some of its common values used in Web applications.

  - **text**, allows the user to enter characters in a single line. The length of the query can be set so the user is limited to enter a maximum of a predefined number of characters.

  - **radio**, creates a button which might be selected by the user. Usually there will be more than one button so that the user selects one among all available ones.

  - **checkbox**, creates a box which might be selected by the user. The user can select one or more boxes at the same time. From IR point of view, the difference between "radio" and "checkbox" is in their "name" attribute.

    "radio" buttons are a single item of data which take one of several values. To have all the buttons as a part of the same set, the value of the "name" attribute corresponding to all these inputs should be the same. In contrast to "radio" buttons "checkboxes" belong to separate items of data and then each "checkbox" has to have a unique "name".

    "checked" attribute also determines which "radio" button or "checkbox" has been selected by the user.

  - **submit**, specifies that its corresponding HTML element is intended to submit the HTML form that contains that element.

[6] `http://www.w3.org/wiki/HTML_forms_-_the_basics`

  - **hidden**, corresponds to a value that is not going to be observed or modified by the user. All the hidden values, which have been set by the Web server, will be sent back to the Web server once the user executes the corresponding HTML element.

- **Textarea**, creates a multiple-line text input area in which the user can enter characters. The number of lines can be set so that the user cannot just enter as many characters as she desires.

- **Select and Options**, provide a multi-line drop-down menu from which the user can select only one value. The values have been defined inside the "option" elements. The selected value will be sent to the Web server in the form of <name, value> pair once the user submits the HTML field.

An HTML form is a collection of HTML fields which are submitted simultaneously to the Web server. Figure 2.7 shows the DOM representation of an HTML form in which different types of field element exist.

```
<form id="sample-form" method="post">
  <input type="text" name="name" id="name" value="" />
  <h1>Please check all the emotions that apply to you:</h1>
  <input type="checkbox" name="angry" id="angry" value="angry">
  <input type="checkbox" name="sad" id="sad" value="sad">
  <input type="checkbox" name="happy" id="happy" value="happy">
  <h1>How satisfied were you with our service?</h1>
  <input type="radio" name="satisfaction" id="vsat" value="vsat">
  <input type="radio" name="satisfaction" id="sat" value="sat">
  <h1>Further comments</h1>
  <textarea name="comments" id="comments" cols="25" rows="3"></textarea>
  <h1>Location visited</h1>
  <select name="location" id="location">
    <option value="">Select location</option>
    <option value="vancouver">Vancouver</option>
  </select>
  <input type="submit" value="submit" />
</form>
```

Figure 2.7: the DOM Representation of an HTML Form

Each HTML field can be sent to the Web server via either clicking on a submit button or by executing a JavaScript function which has been attached to the field element upon triggering its active event. The potential active event for an HTML element depends on the "type" of the input. For instance, the event which can be activated for a "text" field may not be suitable to be activated for a field with the type "select". One possible event for the former one is called "keypress". "onchange" is a potential event which may be activated for an HTML field with type "select".

HTTP defines a standard based on which URLs have to be encoded uniformly. In other words, all characters should be converted into a format transmittable over the Internet[7].

The method by which an HTML form is submitted to the Web server is determined via the "method" attribute of the form element. All the inputs have to be encoded while transmitting over the Internet if they are being sent through the query string of the URL. Otherwise, having the inputs encoded is determined by one of the attributes of the HTML form element.

"enctype" is an attribute of the form element by which the format of submitting the HTML form is specified. There are two possible values which "enctype" can take: "application/x-www-form-urlencoded" and "multipart/form-data". In the first mentioned format, as shown in figure 2.8, the values are encoded before being sent to the Web server. The "application/x-www-form-urlencoded" format is the default format of submitting inputs through form element. In HTTP request specified in figure 2.8, the actual value for "password" is "Ssrg!234" which has been encoded to "Ssrg%21234".

```
POST "https://ssrg-uottawa.demojoomla.com/administrator/index.php" HTTP/1.1
Host: ssrg-uottawa.demojoomla.com
Content-Length: 122
Content-Type: application/x-www-form-urlencoded
Cookie: 57c02f8fabba8d222130709009846047=1nqcg1ar9rcv6dc8jcf14rda75

username=ssrg-uottawa&passwd=Ssrg%21234&option=com_login&task=login&back=aW5kZXgucGhw&
    dde72b9436dfdb1d0d999b4500f64c65=1
```

Figure 2.8: the HTTP request of a submission via default format

[7] http://www.w3schools.com/tags/ref_urlencode.asp

"The content type "application/x-www-form-urlencoded" is inefficient for sending large quantities of binary data or text containing non-ASCII characters. The content type "multipart/form-data" should be used for submitting forms that contain files, non-ASCII data, and binary data"[8]. None of the entered values will be encoded in this format. As shown in figure 2.9 there should be a delimiter, created on the client-side, used to separate inputs. The value of the delimiter is sent to the Web server via the "boundary" HTTP header.

```
POST "http://demo.opencart.com/index.php?route=account/register" HTTP/1.1
Host: demo.opencart.com
Connection: keep-alive
Content-Length: 478
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryk4UpajDDRghnxHFk
Referer: "http://demo.opencart.com/index.php?route=account/register"
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8
Cookie: PHPSESSID=80b4fcd9563216f3383c45473cf194e0; language=en; currency=USD

------WebKitFormBoundaryk4UpajDDRghnxHFk
Content-Disposition: form-data; name="customer_group_id"

1
------WebKitFormBoundaryk4UpajDDRghnxHFk
Content-Disposition: form-data; name="email"

test_ssrg@uottawa.ca
------WebKitFormBoundaryk4UpajDDRghnxHFk
Content-Disposition: form-data; name="password"

Ssrg!234
------WebKitFormBoundaryk4UpajDDRghnxHFk
Content-Disposition: form-data; name="confirm"

Ssrg!234
------WebKitFormBoundaryk4UpajDDRghnxHFk
```

Figure 2.9: the HTTP request of a submission via "multipart/form-data" format

[8] http://www.w3.org/TR/html401/interact/forms.html

## 2.4 Summary

In this chapter we defined some necessary terms and concepts which are used in this document. The way in which a Web browser and a Web server communicate has been explained. This chapter also specified the standard way of defining user inputs in Web applications along with the possible formats based on which the values entered by a user will be transmitted to the Web server.

The next chapter will review the existing works which have been done in the field of user-interactions reconstruction.

# Chapter 3

# The State of the Art

In this chapter we provide a review about existing works in the field of user-interactions reconstruction. First, section 3.1 presents a precise description of what the user-interactions reconstruction is along with introducing steps which make possible the process of reconstructing user-interactions from the given HTTP network traffic. Second, section 3.2 shows fields of study in which the result of this research can be used. Finally, section 3.3 summarizes this chapter.

## 3.1 User-interactions Reconstruction

User-interactions reconstruction refers to the ability of identifying what a user did with a Web application during a user-session. The process of reconstructing user-interactions is helpful in different areas which will be discussed in section 3.2. In addition to reconstructing the entirety of user-interactions, there are also tools designed to store and report the final state of an application from which the user has encountered problems, bugs, etc., so developers can fix the reported issues[1] [1, 18].

Having the final state of an application in case of error occurrence is not always useful, since there might be missing steps necessary to reproducing the reported issue [24]. Consequently, being able to take advantage of all steps will help developers to save time and effort while finding the source of the reported problem. Therefore, a need for having

[1] `https://support.mozilla.org/en-US/kb/mozillacrashreporter`

a user-interactions reconstruction system comes up in order to determine all the user's actions in the same order that they have happened.

The sequence of all of a user's actions can be stored while a user interacts with the website. Vikram et al. [67] introduce Ripley which has been designed for detecting malicious activities on the client-side. They suggest replicating the client-side executions in a trusted Web server. Consequently, one of the key steps is to capture and send each user's action to the replica so the action can be executed as a replica of the client. All detected inconsistencies between the execution results are considered as malicious. Therefore, Ripley can be used to capture the sequence of user-interactions with the Web server on the client-side.

Mugshot [39], which has been implemented in JavaScript, is another tool by which actions are recorded into an in-memory log. Consequently, the final log contains a sequence number and clock time for each recorded action. Andrica et al. [2] introduces WaRR which is a record-replay system embedded in the user's Web browser. It logs the sequence of a user's actions including the type of action, an identifier of the executed HTML element and the elapsed time since the last recorded action. DoDOM, Jalangi and CARENA [48, 58, 46] have also been designed to capture the sequence of user-interactions while the user interacts with the Web application.

One limitation for a user-interactions reconstruction tool which is designed using the above-mentioned approaches is that all of these tools have to be installed on the user's machine to be able to capture all the actions. Therefore, if the user-session has already passed, such a tool will not be able to reconstruct the user-interactions.

HTTP network traffic, exchanged between user's Web browser and a Web server, is the only data available when a user-session terminates. Consequently, a user-interactions reconstruction tool should be able to determine what the user did during the user-session from a given HTTP trace.

As discussed in chapter 2 both server logs and Web proxy logs will keep different user-sessions in one file. In general, users browse a website hosted by a Web server concurrently. the Web server or a Web proxy stores the messages that come from all the users in a log file. Therefore, the HTTP trace is a mix of HTTP messages belonging to different users.

The first important challenge which should be considered is how to separate existing

user-sessions since a user-interactions reconstruction tool needs to be provided with one user-session. Otherwise it fails to reconstruct the user-interactions in the same order that they happened. There are different approaches introduced in order to reconstruct user-sessions efficiently.

As Dohare et al. [53] suggest, there are two main strategies called "reactive" and "proactive" for user-session reconstruction. The former one refers to approaches which analyze HTTP traces after a user stops communicating with a Web server. The latter one refers to approaches which start processing the data while a user browses a website.

Each user-session contains all of a user's HTTP requests which come from the same machine and the same Web browser. The user's IP address, URL address and access time are factors usually used to separate user-sessions [63]. Moreover, different heuristics also consider other fields in order to construct user-sessions more precisely. The reactive strategies are divided into two categories, the time oriented heuristics [62, 7] and referrer based heuristics [4, 7, 43].

The first category considers the time of visiting each Web page by a user. Cooley et al. [7] has defined a condition based on which they determine the user-session to which each HTTP message belongs. They assume that the entire session should not exceed a maximum time. Using this method all the pages, start from $P_1$ and going to $P_n$, are considered as a user-session if the entire duration does not exceed the predefined threshold T. WUM [62] adds a new constraint to the above-mentioned heuristic for putting together the HTTP messages for reconstructing the user-session. Based on the new condition two sequential HTTP messages belong to a user-session if the elapsed time between those two HTTP messages does not exceed a predefined threshold.

The second category finds a user-session by considering the fact that each of the Web pages in a user-session should be accessible from the previous one. Consequently, the "referrer" HTTP header is used to put together the HTTP messages in a log file. This heuristic is not always practical, since the "referrer" HTTP header is not a mandatory header. In fact, there are situations in which an HTTP request does not carry a referrer header because of being triggered by executing a JavaScript function [69]. This approach was first introduced in [7] while [4] and [43] have improved the approach.

The following subsection introduces existing approaches for a user-interactions recon-

struction tool and specifies the limitations based on which the existing approaches fail to reconstruct user-interactions for RIAs.

### 3.1.1 Current Approaches

Reconstructing user-interactions from an HTTP trace is a newly introduced research topic. There are only two implemented tools [69, 44] used in traditional Web applications in which two Web pages are connected by a unique URL. Our research introduces the first approach to reconstruct user-interactions from HTTP network traffic in RIAs.

Xie et al. [69] propose Resurf which is used to reconstruct user-interactions from HTTP network traffic by generating a "Referral Graph". They suggest grouping HTTP requests to determine what the behavior of the user was. In this approach a "Head Request" refers to the HTTP request generated by a user through either clicking on a link or typing a new URL in the address bar of the Web browser. Resurf takes three conditions into account in order to label an HTTP request as a "Head Request".

- The response to a "Head Request" should be an HTML/XML file from a Web server.
- The size of the HTTP response to a candidate "Head Request" should not be less than a predefined threshold S bit since, based on the first mentioned condition, the HTTP response carries information for loading a new Web page.
- The elapsed time between each "Head Request" and the last previously grouped HTTP request has to be more than a predefined threshold T.

Next phase, after determining "Head Requests", is to identify what the "Embedded Requests" are. "Embedded Requests" refer to the HTTP requests generated by the Web browser automatically in order to render the requested Web page properly. Resurf suggests utilizing the referrer header and the timing information as two factors for grouping HTTP requests.

First, Resurf generates the referrer graph G in which each HTTP request is a node. Node $V_1$ is connected to node $V_2$ if the requested URL of $V_1$ is the value of the referrer header in $V_2$. Consequently, Resurf traverses the referrer graph G backward until it reaches

one of the labeled "Head Requests". Secondly, it will use the elapsed time between an "Embedded Request" and a "Head Request" if an "Embedded Request" matches two "Head Requests".

As a result, each path of the referrer graph shows the sequence of triggered HTTP requests belong to one user's actions. Consequently, Resurf groups HTTP requests for each action without specifying the HTML element which has been executed in each step.

Resurf also uses the generated group of requests to extract and analyze statistics from previously captured HTTP network traffic. Therefore, in this approach the user-interactions sequence is reconstructed by grouping HTTP requests with each group representing one user's interaction.

ClickMiner [44] which is the closest approach to ours uses the created referrer graph by Resurf in order to prune the given HTTP trace for making easier the process of reconstructing user-interactions.

Neasbitt et al. [44] suggest replaying HTTP network traffic through an instrumented Web browser in order to reconstruct user-interactions. They extract a referrer subgraph G' from G in which the nodes belong to a direct user-interaction. In other words, ClickMiner first drops "Embedded Requests" to make G' as a graph of user-interactions. Then, it starts replaying the network traffic with regard to G' which shows the list of user-interactions. Therefore, they prune graph G as follows,

- **Pruning by Referrer Delay:** All the "Embedded Requests" will be pruned during this phase. ClickMiner uses the elapsed time T between each pair of HTTP requests since they believe T is longer when a user executes an element inside the Web page.
- **Pruning Asynchronous Request:** The AJAX calls toward the Web server will be pruned at this step. The requested URL of an AJAX call does not exist in the DOM. Each AJAX call is triggered as a result of executing JavaScript functions. Consequently, ClickMiner is neither able to find the corresponding HTML element for triggering such a call nor to render the corresponding state by reloading the requested URL.

  Therefore, ClickMiner uses the "X-Requested-With" HTTP header in order to identify AJAX calls and since this header might not exist in the HTTP request there

would be HTTP requests which are labeled "Head Requests" mistakenly.

- **Pruning Ads and Social Widgets**: All HTTP requests for fetching advertisements or social widgets will be pruned. ClickMiner has access to the list of all host names which host advertisements through AdBlock Plus[2]. Therefore, it prunes the referrer graph by dropping HTTP requests in which the value of the referrer header or the requested URL is equal to one of the host names suggested by AdBlock Plus.

ClickMiner starts reconstructing user-interactions based on extracted referrer graph G' by replaying previously captured HTTP network traffic through its instrumented Web browser. Figure 3.1 shows the way in which ClickMiner works.

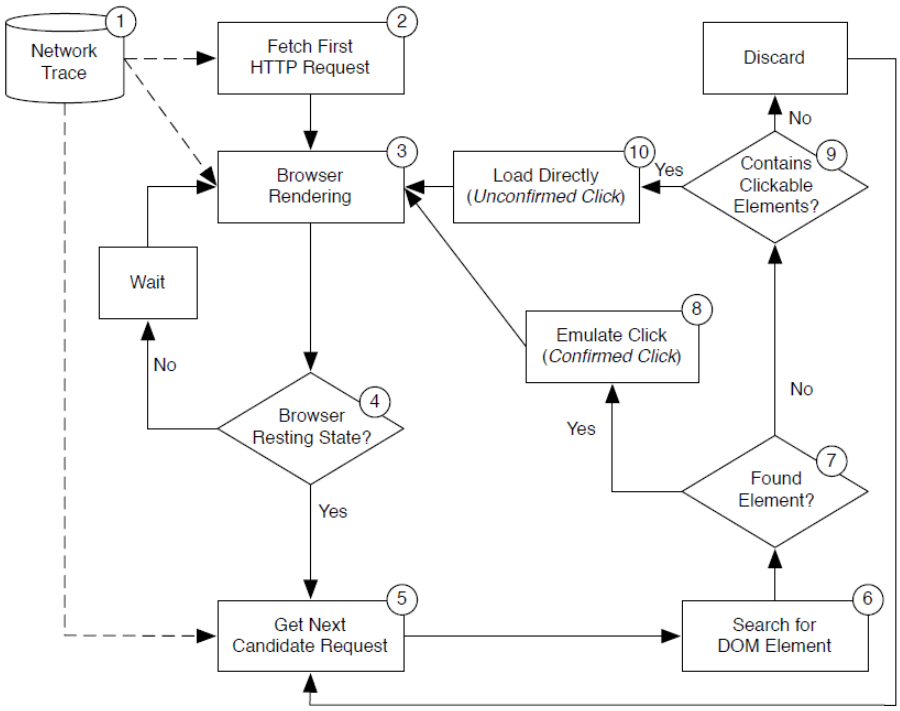


Figure 3.1: ClickMiner's Process (Simplified) [44]

An important challenge which each user-interactions reconstruction tool faces is to retrieve user inputs from a given user-session. Otherwise, the tool misses some of the user's actions to be reconstructed. The next subsection summarizes the existing approaches for retrieving user inputs in the field of Web analytics.

[2] `https://adblockplus.org/`

### 3.1.2 Data and User-Inputs Retrieval

User input plays a significant role in today's Web applications. Users get access to the high volume of information stored in the World Wide Web through search engines. Consequently, a user has to type a query in a corresponding field to reach the data. Although the existence of forms is welcome from the user's point of view, automated tools will encounter problem finding the data behind fields [36].

Finding the possible suitable data to be entered as user input causes the automated tool to potentially work inefficiently, since the tool needs to try different possible user inputs for submitting a form [36]. There are lots of studies for retrieving data from Web applications [10, 54, 36, 31]. All approaches try to make automated tools efficient while working with Web applications that contain user inputs.

Doorenbos et al. [10] propose ShopBot to extract vendor's content using domain-specific heuristics. ShopBot extracts patterns from the source of the HTML document by learning the structure of shopping websites. Consequently, ShopBot retrieves the information about existing products based on the general knowledge it has learned. The focus of ShopBot is on shopping domains which causes the approach to be considered not as a general methodology. The approach finds out where to put user inputs but it does not provide information about what to enter as a user input.

Most of the data available in the Web have been stored in the database of the Web server and are called the "Hidden Web". The "Hidden Web" can be retrieved either by filling out Web forms or sending suitable queries to the Web server [60]. Raghavan et al. [54] suggest a way by which Web crawlers can fill out Web forms automatically to go beyond the "Hidden Web".

HiWE [54] introduces a method to parse an HTML DOM for determining the location of each DOM element, as well as extracting the outer text of the closest element to each input field. HiWE learns and updates its database of task descriptions, which is initially created with a user-provided description, while crawling a website. The proposed method assumes that the crawl is domain-specific. This approach tries predefined values and updates its dictionary once it finds suitable data for a specific type of field element. Consequently, the approach does not work if the exact value of the user input is needed.

Liddle et al. [36] suggest a domain-independent method which takes along a user into the process to retrieve the "Hidden Web" behind a Web form. The user provides the tool with some queries, as well as determining a threshold based on which tool works.

The proposed approach contains two phases. First, the HTML page containing the target form is parsed and the tool initiates an HTTP request to the Web server. The request is equivalent to when the form is filled out manually. Consequently, the tool starts analyzing the received HTTP response. All new HTTP links will be followed in order to retrieve all the "Hidden Web" data behind the target form. Secondly, the tool tries all possible queries until either one of the predefined threshold is reached or it finds that there is no new data to be retrieved.

The second above-mentioned phase of retrieving data is called the exhaustive phase. It can be done more efficiently by using predefined thresholds such as percentage of data retrieved, number of queries issued, number of bytes retrieved, amount of time spent and number of consecutive empty queries. Therefore, the tool stops trying more queries once one of the predefined thresholds is reached.

Liddle et al. [36] also suggest that retrieving the "Hidden Web" would be more efficient and reliable when relevant data is extracted, to be used as a query, from the target website. Although this makes the process of retrieving the "Hidden Web" task-specific, this makes the process more efficient as well.

The issue with retrieving user input is more challenging while reconstructing user-interactions since each user input is an action which has to be identified. Consequently, a user-interactions reconstruction tool should be able to find the exact value of the input which the user had used during a given user-session.

At the time of writing this document there is no user-interactions reconstruction tool which reports all previously submitted user inputs from given HTTP network traffic. This document proposes a way by which all user inputs are retrieved while reconstructing user-interactions from a given user-session.

### 3.1.3 Summary

In this section we gave an introduction to a user-interactions reconstruction tool. We introduced two types of tools for reconstructing user-interactions within a Web application.

- Tools which have to be installed on a user's machine to be able to store all the user-interactions while a user browses a website.
- Tools which are used to reconstruct user-interactions when a user-session has already passed. Therefore, the only input will be captured HTTP network traffic.

We then discussed existing approaches for reconstructing user-interactions, as well as methods proposed for retrieving user inputs in Web applications.

## 3.2 Related Works

This section lists the fields in which a user-interactions reconstruction system can play a significant role. Also, a brief explanation about existing approaches in each field will be given.

- **Web Usage Mining** refers to the process of understanding the behavior of a user in a Web application by extracting and analyzing log files usually stored by the Web servers. Consequently, the behavior of a user can be predicted. There is a lot of research done in the area of Web usage mining [68, 13, 64, 53]. Wu et al. [68] introduces SpeedTracer which is used to create groups of user-interactions paths from the server logs by applying data mining techniques. Etminani et al. [13] also finds the common user-interactions patterns from a preprocessed Web server log.
- **Security** is considered an important topic by the research community especially in the area of Web applications. Making RIAs secure is much harder than traditional Web applications due to the nature of RIAs which make them more interactive and vulnerable [6]. Web server logs can be used to detect a potential attack by an intrusion detection system (IDS) which has been studied in a range of research [22, 30, 35, 50].

Web server logs can also be used to determine the source of an attack in forensic analysis and there is a lot of research done in this area [51, 56]. The majority of studies try to detect the source of the attack by clustering HTTP messages into normal and abnormal ones. Moreover, some approaches identify suspicious groups of requests in order to not store the abnormal ones into storage volumes [29].

A user-interactions reconstruction system can be used in forensic analysis to obtain the high level information from Web server log as low level data. Resurf [69] and ClickMiner [44] which were discussed in section 3.1.1 are two examples of such a tool. This document also introduces a new user-interactions reconstruction tool which can be used for forensic analysis even for RIAs as will be shown in chapter 6.

- **Traffic Replay** tools have been designed to replay previously logged HTTP network traffic for debugging purposes or forensic analysis. "Tcpreplay[3] is used to replay a given pcap onto the network to do a test". Hong et al. [23] developed a tool to replay a captured TCP session.

  Selenium IDE[4] is another tool used for capturing and replaying HTTP network traffic. Recording can be only done using Firefox while the replaying phase is not limited to a specific Web browser.

  A user-interactions reconstruction system such as ClickMiner [44] and our implemented tool can also be used to replay previously captured HTTP network traffic.

- **Model Inference** refers to the process of gaining knowledge about the behavior of an application [11]. Different fields of study use the concept of an inferring model so they can either apply their methodology to the extracted model or do some statistical analysis on the application [37, 20, 26].

  One of the most popular areas, which is used to infer the model of a Web application, is "Crawling". A Crawler is used for a range of purposes from content indexing to automating regression testing [42]. There are lots of approaches to get Crawlers work on RIAs [41, 8, 40].

  A user-interactions reconstruction system also can be used as a tool which extracts

[3] `http://tcpreplay.synfin.net/`
[4] `http://docs.seleniumhq.org/`

a sub-model of a Web application. In other words, the user-interactions show a path within a Web application. The path can be considered as a sub-model of the Web application.

## 3.3 Summary

In this chapter we discussed existing methodologies for reconstructing user-interactions. We specified two type of approaches along with their prerequisites.

- On-line user-interactions reconstruction which is done while the user is browsing a website.
  - **Prerequisite number 1:** Tool has to be installed on a user's machine
- Off-line user-interactions reconstruction which is done after a user has stopped browsing a website.
  - **Prerequisite number 1:** The tool should be provided by a captured user-session. Therefore, the HTTP network traffic has to be analyzed for the user-session reconstruction phase.
  - **Prerequisite number 2:** User inputs retrieval.

We also discussed fields of study such as Web usage mining, security, traffic replay and model inference in which a user-interaction reconstruction system can be used.

The next chapter provides an introduction to the framework based on which this document has been written.

# Chapter 4

# The Proposed Architecture

This chapter provides a summary of the proposed solution along with the architecture of the designed tool for the reconstruction of user-interactions. Section 4.1 introduces the proposed framework, as well as its components. Section 4.2 and section 4.3 give the reader a taste for freely available tools which play an important role in the entire process of reconstructing user-interactions to make it feasible.

This research is being designed in a group[1] and this chapter presents the work, done in the group. Although the author of this document has contributed to implementing the contents of this chapter, her main contributions will be discussed in chapter 5. The current document will address the identified challenges described in section 4.4 and will present the experimental results of having implemented the suggested solutions in chapter 6.

## 4.1 Overview of solution

We have proposed replaying formerly captured HTTP network traffic to reconstruct user-interactions. We claim that if we can regenerate the recorded network traffic in a browser, then we have successfully reconstructed the user-interactions. The reason is that regenerating the given HTTP trace means performing the same actions that the user had. The only limitation is the lack of client-side actions in the HTTP network traffic. In fact, the user's Web Browser triggers no HTTP request while parsing CSS files or executing JavaScript

[1] `http://ssrg.site.uottawa.ca/members.html`

functions on the client-side [61]. Therefore, there might be client-side actions which will miss being replayed.

The only input given to our tool is previously captured HTTP network traffic which is recorded while a user interacts with the Web server. We capture the HTTP network traffic generated by the user's Web browser during the entire user-session. Then, our user-interactions reconstruction tool starts replaying the given user-session in order to reconstruct the user-interactions.

We have no assumption on our input. Although there should be a network sniffer to capture the generated HTTP network traffic, the user does not need to care about any particular prerequisite while browsing a website.

Furthermore, the process of reconstructing user-interactions using our approach is done without accessing the original RIA website. This way we have full control on what is sent to our programmable Web browser as an HTTP response. Also, the Web server will not be re-attacked by replaying a malicious user-session using our method. Moreover, the captured HTTP network traffic can be still replayed when the Web server is down due to an attack. Therefore, being able to construct user-interactions without accessing the Web server makes our approach more practical.

We have developed a system, User-Interactions Reconstruction (IR), to replay previously captured HTTP network traffic. IR has two main components, IR-Proxy and IR-Browser which is a programmable Web browser. Figure 4.1 shows the general architecture of the proposed system.

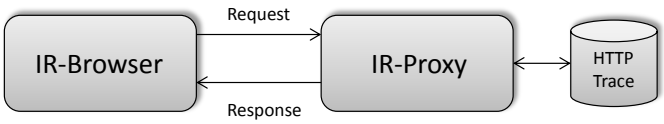


Figure 4.1: Architecture of the IR

IR-Proxy plays the role of the Web server. It has access to the previously captured HTTP network traffic stored in a log file. IR-Proxy uses the HTTP network traffic as a potential response to the requests it receives.

IR-Browser starts from the initial URL. It connects to the IR-Proxy instead of connect-

ing to the Web server. Consequently, IR-Browser sends its HTTP requests to the IR-Proxy in order to ask for the resources it needs for loading each Web page.

IR-Proxy looks into the captured HTTP trace in order to find an HTTP request which matches the one it has received. If IR-Proxy finds a match between the received request and the recorded ones, it will then send back the corresponding response to the IR-Browser's request. Otherwise, IR-Proxy returns an error which tells IR-Browser that the requested resource does not exist.

IR-Browser is limited to "click" as a potential way for executing elements. It generates HTTP requests when an actionable element is executed. Consequently, IR-Browser starts executing elements. If IR-Proxy cannot find a match between the received request and stored ones, then IR-Browser needs to execute another element in order to find the next user-interaction.

To have a clear understanding of the proposed solution, there are basic terms that need to be defined.

- **"State"** refers to the DOM state in which a user either executes one of the available elements inside the DOM or decides to terminate the session.

- **"Stable Condition"** refers to the state in which there will not be any further changes on the current DOM until IR-Browser executes a new event [3].

  IR-Browser must wait until reaching a stable condition after executing any element. IR-Browser decides whether or not it has reached the stable condition by counting the number of triggered HTTP requests along with the number of received HTTP responses. In other words IR-Browser is in a stable condition if the number of triggered HTTP requests equals the number of received HTTP responses.

- **"Sequence Check"** is the message sent by IR-Browser to IR-Proxy after reaching the stable condition. IR-Browser must check whether or not it has executed the correct HTML element in the current step. IR-Proxy is in charge of determining the correctness of the executed element since it has access to the captured HTTP trace.

  IR-Proxy makes a decision about the correctness of the executed element by checking three factors. First, all the triggered HTTP requests should have a match in the

previously captured HTTP network traffic. Second, the set of received requests should have consecutive matches in the log file. This means that there should not be any unmatched request between the first and the last matched requests in the log file. Finally, the block of matched requests that has been found has to be the first unmatched series of requests in the previously captured HTTP trace. Figure 4.2 shows when IR-Proxy passes the "Sequence Check" message, as well as when it fails the message.

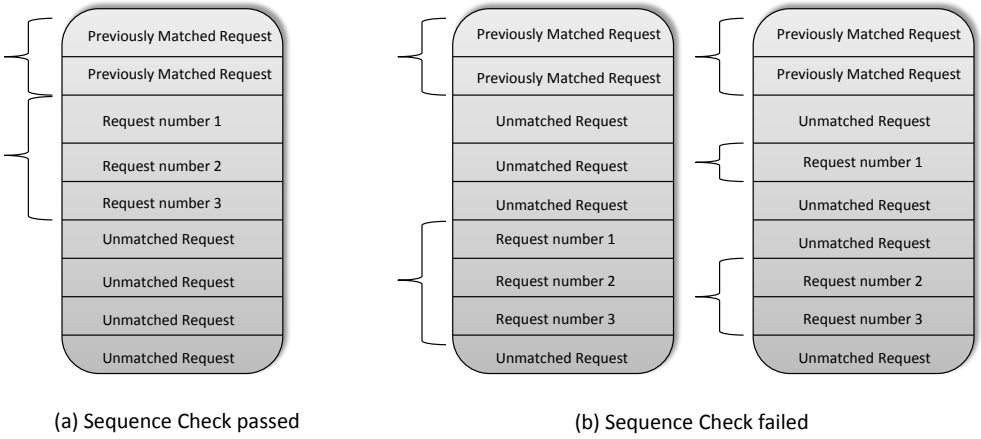


Figure 4.2: Three HTTP requests have been triggered after executing an HTML element

Figure 4.2 (a) shows the situation in which the three aforementioned criteria have been met by the executed element. Figure 4.2 (b) shows the situations in which IR-Browser either has executed an element which should be executed later or it has executed an element which the user had not executed during the user-session. In the former situation the third factor has been broken and caused IR-Proxy to reject the "Sequence Check" message. In the latter situation the first or the second factor has not been met which means that IR-Browser has executed a wrong element.

- **"Previously Known Correct Element"** refers to the element which IR-Proxy has approved as an element which the user had executed during the user-session. Consequently, the stable condition reached by IR-Browser after executing such an element is called the "Previously Approved State".

- **"Reset"** must be performed when IR-Browser has executed an element for which

IR-Proxy rejects the "Sequence Check" message. IR-Browser must go back to the last previously approved state in order to execute another element.

To perform a "Reset", IR-Browser reloads the initial URL and executes all the previously known correct elements in order. This way IR-Browser reaches the last previously approved state from which it can select and execute another element.

An important aspect which IR-Browser should take it into consideration is that IR-Browser must wait to reach a stable condition after executing each previously known correct element during the "Reset". Otherwise, it might try to execute an element which has not been loaded into the DOM yet.

The basic reconstruction algorithm executed by IR-Browser and IR-Proxy is sketched in Algorithms 1 and 2 respectively [3]. Each of these two main components works as follows:

**Algorithm 1:** Sketch of the IR-Browser Algorithm

```
1  ReconstructedElementsList := nil;
2  while not finished do
3      LoadLastPreviouslyApprovedState(ReconstructedElementsList);
4      ElementCandidate = ChooseAndExecuteNextElement();
5      WaitForStableCondition ();
6      if SequenceCheck () then
7          ReconstructedElementsList.push(ElementCandidate);
8      else
9          Reset()
10 return ReconstructedElementsList;
```

At each step, IR-Browser loads/reloads the initial URL/last previously approved state (3) using the list of previously known correct elements. It then needs to select and execute the next element candidate (4). IR-Browser then waits for the stable condition (5) to ask IR-Proxy about the correctness of the executed element (6). IR-Browser adds the executed element into the list of previously known correct element if IR-Proxy confirms the received set of requests. Otherwise, IR-Browser should perform Reset (9).

**Algorithm 2:** Sketch of the IR-Proxy Algorithm

```
1 while not finished do
2     req := GetRequestsFromBrowser();
3     if req is not a "Control Message" request then
4         if matchFound (req) then
5             ReturnMassagedResponse(req);
6         else
7             return 404NotFound;
8     else
9         respondToCtrlMessage (req);
```

IR-Proxy waits until it receives a request from IR-Browser (2). The received request is either an HTTP request for fetching a resource or a request generated by IR-Browser in order to communicate with IR-Proxy. We call the second type of request a "Control Message" and the "Sequence Check" is an example of such a message. Then, IR-Proxy checks the type of received request (3). If it is not a "Control Message", then IR-Proxy tries to find a match for the received request in the log file. IR-Proxy returns the corresponding response to the IR-Browser once it finds the match (5). Otherwise an error message is sent back to the IR-Browser (7). If the received request is "Sequence Check", then IR-Proxy sends back the response after checking the correctness of the executed element (9).

IR uses freely available tools during the process of reconstructing user-interactions. The next two sections describe why IR needs to get help from each of these tools, as well as how IR uses them.

## 4.2 PhantomJS: Headless Web Browser

*PhantomJS*[2] is an open source Web browser which has been developed based on WebKit[3], an open source Web browser engine. PhantomJS is a headless Web browser, which means

[2] `http://phantomjs.org/`

[3] `https://github.com/WebKit/webkit`

that it has no Graphical User Interface (GUI). This feature makes it faster than a Web browser with a GUI. PhantomJS provides a JavaScript API that allows developers to have all the functionalities which they would have available while working with a graphical Web browser [16].

IR-Browser needs to have the basic functionalities of a Web browser to be able to act as a real Web browser. IR-Browser must be able to execute HTML elements and JavaScript functions, as well as rendering various types of Web pages. Consequently, IR-Browser takes advantage of PhantomJS in order to reconstruct the user-interactions. PhantomJS enables IR-Browser to have a full control on the Web page which is an essential requirement for what IR-Browser is supposed to do.

As an example, as mentioned earlier, IR-Browser decides whether or not it has reached the stable condition by counting the number of triggered and received HTTP messages. Counting the number of exchanged HTTP messages is possible because PhantomJS provides IR-Browser with two callback functions as shown in figure 4.3. One is called once an HTTP request is triggered and the other is called upon receiving an HTTP response.

```
var webPage = require('webpage');
var page = webPage.create();
var numberOfHTTPResponses = 0;
var numberOfHTTPRequests = 0;

page.onResourceReceived = function(response) {
  numberOfHTTPResponses ++;
};

page.onResourceRequested = function(request) {
  numberOfHTTPRequests ++;
};
```

Figure 4.3: "onResourceRequested" and "onResourceReceived" callback functions

PhantomJS is a newly released project which makes it unstable in a few aspects. There are features which PhantomJS[4] does not support. It is a headless Web browser which does not need to support features accessible through a GUI. For instance, a user is not able to watch movies through a headless Web browser.

[4]version 1.9.7

As an illustration[5], PhantomJS has no support for plug-ins such as Flash, Videos and Audios, CSS 3-D and WebGL since it is a pure headless Web browser. Also, PhantomJS does not request for the favorite icon known as favicon which is used by the graphical Web browsers in order to show a representative picture of the website to the user.

Each of the available Web browsers has its own features and limitations. We have chosen PhantomJS because it is faster than the other Web browsers since it is a headless Web browser with no need for rendering heavy images or videos. Moreover, PhantomJS provides IR-Browser with callback functions which play an important role in being a stable programmable Web browser.

## 4.3 Web Proxies

A Web proxy is a service which can be used to access Web servers on behalf of a user's Web browser. Consequently, there is no direct connection between the user's Web browser and the Web server [5]. In other words, a Web proxy passes HTTP network traffic between Web browsers and Web servers as shown in figure 4.4. Therefore, a Web proxy has access to both a Web browser's requests and a Web server's responses.

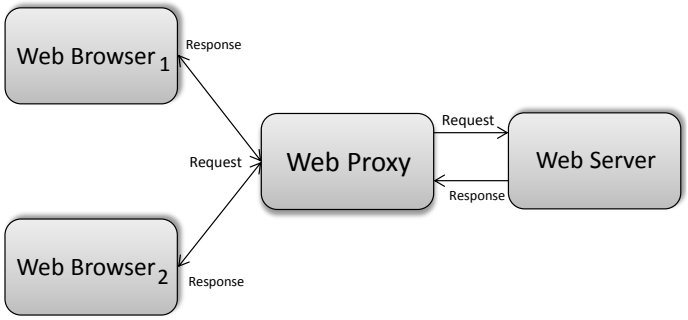


Figure 4.4: Web Proxy

The input provided for IR is a previously captured HTTP trace. The HTTP network traffic exchanged between the user's Web browser and the Web server can be captured either through the Web server or an HTTP network sniffer as discussed in chapter 2.

[5] `http://phantomjs.org/supported-web-standards.html`

Getting access to the server log is not always feasible especially when we have not hosted the website.

Our process of reconstructing user-interactions can be considered as a two-phase process. First, a user-session should be captured and logged. Secondly, IR starts working in order to construct user-interactions from the given user-session. We use two Web proxies. One is used in the first phase for capturing HTTP network traffic and the other is used while IR works.

- *Fiddler*[6] which runs on any Windows-based computer is an example of a Web proxy which also decrypts HTTPS network traffic using a man-in-the-middle decryption technique [32]. The user runs Fiddler to which the user's Web browser has been connected. Then, the user starts browsing the website while Fiddler captures all the exchanged HTTP messages in the background.

  Although the input might be a server log, we have chosen Fiddler to provide IR with an input because Fiddler is freely available and would provide us with decrypted HTTP network traffic. Also, Fiddler creates a single file for each HTTP request, as well as putting together all the received chunked HTTP responses.

- On the one hand, as mentioned above, the given input to the IR is the decrypted HTTP network traffic exchanged between the user's Web browser and the Web server. Consequently, IR-Proxy only has access to the decrypted data and is able to send back only decrypted HTTP responses to the IR-Browser.

  On the other hand, IR-Browser triggers encrypted HTTP requests while communicating with secure websites. Consequently, IR-Proxy cannot find a match for the received encrypted HTTP request in the log file. Therefore, there should be a way by which the triggered requests through IR-Browser be converted to decrypted data before being delivered to the IR-Proxy.

  To provide IR-Proxy with a decrypted set of requests we can either implement the HTTPS protocol in our IR-Proxy or use an available Web proxy as an intermediary component. As discussed earlier, a Web proxy has access to all of the HTTP network

[6] `http://www.telerik.com/fiddler`

traffic. Consequently, it can modify the data before passing it to the next party inside the loop.

*MITMproxy*[7], which is an Open-Source Web proxy, runs on any OSX-based or Linux-based machine. "Mitmproxy is an interactive, SSL-capable man-in-the-middle proxy"[8] which can intercept the network traffic exchanged between a Web browser and a Web server [66].

We have chosen MITMproxy in order to provide IR-Proxy with decrypted network traffic since it is a freely available tool which helps us by eliminating the need to implement the HTTPS protocol in our IR-Proxy. Figure 4.5 shows IR-Proxy and IR-Browser communication through MITMproxy.


Figure 4.5: IR-Browser and IR-Proxy communicates through MITMproxy

As shown in figure 4.5 MITMproxy encrypts HTTP responses sent by IR-Proxy before passing them to the IR-Browser. Otherwise, IR-Browser will not trust the received response since it expects to receive encrypted data in response to its encrypted HTTP requests.

## 4.4 Challenges

The proposed framework has difficulties reconstructing user-interactions for real websites especially for RIAs. There are challenges which should be addressed. Otherwise, IR will fail to reconstruct user-interactions.

- First and foremost, a user-interactions reconstruction system must handle user-inputs if that is a part of the captured user-session. IR-Browser must generate the exact

[7] `https://mitmproxy.org`
[8] `https://github.com/mitmproxy/mitmproxy`

sequence of requests that occurred during the recorded session which is impossible if IR-Browser cannot redo all the user's actions. Therefore, IR should be able to submit the user-inputs if that is a part of the user-session.

- Secondly, a real RIA contains Web pages with a large number of elements. In the proposed framework, IR-Browser executes all existing elements one after the other until it finds the next correct one. Consequently, the process of reconstructing user-interactions is a costly and time-consuming process. Therefore, there should be a way by which IR-Browser can efficiently find the next correct element inside the DOM.

- Finally, a Web browser might generate a different set of HTTP requests every time a user executes an element. This happens because there might be a request contains parameters which take different values each time. Therefore, IR must determine whether or not the difference between received HTTP requests and the ones in the log file is due to the client-side non-determinism. Otherwise, IR will fail in reconstructing user-interactions for the given user-session.

## 4.5 Summary

In this chapter, we presented a framework for a user-interactions reconstruction system. We have also identified three important issues which should be considered.

1. **Handling user-inputs** which has to be done if there are user-inputs involved during the captured user-session.

2. **Finding the next correct action efficiently** is an important challenge which should be addressed. Otherwise, IR would not act as an efficient user-interactions reconstruction tool if the website is a real RIA with a large number of elements in each Web page.

3. **client-side non-determinism** which causes IR to fail replaying a given captured HTTP trace.

The next chapter provides the solution for each of the identified challenges. It shows the way in which the proposed framework works as a real user-interactions reconstruction system.

# Chapter 5

# User-Interactions Reconstruction System

In chapter 4 we reviewed the framework by which user-interactions reconstruction is feasible. We identified three challenges which cause IR to be either inefficient or unable to reconstruct user-interactions. In this chapter we provide a solution for *"handling user-inputs"*, *"finding the next correct action efficiently"*, and *"client-side non-determinism"*.

The rest of this chapter is organized as follows; section 5.1 shows the way by which IR can find the next correct user action efficiently. Section 5.2 suggests the way by which IR can recognize client-side non-determinism while replaying the HTTP network traffic. In section 5.3 we explain the challenges we faced while implementing our solutions. Finally, section 5.4 summarizes the proposed solutions.

## 5.1 Next Correct User Action

As will be shown in chapter 6 the cost of reconstructing user-interactions is extremely high if IR simply executes all the DOM elements to find the correct one. This section explains the way in which IR-Browser and IR-Proxy collaborate for finding what the next action should be in order to generate the next unconsumed block of HTTP requests inside the IR-Proxy's database.

IR-Browser has the full control on the current DOM and IR-Proxy has access to the full

HTTP trace which should be replayed. Therefore, the idea is to use the first unconsumed HTTP request inside the IR-Proxy's database to find out which action should be done on the current DOM.

Each action is considered either clicking on an HTML element or submitting user inputs to the Web server. Consequently, IR should figure out what the type of next action is in each step. Therefore, the rest of this section is organized as follows.

First we describe what the actionable element is and how we find such an HTML element inside the DOM in 5.1.1. Secondly we explain the way in which IR decides what the type of next action is in 5.1.2. Then we describe how IR finds the next correct action efficiently in 5.1.3.

### 5.1.1 Actionable HTML Elements

First and foremost not all the DOM elements are actionable. There are elements to which there is no event handler attached, as well as those which are invisible from the user. IR-Browser does not trigger HTTP requests after executing a non-actionable element. Consequently, IR must be able to distinguish actionable elements from those which are not intended to change the state of the DOM. Otherwise, IR-Browser has to perform "Reset" after executing a non-actionable element which causes IR-Proxy to reject the corresponding "SequenceCheck" message.

As discussed in chapter 4 IR has to perform "Reset" after executing a wrong element. IR-Proxy is responsible for determining the correctness of the executed element. It refuses the "SequenceCheck" message when no HTTP request has been triggered. Therefore, the idea is to identify non-actionable elements in order to postpone executing them.

An HTML element is considered actionable if either there is an event handler attached to the element or it is an <a> element in which a hyperlink has been specified. Although the second type of actionable elements is distinguishable by traversing the DOM and extracting <a> elements, the first type of actionable elements are not always easy to identify.

An event handler can be attached to an HTML element via two different ways. First, a JavaScript function can be explicitly assigned to the "onClick" attribute of an element. Consequently, such an element can be identified by parsing the DOM and scanning the

"onClick" attribute of all HTML elements. Figure 5.1 shows the DOM representation of a  element with an "onClick" attribute.

```
<div onclick="ajaxPageChange(&quot;tm_services.txt&quot;,&quot;TopMenuID&quot;);
    ajaxPageChange(&quot;index_services.txt&quot;,&quot;ContentID&quot;);
    ajaxPageChange(&quot;menu_services.txt&quot;LeftMenuID&quot;);">Services</div>
```

Figure 5.1: DOM representation of a  element with "onClick" attribute

The "onClick" attribute can also be assigned to an HTML element dynamically. Figure 5.2 shows an example of a JavaScript function which assigns a value to the "onClick" attribute of some HTML elements in a Web application dynamically. In such a case, the JavaScript function will be called upon executing the corresponding HTML element.

```
window.onload = function() {
  var all_tds = document.getElementsByTagName("td");
  for (var e = 0; e < all_tds.length; e++) {
    all_tds[e].onClick = function() {
      id = this.getElementsByTagName('strong')[0].firstChild.nodeValue;
      getAjax("GET", "content/content" + id + ".html", true);
    }
  }
}
```

Figure 5.2: Assigning value to the "onclick" attribute in a RIA dynamically [3]

The second way by which an event handler can be attached to an HTML element is to use method "addEventListener" [65]. This way the event handler is added through a call to "addEventListener" method within a user's Web browser. An event handler can also be removed from an HTML element by making a call to the "removeEventListener" method.

The user's Web browser starts executing JavaScript functions, which should be executed upon loading the HTML page, after fetching all resources from the Web server. Consequently, the call to "add/removeEventListener" will be made if there is any event handler which is intended to be attached to HTML elements via these methods.

The process of replaying the captured HTTP network traffic is done without accessing the original Web application. IR can have full control of the target Web application since

IR-Proxy has access to the entire user-session and IR-Browser renders the Web application. Therefore, having overwritten the above-mentioned methods is our solution to keep track of HTML elements with attached event handlers. Figure 5.3 shows the way in which the "add/removeEventListener" methods are overwritten.

```
<script>
  var eventHandlers = new Array();
  var Original_addEventListener = Element.prototype.addEventListener;
  var Original_removeEventListener = Element.prototype.removeEventListener;
  var EventListener = function(type, listener) {
    Original_addEventListener.call(this, type, listener);
    eventHandlers.push(getXPath(this));
  };
  var removeEventListener = function(type, listener) {
    Original_removeEventListener.call(this, type, listener);
    for(var i = 0; i < eventHandlers.length; i++) {
      if (eventHandlers[i] == getXPath(this)) {
        eventHandlers.splice(i, 1);
        break;
      }
    }
  };
  Element.prototype.addEventListener = EventListener;
  Element.prototype.removeEventListener = removeEventListener;
</script>
```

Figure 5.3: Hijacking event listener methods to keep track of event handlers.

We use the XPath of HTML elements to keep track of them. Consequently, we store the XPath of the elements which are passed to the "addEventListener" method into our global object and remove the Xpath of the elements which are passed to the "removeEventListener" method from our global object. We also keep the actual listener which is attached to each HTML element in order to use it in our further analysis. This way IR-Browser has the list of all actionable elements once the Web page is loaded.

Our proposed script has to be executed before any other JavaScript function. Otherwise, we will miss the chance of storing actionable elements. Consequently, our script should be injected on top of all existing scripts inside the DOM. This way, a user's Web browser executes our script first since it starts parsing the DOM from top to bottom.

We use IR-Proxy for injecting our script to the DOM. As mentioned earlier IR-Proxy has full control on what it sends to the IR-Browse. Therefore, IR-Proxy adds our script to the DOM by modifying the HTTP response which is going to be sent to the IR-Browser. This way IR will have the list of all actionable HTML elements when the page is loaded.

### 5.1.2 Type of Next User's Action

Each user-action can be either submitting user inputs to the Web server or clicking on an HTML element inside the current DOM of the Web application. Therefore, IR should determine what the type of the next action is after reaching the stable condition at each step.

We suggest two new control messages called "UserInputSubmission" and "NextUnconsumedHTTPRequest" so that IR-Browser and IR-Proxy collaborate in order to figure out what to do at each step. The idea is to scan the first unconsumed HTTP request inside the IR-Proxy's database to figure out what should be done.

In any step, after reaching the stable condition, IR checks if there is any user input that could be submitted. Consequently, IR-Browser traverses the current DOM of the Web application and follows the following steps once it extracts the list of all the form elements inside the DOM.

- **Extracting Specifications of the Form Element:** IR-Browser extracts all the <name, value> pairs of the inputs with regard to their specified attribute "type".

  Attribute "type" has an impact on the process. It helps both IR-Browser and IR-Proxy to proceed in their task. IR-Proxy should consider the "type" of elements while scanning the first unconsumed HTTP request in order to determine whether or not this HTTP request stands for the extracted form element. IR-Browser should consider the "type" of each element while filling them out.

  As an example, elements with type "checkbox" will be a part of the query string in generated HTTP requests only if they have been selected by the user. Otherwise, they will not be submitted to the Web server. Consequently, IR-Proxy should take the "type" of elements into consideration while scanning the next unconsumed HTTP

request. Also, IR-Browser fills out field elements using JavaScript functions. It has to consider the "type" of element since there are different functions which should be used for assigning value to different types of field element.

We extract the current value of the elements inside the DOM because, there might be field elements with default values. Consequently, IR-Browser should not rewrite these values, since they will be submitted to the Web server eventually if they are supposed to be submitted.

- **Issuing an HTTP POST Request:** After extracting all the specifications of each form element IR-Browser sends a control message to IR-Proxy. The control message contains all the extracted <name, value> pairs in the body of the HTTP request.

  IR-Proxy distinguishes between the elements with type "hidden" and other elements while scanning the first unconsumed HTTP request. The reason is that all the "hidden" elements will be submitted to the Web server once the user submits the form element. Therefore, IR-Proxy considers these elements as a part of its analysis. All the "hidden" elements should exist in the first unconsumed HTTP request. Otherwise, IR-Proxy rejects submission of the target form element.

  IR-Browser issues one control message for each extracted form element to the IR-Proxy. Consequently, IR-Browser keeps sending control messages, with different specifications, to IR-Proxy until either there is no more form element or IR-Proxy sends the message containing all the values which means it has approved the submission of the current form element.

  IR-Browser continues applying the following steps if IR-Proxy approves the submission of a form element.

  - **Extracting User Inputs from the Received HTTP Response and Filling the Form Out:** IR-Browser parses the received message from IR-Proxy which is the part of the first unconsumed HTTP request containing the values used by the user. Consequently, IR-Browser extracts the value of each field element and puts the extracted value inside the corresponding field element within the DOM.

– **Submitting the Form Element:** IR-Browser submits the filled form element as will be discussed in 5.1.3

IR-Browser starts extracting the specifications of all the single field elements if IR-Proxy refuses to submit all the form elements. IR-Browser will redo all the above-mentioned steps to figure out whether or not the user had submitted a single field element at this step.

The type of next user-action is considered "click" if IR-Proxy refuses the submission of all the existing user inputs inside the current DOM of the Web application. Therefore, IR-Browser should execute HTML elements to find the next user-interaction.

Trying all HTML elements is an inefficient way of finding the user action especially when dealing with a real RIA in which there are thousands of HTML elements. Consequently, IR-Browser issues "NextUnconsumedHTTPRequest" control message to the IR-Proxy in order to get the first unconsumed HTTP request from the IR-Proxy to figure out which HTML element is more likely to generate the next unconsumed HTTP request. We describe how IR finds the next user-interaction efficiently in 5.1.3.

### 5.1.3 Execution of the User's Action

It was explained in section 5.1.2 how IR decides what should be the type of the next user-action at each step. In this section we explain the way in which IR executes the selected user-action.

The idea is to prioritize and to sort HTML elements instead of executing elements one after the other. This way IR-Browser creates an ordered list of HTML elements to be executed in order to generate the next unconsumed HTTP request inside the IR-Proxy's database. Consequently, we define different factors for prioritizing HTML elements based on determined type of user-action.

- **User-Inputs:** The way by which IR-Browser gets and enters the value of the field elements from the given HTTP trace was explained in 5.1.2. Here we explain how IR-Browser submits the user inputs to the IR-Proxy.

  IR-Browser should submit either a single field element or a collection of them. In the former case IR-Browser first determines whether or not there is an event handler

attached to the field element. IR has the list of all actionable HTML elements along with their attached event handler as discussed in 5.1.1. Consequently, IR-Browser makes a call to the event handler of the field element if it finds there is an event handler attached to the element. Otherwise, as for the latter case, it prioritizes HTML elements to create the ordered list which should be executed in order to submit user-inputs.

IR-Browser traverses the current DOM of the Web application to assign a priority to each HTML element based on following factors.

- **Actionable element:** The first factor which IR-Browser checks is whether or not the element is actionable.
- **<button> elements and <input> elements with type "submit":** These elements take the higher priority compared to other HTML elements. IR-Browser considers following factors to assign the final priority to each of these elements.
  - **How close the HTML element is to the target field/form element:** IR-Browser gives the higher priority to the elements which are closer to the HTML field. To identify how close the elements are we compare the depth of each HTML element inside the DOM with respect to the target field/form element.
  - **OuterText of the HTML element:** Users figure out which HTML element submits the target field/form by looking at the outer text of the HTML elements. Consequently, IR-Browser uses a dictionary of the words which are used as the outer text of the submit buttons. Consequently, IR-Browser assigns the higher priority to the element which has the same outer text as one of existing ones in our dictionary.
- **Rest of the HTML elements:** IR-Browser uses the two above-mentioned factors to assign the priority to other HTML elements inside the DOM. Consequently, it takes into the account the "OuterText" of HTML elements, as well as determining how close the element is to the target field/form element to assign the priority to each HTML element.

- **Click:** IR-Browser gets the first unconsumed HTTP request from IR-Proxy via a "NextUnconsumedHTTPRequest" control message in order to find the HTML element which might trigger the HTTP request.

  To find the next correct element IR-Browser first parses the received HTTP request in order to extract the URL of the received request together with all the parameters and their values. Then, IR-Browser parses the current DOM of the Web application in order to scan all HTML elements. Therefore, IR-Browser assigns a priority to each HTML element, with regard to the extracted data from the received HTTP request, based on the following factors.

  - **Actionable element:** The first factor is to check whether or not the element is actionable. IR-Browser has access to the list of all actionable elements inside the DOM. Therefore, an element gets the lower priority if it is not an actionable one. Otherwise, IR-Browser checks the remaining factors before assigning the priority to the HTML element.

  - **<a> elements:** The second factor which is considered by IR-Browser is the value of the "href" attribute if the target element is an <a> one. Consequently, if the value of the "href" attribute is the same as the URL of the next unconsumed HTTP request the element takes the highest priority. Otherwise, it takes the lowest possible priority since it is intended to generate a different HTTP request after being executed.

  - **"onClick" attribute or attached "listener":** IR-Browser scans the value of the "onClick" attribute or the previously stored "listener" of the HTML element in order to assign a priority to the target HTML element. Consequently, the priority will be assigned to the elements based on the number of extracted data which are found in the value of their "onClick" attribute or their attached "listener".

  After traversing the current DOM of the Web application IR-Browser creates a sorted list of HTML elements. Then it starts executing HTML elements in order so that IR can find the correct action by generating the first unconsumed block of HTTP requests inside the IR-Proxy's database.

### 5.1.4 Summary

This section explained the way by which IR determines what the next action should be in order to reconstruct the user-interactions. Two new control messages called "UserInput-Submission" and "NextUnconsumedHTTPRequest" by which IR-Browser and IR-Proxy collaborate were introduced in this section.

The idea behind our proposed solutions is that IR has the full control of Web application while reconstructing the user-interactions, as well as having access to all the information about the captured user-session.

## 5.2 Non-Determinism Client-Side Actions

As discussed in chapter 4 IR reconstructs user-interactions by replaying a previously captured user-session. In this section we explain in more detail how IR-Proxy responds to a "SequenceCheck" control message in order to describe one possible challenge which might prevent IR from reconstructing the user-interactions.

IR-Proxy approves the "SequenceCheck" control message if and only if it has received the first unconsumed block of HTTP requests from the IR-Browser side. This will not happen if IR-Browser executes a wrong action at that time.

IR-Proxy searches inside its database to find the received HTTP request. It rejects the existence of the request unless it finds an HTTP request which matches exactly what it has received.

There is a possibility to have an HTML element for which the triggered block of HTTP requests is different from one execution to the other.

- **Server-side non-determinism:** As mentioned in 2.3 there are input elements with type "hidden" for which the value is set by the Web server. The value of these elements cannot be modified by the user. They are used to provide the Web server with more data about the form submission [38]. These values also allow Web developers to retain data which should be submitted to the Web server as a part of processing data

[52]. Therefore, it is possible that the Web server randomly generates and assigns a value to an input.

IR will not encounter a problem replaying HTTP network traffic captured from a Web application in which there is server-side non-determinism. The reason is that IR uses a previously captured user-session and it does not modify any of the assigned values inside the DOM. Therefore, IR-Browser renders the same DOM as the one that the user's Web browser had rendered and the triggered HTTP requests will carry the same values as the captured HTTP trace.

- **Client-side non-determinism:** There might be HTML elements with an event handler which randomly generate values for the parameters that are going to be sent to the Web server upon being executed. Consequently, the HTTP request triggered by executing such an element will be different from one execution to the other.

  One reason for having such an event handler is to force the user's Web browser to fetch the corresponding resource from the Web server every time that the element is executed.

  IR will face a problem reconstructing user-interactions if there is client-side non-determinism involved in the Web application. The reason is that IR-Browser executes the attached event handler upon executing the corresponding HTML element. Consequently, it makes an HTTP request with newly generated values different from the ones that the user's Web browser had generated during the captured user-session.

With regard to the above explanation, IR fails to reconstruct user-interactions if IR-Proxy rejects the execution of an HTML element when the generated block of HTTP requests does not match any block inside the previously captured HTTP network traffic. Consequently, IR-Proxy should figure out why the received set of HTTP requests does not match any block inside the previously captured HTTP trace so that it approves the "SequenceCheck" control message if there is client-side non-determinism involved in the performed action.

We suggest re-executing an HTML element if there is a possibility for having involved client-side non-determinism in the generated block of HTTP requests. Consequently, IR-Proxy should analyze the received HTTP requests against the block which has to

be matched inside the given user-session. IR-Proxy asks IR-Browser to re-execute the same action when it finds that the received set of HTTP requests matches the first unconsumed block of HTTP requests inside the IR-Proxy's database if there is client-side non-determinism involved.

IR-Proxy stores all received HTTP requests for which it cannot find any match inside the previously captured user-session. IR-Proxy starts analyzing the received set of HTTP requests once it receives the "SequenceCheck" control message.

First IR-Proxy checks whether or not the number of received matched and unmatched HTTP requests can create a block of requests which matches the first unconsumed block of HTTP requests inside the captured user-session. Then, IR-Proxy compares each unmatched received HTTP requests with the unmatched HTTP requests inside its database. If the only difference is in the value of some parameters within the received URL, then IR-Proxy asks IR-Browser to re-execute the same action. This way IR-Proxy will check whether or not the values of those identified parameters change in the next execution. IR-Proxy sends the corresponding HTTP response to IR-Browser if it finds that the received HTTP request can match the one inside its database.

Figure 5.4 shows the situation in which IR-Browser has triggered a sequence of eight HTTP requests after executing an HTML element.

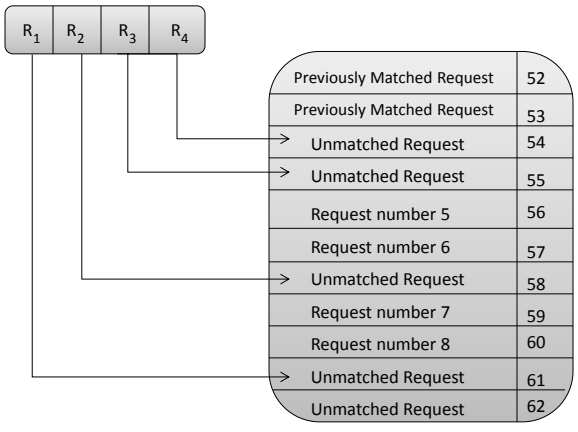


Figure 5.4: Diagram for possible client-side non-determinism

There are four unmatched requests which IR-Proxy has stored for further analysis. IR-Proxy selects the first unconsumed block of eight HTTP requests which might match the

received one if there is client-side non-determinism involved.

IR-Proxy compares $R_i$ ($0<i<5$) with each unmatched HTTP request in the captured user-session. IR-Proxy stores the specifications of $R_i$, as well as the possible index in which the corresponding response has been located as shown in figure 5.5.

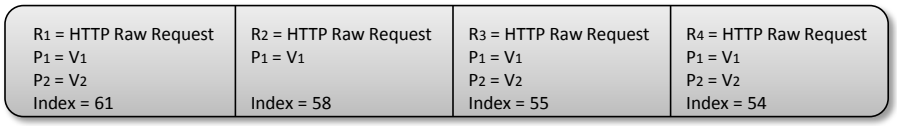


Figure 5.5: Quick view of what IR-Proxy keeps before IR-Browser re-executes an action

IR-Browser re-executes the same action once IR-Proxy asks. If IR-Proxy cannot find any match between the received HTTP request and the ones inside the captured user-session, then it compares the received HTTP request with each of stored ones in the previous step. It will send the corresponding response to the IR-Browser if the only difference is in the value of the parameters which had been flagged in the prior step.

## 5.3 Other Issues

In this chapter we proposed our solutions for challenges identified in chapter 4. We faced other challenges while implementing our suggested solutions. We itemize those problems and the way in which we dealt with them in this section.

- **Keeping track of HTML elements** is a challenge for IR. IR needs to collect all actionable HTML elements so it can distinguish between these elements and non-actionable ones during its analysis. In addition, IR-Browser creates an ordered list of elements either for submitting user inputs or executing elements. In both cases IR should be able to uniquely identify HTML elements.

  Using HTML attributes can cause a problem for IR, since not all the HTML elements have a unique value for their attributes. Considering the "id" attribute as an example, there might be elements which have no "id" attribute or others which have been defined with the same value.

Using JavaScript IR can create a list of all HTML elements in the order they have been located in the DOM. This way, IR should keep track of indices of the elements. Consequently, IR has to recreate the list of elements every time it wants to use an element. Using JavaScript for keeping track of elements will be costly for IR since it needs to recreate the list of elements to analyze them at each step.

XPath is the identifier which IR uses. Figure 5.6 shows a form element along with all its identifiers used by IR to submit extracted user inputs.

```
<form name="cmt" method="post" action="comment.aspx">
  <input type="hidden" name="cfile" value="comments.txt">
  <table border=0>
    <tr><td>To:</td><td valign=top><b>Online Banking</b> </td></tr>
    <tr><td>Your Name:</td><td><input name=name type=text value=" "></td></tr>
    <tr><td>Your Email Address:</td><td><input name=email_addr type=text></td></tr>
    <tr><td>Subject:</td><td valign=top><input name=subject size=25></td></tr>
    <tr><td>Question/Comment:</td><td><textarea name=comments></textarea></td></tr>
    <tr>
      <td> </td>
      <td><input type=submit value="Submit" name="submit"> <input type=reset
          value="Clear Form" name="reset"></td>
    </tr>
  </table>
</form>

Form: div[class=fl]/form[1]
  input #1: div[class=fl]/form[1]/table[1]/tbody[1]/tr[2]/td[2]/input[1]
    input #2: div[class=fl]/form[1]/table[1]/tbody[1]/tr[3]/td[2]/input[1]
    input #3: div[class=fl]/form[1]/table[1]/tbody[1]/tr[4]/td[2]/input[1]
    input #4: div[class=fl]/form[1]/table[1]/tbody[1]/tr[5]/td[2]/textarea[1]
    submit button:  div[class=fl]/form[1]/table[1]/tbody[1]/tr[6]/td[2]/input[1]
```

Figure 5.6: Form Element and Its Unique Identifiers

- **Cache** of the Web browsers has been designed to speed up loading Web pages. Web browsers keep a copy of some received resources in order to use them later instead of fetching the resource from the Web server every time. Consequently, IR may encounter problems replaying captured HTTP network traffic because of the logic on which it works.

  IR-Browser finds each user action by trial and error. It executes an HTML element

and generates a set of HTTP requests to the IR-Proxy. IR-Browser caches some of the resources upon receiving them from IR-Proxy. Figure 5.7 shows how the caching mechanism used by IR-Browser can cause problems for IR.

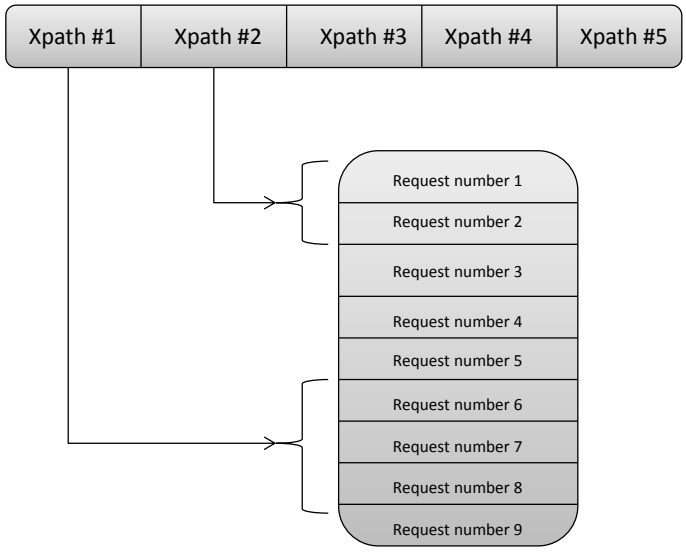


Figure 5.7: PhantomJS' cache may effect user-interactions reconstruction

IR-Browser creates an ordered list of the XPath of all actionable elements which should be executed. If "XPath #3" corresponds to the correct user action, then IR-Browser has to have executed "XPath #1" and "XPath #2" before executing this element based on the created list of elements.

The block specified in figure 5.7 corresponds to the one which is triggered upon executing "XPath #3". "XPath #1" and "XPath #2" have some common HTTP requests in this block. Consequently, IR-Browser may cache these common HTTP requests once it executes these two HTML elements. Therefore, IR-Browser will not trigger the entire unconsumed block of HTTP request upon executing "XPath #3". In other words, if some HTTP requests have been wrongly issued early, then they can be cached and used when they should be generated. This causes IR-Proxy to reject the correctness of the executed action which is in fact the correct one.

At the time of writing this document the stable version of PhantomJS[1] does not

[1] 1.9.7

have the ability of clearing or even disabling the cache. Therefore, we suggest killing the entire process when IR-Proxy fails the "SequenceCheck" control message. This way, IR-Browser starts working while it has no resource cached when IR performs "Reset".

In addition to above-mentioned scenario the data can also be stored on the client's machine in order to be used in further processes using "localStorage"[2]. The stored data is considered to be permanent cookies so that it can even be used for different user-sessions on the same machine.

IR should make sure that IR-Browser has no previous knowledge about the user-session. It has to learn both user behavior and the Web application while replaying the user-session. Otherwise, the above-mentioned scenario may happen. Since PhantomJS does not delete the locally stored data automatically IR should remove the corresponding file when it wants to start replaying the captured user-session.

- **"Sequence Check"** fails when IR-Browser has not generated the first unconsumed block of HTTP requests after executing an HTML element. We mentioned that it is possible for IR-Browser to execute the correct element and generate a different set of HTTP requests. We suggest the way by which we handle such a situation in section 5.2. Moreover, we talked about the caching mechanism of the Web browser as another source of this problem. In this part we describe some of the limitations we face because of our headless Web browser, as well as solutions we use to overcome those limitations.

  - As discussed in chapter 4 PhantomJS does not support fetching some kind of HTTP requests such as Flash and favorite icon. Consequently, it causes IR-Browser to not generate the same block of HTTP requests after performing the same action as the user. To deal with this problem, the idea is to not load those HTTP requests which will not be triggered by IR-Browser. Therefore, IR-Proxy checks each HTTP request while loading the user-session in its database. It drops the HTTP requests which it knows are not going to be generated by IR-Browser.

[2] `http://www.w3.org/TR/webstorage/`

As an example, the HTTP request for fetching the favorite icon can be triggered in different ways depending on how it has been specified in the HTML document. Figure 5.8 shows different ways of having the favorite icon specified within an HTML file.

```
1. <link rel="shortcut icon" href="http://example.ca/image.ico" />
2. <link rel="icon" type="image/vnd.microsoft.icon" href="http://example.ca/image.
    ico" />
3. <link rel="icon" type="image/x-icon" href="http://example.ca/image.ico" />
4. <link rel="icon" href="http://example.ca/image.ico" />
5. <link rel="icon" type="image/gif" href="http://example.ca/image.gif" />
6. <link rel="icon" type="image/png" href="http://example.ca/image.png" />
7. "favicon.ico" located inside the Web server
```

Figure 5.8: Different Ways of Fetchin “favicon”

Therefore, IR-Proxy should do a preprocessing step on captured HTTP network traffic to figure out which HTTP request corresponds to the favorite icon so that it is able to not load it in its database.

– There are newly introduced HTML attributes such as “srcset”[3] which the current stable version of PhantomJS[4] does not support. “srcset” has been designed to provide a user’s Web browser with responsive images. The “srcset” attribute is used in <img> elements. It allows developers to use multiple images with different resolutions for a resource. There should be clues for the user’s Web browser about when each image is requested [14]. If the Web browser does not support the “srcset” attribute, then it will only request the resource specified in the “src” attribute.

  With regard to the above-mentioned limitation, IR-Browser might trigger an HTTP request for an image for which there is no match in the previously captured user-session. If that happens, there will be an HTTP request for that image in IR-Proxy’s database which IR-Browser does not request. To deal with this potential problem the idea is to ignore unmatched HTTP requests if they

[3] http://www.w3.org/TR/2013/WD-html-srcset-20130228/
[4] 1.9.7

have been triggered to fetch some images from the Web server while analyzing the correctness of the executed action.

Another preprocessing step which is done by IR-Proxy is to identify what the domain of the initial URL is in order to drop all the captured HTTP network traffic that goes to a different domain. The reason is that if there is going to be HTTP requests going to another domain like the ones for advertisements, the chance of having a different behavior from one Web browser to another increases. As an example figure 5.9 compares the URL of an HTTP request to go to Google Analytics triggered by "Google Chrome" and "PhantomJS". Therefore, we limit ourselves within the domain specified in initial URL.

```
1. Google Chrome
  "http://www.google-analytics.com/__utm.gif?utmwv=5.6.7&utms=2&utmn=1073906570&utmhn=
      demo.oscommerce.com&utmcs=UTF-8&utmsr=1600x900&utmvp=1583x731&utmsc=24-bit&utmul
      =en-us&utmje=1&utmfl=19.0%20r0&utmdt=Software%2C%20osCommerce%20Demo&utmhid
      =1726503788&utmr=0&utmp=%2Findex.php%3FcPath%3D2&utmht=1446422348003&utmac=UA
      -18245216-7&utmcc=__utma%3D231367050
      .382084132.1437703404.1445266082.1446422338.7%3B%2B__utmz%3D231367050
      .1445266082.6.5.utmcsr%3Dgoogle%7Cutmccn%3D(organic)%7Cutmcmd%3Dorganic%7Cutmctr
      %3D(not%2520provided)%3B&utmjid=&utmu=qAAAAAAAAAAAAAAAAAAAAE~"
2. PhantomJS
  "http://www.google-analytics.com/ga.js"
```

Figure 5.9: The URL of the HTTP request goes to Google Analytics Triggered by "Google Chrome" and "PhantomJS"

## 5.4 Summary

In this chapter we suggested solutions for each of mentioned challenges in chapter 4. We defined new control messages through which IR's main components communicate in order to make feasible and efficient the process of reconstructing the user-interactions. The improved reconstruction algorithm is sketched in Algorithms 3 and 4 respectively.

Loading/reloading the initial/last known correct state is still the first step of the algorithm (3). Then IR should check to see if there is any user input to be submitted (4).

**Algorithm 3:** Sketch of the Improved IR-Browser Algorithm

```
1  ReconstructedActionsList := nil;
2  while not finished do
3      LoadLastKnownGoodState(ReconstructedActionsList);
4      UserInputSubmission(UserInputSpecificationsList);
5      if UserInputApproved(UserInputSpecificationsList) then
6          SetCorrespondingValues ()
7          EventCandidate = UserInputSubmission();
8      else
9          NextUnconsumedHTTPRequest ()
10         EventCandidate = ChooseAndExecuteNextEvent();
11     WaitForStableCondition();
12     if SequenceCheck () then
13         ReconstructedActionsList.push(EventCandidate);
14     else
15         if TryCurrentActionAgain () then
16             GoOneStepBack ()
17         else
18             Reset ()
19 return ReconstructedActionsList;
```

IR-Browser fills the field elements out (6) and submits the target element (7) once IR proxy approves the submission of the user inputs (5). IR-Browser asks for the first unconsumed HTTP request (9) in order to choose and execute the next HTML element (10) if IR-Proxy rejects the submission of the user inputs.

IR-Browser waits for the stable condition (11) after executing each action. It sends a "SequenceCheck" control message to the IR-Proxy (12) to check the correctness of the executed action. It updates the list of correct actions (13) if IR-Proxy approves the "SequenceCheck" control message. Otherwise, IR-Browser will either re-execute the same action (16) or perform reset (18) depending on why IR-Proxy has rejected the "SequenceCheck" control message (15).

On the other side, as shown in algorithm 4, IR-Proxy does preprocessing on the captured HTTP network traffic (1) in order to identify requests which IR-Browser does not support due to the limitations of PhantomJS mentioned in section 5.3, as well as all the out-of-domain requests. Then it waits to receive an HTTP request from IR-Browser (3). IR-Proxy searches inside the database to find the corresponding response to the received HTTP request (6) if the received request is not a control message (4).

IR-Proxy analyzes the first unconsumed HTTP request upon receiving the "UserInput-Submission" control message (10) and returns the corresponding values (12) if the next unconsumed HTTP request stands for user inputs submission (11). It also sends the first unconsumed HTTP request to the IR-Browser (17) if the received control message is "NextUnconsumedHTTPRequest" (16).

IR-Proxy starts analyzing the received sequence of HTTP requests upon receiving a "SequenceCheck" control message. IR-Proxy sends the result of its analysis to the IR-Browser (19) in order to let it know whether or not it has performed a correct action.

The effect of all described heuristics in this chapter will be measured experimentally in chapter 6.

**Algorithm 4:** Sketch of the Improved IR-Proxy Algorithm

```
1  DoPreProcessingToLoadTheTrace ()
2  while not finished do
3      req := GetRequestsFromBrowser();
4      if req is not a Control Message then
5          if matchFound (req) then
6              ReturnMassagedResponse(req);
7          else
8              return 404NotFound;
9      else
10         if req is to ask for user input submission then
11             if UserInputShouldBeSubmitted(req) then
12                 ReturnCorrespondingValuesForUserInput ();
13             else
14                 return 404NotFound;
15         else
16             if req is to get the next unconsumed HTTP request then
17                 ReturnNextUnconsumedHTTPRequest ();
18             else
19                 respondToSequenceCheck ()
```

# Chapter 6

# Experimental Results

In this chapter we provide the experimental results of our proposed system. Any system which has been designed to reconstruct the user-interactions by replaying HTTP network traffic should be able to handle user input recovery and client-side non-determinism. Such a system should also be able to check the correctness of the triggered sequence of requests along with reloading the last previously approved state. Otherwise, reconstruction of the user-interactions will not be feasible. We call such a system the "Basic Solution" which performs an in-depth search on all the DOM elements in order to find the next correct action [3].

The rest of this chapter is organized as follows; section 6.1 describes the way in which our experiments are conducted. It also determines what the criteria are for comparing our proposed method with the "Basic Solution". Section 6.2 introduces the target websites which have been selected as our test cases. Section 6.3 provides the experimental results for each of the mentioned test cases, as well as giving an explanation on the effectiveness of our approach. To conclude this chapter, section 6.4 summarizes our observations from the experimental results.

## 6.1 Experimental Setup

The goal of this chapter is to provide experiments to measure the effectiveness of our proposed system for reconstructing the user-interactions. Section 6.3 puts our implemented

algorithms against "Basic Solution" to show how our modifications affect the process of reconstructing the user-interactions. We compare the performance of each approach from various factors in section 6.3. Our criteria are as follows;

- **Average number of actionable elements** from which IR-Browser is supposed to pick the correct one. In chapter 5 we have claimed that the number of elements inside the DOM affects the performance of the user-interactions reconstruction system. We have suggested reducing the number of elements to the actionable ones.

- **Number of control messages** sent by IR-Browser to the IR-Proxy. The number of control messages sent by IR-Browser depends on how fast and efficient the user-interactions reconstruction system works. The number of triggered control messages increases if the number of wrong actions done by IR increases. Moreover, the number of triggered control messages has a direct impact on the execution time of reconstructing the user-interactions.

- **Execution time** of reconstructing the user-interactions is a factor by which the performance of a system is measured. Although the execution time depends on the specifications of the machine on which the tool runs, it can be considered as a good criterion if both approaches are run on the same machine.

- **Number of "Reset"** gives the reader a taste for the number of wrong decisions made by each approach.

- **Cost** of reconstructing the user-interactions is the last factor we have chosen to measure. The reported cost is a combination of the number of actions executed for reconstructing the user-interactions and the actual cost of "Reset".

  The cost of "Reset" is determined based on the number of potential executable actions during time of performing "Reset". The cost of "Reset" might be expensive and it depends on how target application has been designed.

  Consequently, for each of our test cases we have measured two factors based on which we assign a cost to "Reset" in that application [9].

– $\mathbf{t_{(e)avg}}$, is the average time of element execution. This value is measured by randomly picking HTML elements from the application and taking the average time of executing them.

– $\mathbf{t_{(r)avg}}$, is the average time for performing a "Reset". We measure the value of this variable by loading the initial URL of the Web application multiple times to take the average time of reloads.

Therefore, the final cost of "Reset" for each Web application is determined through

$$C_{(\mathrm{r})\mathrm{avg}} = \frac{t_{(\mathrm{r})\mathrm{avg}}}{t_{(\mathrm{e})\mathrm{avg}}}$$

Finally, the cost of reconstructing the user-interactions for each Web application is measured through

$$C = n_{(\mathrm{e})} + (n_{(\mathrm{r})} * C_{(\mathrm{r})\mathrm{avg}})$$

where $\mathrm{n}_{(\mathrm{e})}$ and $\mathrm{n}_{(\mathrm{r})}$ are the total number of executed elements and performed "Resets" respectively.

As discussed in chapter 4, we use Fiddler to capture HTTP network traffic which is the only input given to IR. Consequently, we need to have Fiddler installed on a Windows-based computer. In order to use Fiddler without security warnings, Fiddlers root certificate is installed automatically in the Trusted Certificates store of either the system or the Web browser [32]. We ask the members of our research group to browse each of our target Web applications in order to provide IR with input.

Our users have been requested to use Google Chrome[1] in place of the Web browser by which they browse the target Web applications. The caching behavior of Web browsers might be different. Consequently, we select Google Chrome because it uses the same search engine as PhantomJS which is the main core of IR-Browser. This way the generated HTTP network traffic will be more similar to the one which IR-Browser triggers while replaying the captured HTTP trace in terms of caching behavior.

Our proposed user-interactions reconstruction system has been implemented in two languages. To implement IR-Browser we have used JavaScript since our headless PhantomJS

[1] `http://www.google.com/chrome/`

browser has been developed in JavaScript. Moreover, JavaScript allows us to modify any aspect of the target Web application. On the other side, IR-Proxy has been implemented as a PHP application.

IR is run on a Linux-based computer with an Intel® Core™i7 CPU at 2.7 GHz and 4GB of RAM. The output of our implemented user-interactions reconstruction tool would be as follows;

- the **DOM** of all the pages which the user had visited during the user-session.
- the **XPath** of all the elements executed by the user during the captured user-session.
- the **Screen Shot** of all the DOMs visited by the user during the user-session
- the **Inputs** provided by the user during the previously captured user-session

## 6.2 Test Cases

This section introduces Web applications which we use as our test cases in our experiments. Two of our selected Web applications have been implemented and maintained by our research group. The rest of our target applications are instances of the real world websites.

We do not limit our test cases to RIAs since there are already tools [44] which can perform user-interactions reconstruction on traditional Web applications. Therefore, we also run our user-interactions reconstruction tool on non-AJAX Web applications in order to show the effectiveness of our proposed solution on traditional Web applications.

Breton et al. [33] introduces Web applications which have been designed or selected for testing purposes. We have chosen three of their suggested Web applications, OpenCard, osCommerce and Joomla, to run our user-interactions reconstruction tool on the real Web applications which have been suggested to be used for testing automated tools.

The rest of this section is written to introduce each of our test cases separately. This way we give the reader a taste for the factors which will be measured with each of our test cases.

- **TestRia**[2] is one of the Web applications designed by our research group. TestRIA is an example of a fully AJAX-based E-commerce website. There is a single URL associated with all the pages visited by a user. Users can select different menu items on each page and the contents will be fetched via AJAX calls interactively.

  There is no user-input involved in the website and the user is not able to edit any part of the Web application. There is no random value assigned to the parameters sent to the Web server and the event handlers are not assigned through the "addEventListener" method.

- **Periodic Table**[3] is a Web application which has been designed as a large and dense graph. Periodic Table is also an example of a fully AJAX-based website in which a single URL is associated to all the pages visited by a user. The user chooses a chemical element of the table and the detailed information about the selected element will be fetched via an AJAX call interactively.

  There is no user-input involved in the website. The user is provided with two views of the Web application which is switched by JavaScript. There is no random value assigned to the parameters sent to the Web server.

  Periodic Table is an example of a Web application in which the values of the parameters sent to the Web server are generated on the fly. In other words the values are determined by executing JavaScript functions assigned to the "onClick" attribute of actionable HTML elements.

- **elFinder**[4] is an open-source AJAX-based file manager for the Web designed in JavaScript. User can browse the files and folders via the tree-view provided on the left pane [42].

  Although there is no user-input involved in the website, there is a parameter which will be sent to the Web server after executing any of the HTML elements of the Web application. The value of this parameter is assigned by a JavaScript function which has been attached to the elements. Therefore, client-side non-determinism happens in elFinder and we show how our proposed heuristic identifies this non-determinism.

[2] `http://ssrg.site.uottawa.ca/TestRIA/`

[3] `http://ssrg.site.uottawa.ca/apr5/success1/`

[4] `http://elfinder.org`

elFinder is an example of a Web application in which event handler are attached to the HTML elements through a call to the "addEventListener" method.

- **Altoro Mutual**[5] is a demo website for a fictional bank and is maintained by the IBM® AppScan® team for security testing. The original website[6] is a traditional web application which uses hyper-links for users to navigate. Our research group has created an AJAX-based version of the website that makes AJAX calls to the Web server when a user interacts with the Web application [42].

  Altoro Mutual contains different user-inputs through which a user is able to modify her profile, perform transactions etc. Also, the administrator of the website can modify the list of users.

  There is no random value assigned to the parameters sent to the Web server and the JavaScript functions are assigned as a value to the "href" attribute of each element.

- **osCommerce**[7] is a secure leading Open-Source e-commerce platform. It is a demo of a real RIA in which the user can search among products, change the currency of a product's price, and add the products to the cart in order to buy the selected ones.

  There are different user-inputs involved in osCommerce by which the user has more control of the Web application. There is no random value assigned to the parameters sent to the Web server and the event handlers are assigned through a call to the "addEventListener" method.

- **arXiv**[8] is a repository of electronic scientific papers in the field of science. In various fields of mathematics and physics, almost all the papers are archived in arXiv. The website is a traditional Web application featuring hyper-links for navigating by users.

  There are user-inputs by which the user can search among scientific papers, log in to the Web application, etc. There is no random value assigned to the parameters sent to the Web server and there is no call to the "addEventListener" method.

[5] Local version is used

[6] `http://altoromutual.com/`

[7] `http://demo.oscommerce.com/`

[8] `http://arxiv.org/`

- **OpenCart**[9] is an Open-Source PHP-based online shopping cart platform. In this RIA the user can create her own account, search among available products, add a product to the cart, buy her selected products, etc. OpenCart has been designed to be feature-rich and easy to use.

  There are user-inputs involved in the Web application but there is no random value assigned to the parameters sent to the Web server. The event handlers are attached to the elements through a call to the "addEventListener" method.

- **Joomla**[10] is a free, Open-Source and PHP-based content management system, used for publishing Web content. It enables users to build websites and on-line applications. The extensions of Joomla, components, modules, plug-ins, templates, and languages extend the functionality of the websites.

  There are many user-inputs by which the user has more control of the Web application. There is no random value assigned to the parameters sent to the Web server and there is no call to the "addEventListener" method.

## 6.3 Comparison

In this section we provide the experimental results of our proposed solution against the "Basic Solution" which refers to a system that reconstructs user-interactions by an in-depth search on all the DOM elements. Such a system should handle user input recovery, client-side non-determinism, as well as performing "SequenceCheck" and "Reset". The results will be given in two tables and eight charts.

Table 6.1 reports the factors which specify the specifications of the captured HTTP network traffic for each of the Web applications.

- **Actions** refers to the number of actions done by a user during the user-session from which user-interactions are going to be reconstructed.

- **Original db** refers to the number of HTTP requests generated by the user's Web browser during the user-session.

[9] `http://demo.opencart.com/`

[10] `https://www.joomla.org/`

- **Loaded db** specifies the number of HTTP requests which have been loaded into the IR-Proxy's database. As discussed in chapter 5 IR-Proxy does not load all the generated HTTP messages.

  The number of generated HTTP requests by the user's Web browser is equal to the number of generated HTTP responses sent by the Web server. The number which is reported in **"Original db"** and **"Loaded db"** will be

$$\frac{The Number of Exchanged HTTP Messages}{2}$$

- **User Inputs** refers to the number of field elements which were filled by a user during the captured user-session.

- **Cost of Reset** is measured for each of the target Web applications using the formula explained in section 6.1.

- **Delay** refers to the time which IR-Browser waits after performing each action in order to make sure that the content of the current DOM of the Web application has been loaded properly. This factor is reported in milliseconds.

| | Actions | Original db | Loaded db | User Inputs | Cost of Reset | Delay |
|---|---|---|---|---|---|---|
| TestRia | 30 | 74 | 71 | 0 | 2 | 50 |
| Periodic Table | 50 | 55 | 54 | 0 | 8 | 50 |
| elFinder | 30 | 57 | 55 | 0 | 10 | 50 |
| Altoro Mutual | 100 | 133 | 132 | 12 | 2 | 50 |
| osCommerce | 100 | 452 | 191 | 19 | 2 | 50 |
| arXiv | 100 | 104 | 103 | 0 | 2 | 50 |
| Open Card | 100 | 203 | 192 | 57 | 5 | 50 |
| Joomla | 100 | 180 | 178 | 459 | 6 | 5000 |

Table 6.1: Specifications of the Captured HTTP Network Traffic

Table 6.1 helps the reader to not only have a general view of the target Web application but also get a detailed information about the captured HTTP network traffic for each test case.

The first three specified Web applications "TestRia", "Periodic Table" and "elFinder" are small Web applications in which the user cannot interact with the Web application for long and get new data. Consequently, the number of performed actions is less than the number of actions performed for our other test cases.

The number of HTTP requests dropped by IR-Proxy is high in the case of "osCommerce". The reason is that this website uses Google Analytics[11], as well as being connected to different advertisement websites. Consequently, there is an HTTP request corresponding to Google Analytics in order to gather statistical data, by executing any action within the Web application. Therefore, there are many HTTP requests which are dropped by IR-Proxy while loading the HTTP trace captured from "osCommerce".

In the case of "Joomla" the amount of specified delay is high compared to our other test cases. IR-Browser needs to wait after executing each action to make sure the DOM has been rendered properly. Otherwise, it may not have access to some parts of the DOM and this can cause IR to fail in reconstructing user-interactions.

"Joomla" is a big Web application with heavy JavaScript functions. As discussed in chapter 2 the Web browser should execute the JavaScript functions which are supposed to be executed once the DOM starts loading. Consequently, IR-Browser should wait for the heavy functions to be executed.

Moreover, there is a timeout event, specified in most of the pages, in order to check whether or not the current installed Web application needs to be updated. This is determined by generating an automated AJAX call to the Web server. Therefore, IR-Browser has to wait until this AJAX call is triggered.

Table 6.2 specifies factors which determine how efficient each solution works. The table contains the results of running IR on all our test cases using each of the mentioned solutions.

- **Elements** specifies the average number of HTML elements which are considered actionable while reconstructing user-interactions.
- **Control Messages** specifies the number of control messages exchanged between IR-Browser and IR-Proxy while reconstructing user-interactions. To have a concrete

[11] `http://www.google.com/analytics/`

analysis we have broken this factor into three different control messages which are triggered.

1. **UI** determines the number of messages triggered by IR-Browser in order to check whether or not the next unconsumed HTTP request corresponds to the user inputs.

2. **NR** determines the number of messages triggered by IR-Browser for getting the next unconsumed HTTP request in order to prioritize the list of actionable HTML elements which should be executed.

3. **SC** determines the number of "SequenceCheck" messages triggered by IR-Browser in order to check whether or not it has executed the correct action.

- **Time** specifies the execution time (hh:mm:ss) in which IR has reconstructed the entirety of the user-interactions.

- **Reset** reports the number of "Resets" performed by IR while reconstructing the user-interactions.

- **Cost** determines the cost of reconstructing the user-interactions as discussed in section 6.1

As table 6.2 shows our proposed solution is successful in reconstructing user-interactions for all our test cases. Moreover, our improvements have a significant impact on both the execution time and the cost of reconstructing user-interactions.

In the case of "Joomla" as shown in table 6.2 the execution time is high even for our proposed solution. It is more surprising when we notice that the execution time is high even when there is no "Reset" performed by our proposed solution. The specified delay mentioned in table 6.1 tells us the reason behind this result.

The number of control messages exchanged between IR-Browser and IR-Proxy is also a reason based on which the execution time of reconstructing user-interactions is high for "Joomla" . As table 6.2 shows the number of messages triggered to ask about user inputs submission is considerable. There are many field elements in this Web application and IR

| | | Elements | Control Messages | | | Time | Reset | Cost |
|---|---|---|---|---|---|---|---|---|
| | | | UI | NR | SC | | | |
| TestRia | Our Solution | 10 | 0 | 29 | 86 | 00:01:43 | 28 | 724 |
| | Basic Solution | 49 | 0 | 0 | 1,888 | 00:20:25 | 929 | 17,670 |
| Periodic Table | Our Solution | 129 | 0 | 49 | 4,806 | 02:34:57 | 2,378 | 90,984 |
| | Basic Solution | 667 | 0 | 0 | 26,366 | 16:00:04 | 13,158 | 507,037 |
| elFinder | Our Solution | 17 | 0 | 29 | 390 | 00:09:53 | 180 | 4,562 |
| | Basic Solution | 89 | 0 | 0 | 2,242 | 00:27:46 | 1,106 | 27,835 |
| Altoro Mutual | Our Solution | 38 | 266 | 93 | 106 | 00:01:13 | 3 | 315 |
| | Basic Solution | 117 | 266 | 0 | 14,648 | 14:26:17 | 7,274 | 372,486 |
| osCommerce | Our Solution | 68 | 594 | 82 | 100 | 00:01:07 | 0 | 100 |
| | Basic Solution | 224 | 594 | 0 | 15,032 | 17:10:45 | 7,466 | 411,870 |
| arXiv | Our Solution | 202 | 321 | 99 | 100 | 00:01:17 | 0 | 100 |
| | Basic Solution | 387 | 321 | 0 | 19,582 | 19:26:55 | 9,741 | 484,724 |
| Open Card | Our Solution | 95 | 421 | 82 | 132 | 00:03:43 | 16 | 922 |
| | Basic Solution | 288 | 421 | 0 | 27,472 | 31:59:39 | 13,686 | 799,108 |
| Joomla | Our Solution | 116 | 21,148 | 67 | 100 | 00:27:45 | 0 | 100 |
| | Basic Solution | 655 | 21,148 | 0 | 33,848 | 377:34:18 | 16,874 | 839,610 |

Table 6.2: Comparison Between "Basic Solution" and Our Proposed Solution

needs to check whether or not each of them should be submitted. Therefore, most of the time the two main components of IR are communicating to figure out what should be done at each step.

The following charts show the effectiveness of our proposed solution. We report a chart per each test case in order to put our solution against the "Basic Solution".

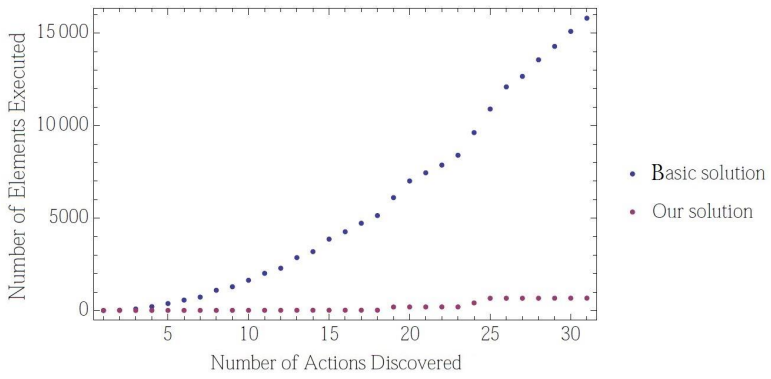


Figure 6.1: TestRia: Discovered Actions vs Elements Executed

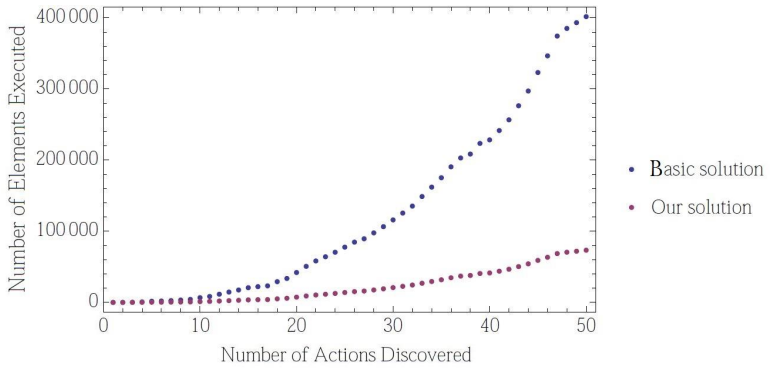


Figure 6.2: Periodic Table: Discovered Actions vs Elements Executed

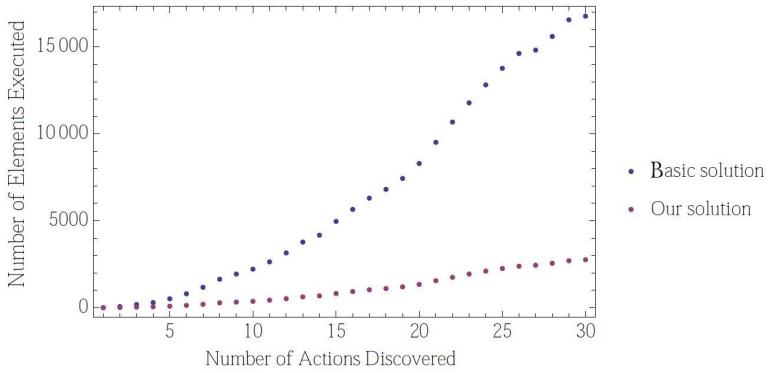


Figure 6.3: elFinder: Discovered Actions vs Elements Executed

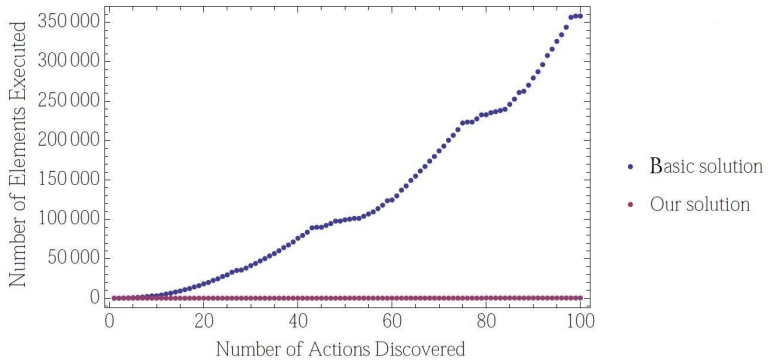


Figure 6.4: Altoro Mutual: Discovered Actions vs Elements Executed

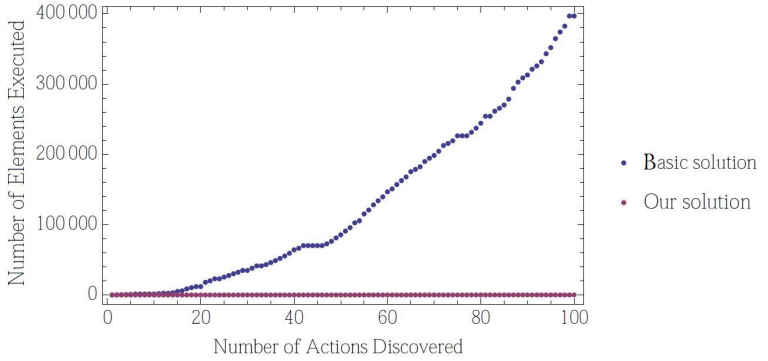


Figure 6.5: osCommerce: Discovered Actions vs Elements Executed

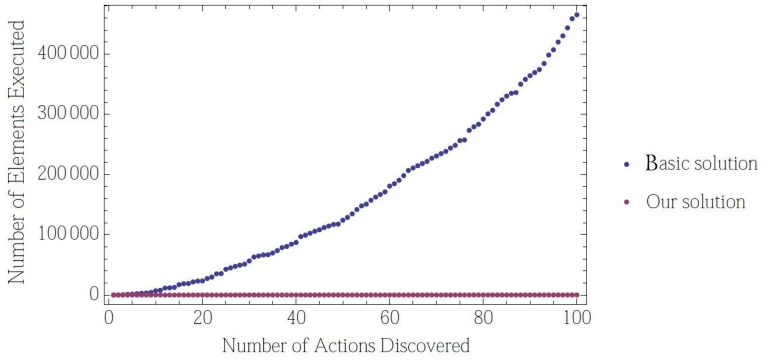


Figure 6.6: arXiv: Discovered Actions vs Elements Executed

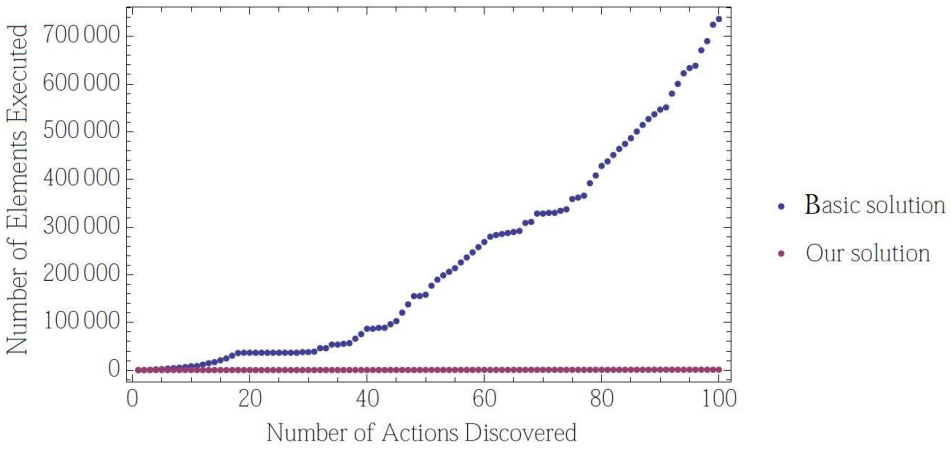


Figure 6.7: Open Card: Discovered Actions vs Elements Executed

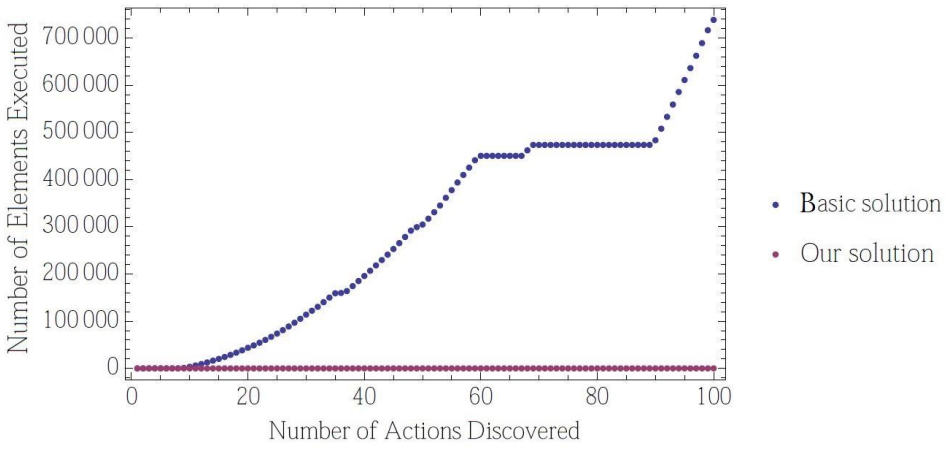


Figure 6.8: Joomla: Discovered Actions vs Elements Executed

- **Case study:** In order to demonstrate the capabilities of our implemented tool, we have captured a sample scenario accompanied by a video which shows the act of the hacker, as well as using the tool to reveal her acts. The hacker visits AltoroMutual which is one of our test cases and does some illegal actions because of the weak security design of the Web application. We then use our implemented tool in order to construct the user-interactions in relatively short amount of time[12] [3].

## 6.4 Summary

In this chapter we presented the experimental results of our proposed solutions to show how effective our approach is. We introduced our test cases, as well as explaining the experimental setups to have our implemented tool work.

We conclude our work along with describing the drawbacks of our proposed solutions in the next chapter.

[12] A demonstration of this case study, as well as sample inputs/outputs of the tool can be found on `http://ssrg.site.uottawa.ca/sr/demo.html`

# Chapter 7

# Conclusion

In this research we provided a way by which user-interactions can be constructed from an unaltered HTTP network traffic file. We proposed replaying the captured HTTP trace in order to identify what the user did during the given user-session.

Our implemented tool can be used in various situations such as forensic analysis. We reconstruct user-interactions without accessing the RIA website and this minimizes security concerns. The Web server will not be re-attacked by replaying the captured HTTP traffic using our approach. Moreover, we rely on previously captured HTTP network traffic and do not need to manipulate the Web application which means that our tool can work on user-sessions which have already been passed [3].

Our tool also can be used to retrace all the actions done by the user before reaching the problematic state of the Web application. Therefore, we can automatically reproduce the bug report.

We explained our improvements on the existing framework in order to have an efficient user-interactions reconstruction system.

Replaying user inputs is the first step which has to be handled. Otherwise, the entire process will fail. Client-side non-determinism is another factor which can stop the process of reconstructing the user-interactions if it is not handled. We explained how we have addressed these challenges, as well as describing our optimizations to find the correct user-action efficiently.

## 7.1 Improvements to Suggested Solutions

Our proposed solutions along with our implemented tool have some shortcomings two of which have been addressed in [3].

- **Client-side non-determinism:** We proposed a way by which our tool can detect client-side non-determinism within the target Web application. Our solution is to re-execute the same action in order to compare the generated set of HTTP requests in two sequential execution of an HTML element. This way IR has to perform "Reset" every time that there is a possibility of involving client-side non-determinism. Therefore, the cost of exploration increases.

  The authors of [3] suggest running two instances of IR-Browser at the same time. Consequently, the generated set of HTTP requests by each instance of IR-Browser can be compared after executing the same action. This way the cost of exploration will not change, since the process is done in parallel and the client-side non-determinism can be identified at the time of executing each action.

- **Timeout event:** We mentioned that our instrumented Web browser needs to wait after receiving the response of all its HTTP requests to make sure that the DOM has been loaded properly. Moreover, we need to identify if there will be an automated AJAX call to the Web server after a specific time. In such a case we increase the idle time after executing each action as we did for "Joomla", one of our test cases, in chapter 6.

  The authors of [3] suggest overwriting the "setTimeout" method within the Web browser in order to determine how much it takes for the AJAX call to be triggered. Therefore, the instrumented browser waits no longer than it should.

- **Dropping out-of-Domain HTTP Requests:** Our implemented tool filters out all the HTTP requests corresponding to domains different from the domain of the initial URL. There are Web applications hosted in more than one Web server. Stack overflow[1] is an example of such a Web application, the corresponding JavaScript functions of which are hosted in "static.adzerk.net". Consequently, our instrumented

[1] urlhttp://stackoverflow.com/

Web browser fails in loading such a Web application, since IR-Proxy does not have the corresponding response to the HTTP requests for fetching JavaScript functions.

One solution could be having a black list of URLs which are intended to either be the requests for fetching advertisements or that go to "Google Analytics". As authors of ClickMiner [44] have suggested, AdBlock Plus[2] can be used in order to create such a black list of URLs. Consequently, IR will not drop the HTTP requests which are going to fetch important resources from the Web server.

## 7.2 Future Work

There are some opportunities to improve our work.

- **User Inputs:** It is a common behavior in RIAs to have a JavaScript function attached to a form element. The event handler will be executed upon submitting the corresponding HTML form. Consequently, there would be parameters which are added to the generated HTTP request on the fly. Google search is an example of such a form element. Figure 7.1 shows the HTTP request triggered by a user's Web browser after searching for a query. The parameters "biw", "bih", "gs_l", "pbx", "bav", "bvm", "fp", "tch", "ech", and "psi" are added to the query string by executing a JavaScript function attached to the corresponding form element.

```
GET "https://www.google.ca/search?sclient=psy-ab&biw=1600&bih=425&q=ssrg+uottawa&oq=
    ssrg+uottawa&gs_l=hp.12..0.10821.15182.0.24801.14.8.0.6.6.0.516.1219.2j5j5
    -1.8.0....0...1c.1.64.psy-ab..2.12.1109.0.hnC6qoYRUxk&pbx=1&bav=on.2,or.&bvm=bv
    .107467506,d.eWE&fp=4f0dd3b5f5b7a6d4&tch=1&ech=1&psi=T9dGVt3nC8bgmwHwmqpY
    .1447483219755.3" HTTP/1.1
Host: www.google.ca
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
     Chrome/46.0.2490.80 Safari/537.36
Referer: "https://www.google.ca/"
```

Figure 7.1: HTTP request triggered after submitting an HTML form with attached handler

[2] https://adblockplus.org/

Moreover, there are Web applications built on JavaScript frameworks, such as DoJo and AngularJS, in which the format of submitting field elements has been defined differently from the standard format used in HTTP.

Regarding the way in which our tool determines the submission of user inputs, it fails to replay the captured HTTP trace in above-mentioned Web applications.

- **Find User-Actions Efficiently:** In RIAs most of the generated HTTP network traffic contains data which has been processed by JavaScript functions on the client-side. Consequently, looking at the HTTP message exchanged between a Web browser and a Web server does not provide enough information. Figure 7.2 shows an example of an HTTP request triggered by user's Web browser from "https://mail.google.com".

```
GET "https://mail.google.com/mail/u/0/channel/bind?VER=8&at=
    AF6bupPHaJvL4DyeY8s9GS_lE4Q1xzY6ag&it=1603827&RID=rpc&SID=11248E9618E9E970&CI=0&
    AID=207&TYPE=xmlhttp&zx=1k738kizilxq&t=1" HTTP/1.1
Host: mail.google.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
     Chrome/46.0.2490.80 Safari/537.36
Referer: "https://mail.google.com/_/scs/mail-static/_/js/k=gmail.main.en.n1WzaaOs8ws.O
    /m=m_i,t/am=PiPeQMA9GfcHcY1xQLP0FQp77z8_u0jxkYMX9SdMAJF1AfB_s_8H8HvQ3rIFBg/rt=h/d
    =1/t=zcms/rs=AHGWq9CJXd9uUKVcIFdPzpIonvG0Jzr0fg"
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

Figure 7.2: HTTP request triggered by a Web browser loading a RIA

Regarding the way in which our tool prioritizes HTML elements, it cannot make a distinction between elements in such an HTTP message. Consequently, the process of constructing the user-interactions cannot be done efficiently.

- **Cache of the Web Browser:** All Web browsers do not work in the same way, since they have their own implementation. Consequently the caching behavior of Web browsers can differ from one Web browser to another. Therefore, the generated set of HTTP requests after executing an action can be different, since one Web browser may use the cached version of some resources while another one wants to fetch those resources from the Web server.

Regarding the way in which our implemented tool decides on the correctness of the executed action, it fails to construct user-interactions when the caching behavior of our instrumented Web browser is not the same as the one used by the user during the user-session.

- **Client-Side Actions:** It is a common behavior in RIAs to have an HTML element to which there is an event handler attached in order to update the state of the DOM without communicating with the Web server. Consequently, there will not be any HTTP request triggered by executing such an action on the client-side.

  Regarding the way by which we construct user-interactions, our implemented tool cannot identify client-side actions. Moreover, our approach fails if the next user-action should be picked up from the updated part of the DOM, since our solution misses updating the DOM by executing client-side actions.

- **Different Web Browsers:** PhantomJS is the main core of our instrumented Web browser. Consequently, we are limited to our headless Web browser to replay the given user-session. There are Web applications that are designed differently with regard to the Web browser which renders the Web application. Moreover, not all the Web browsers support all defined features of HTML or scripting languages. Consequently, it is probable for Web browsers to request a particular resource in different ways.

  Regarding the way in which our implemented tool checks the correctness of a triggered set of HTTP requests, it fails to construct user-interactions for the Web applications which use features not supported by PhantomJS.

# References


[1] Mozilla crash reporter. Technical report, Firefox Help, 2015. `https://support.mozilla.org/en-US/kb/mozillacrashreporter`, visited 2015-12-07.

[2] S. Andrica and G. Candea. WaRR: A tool for high-fidelity web application record and replay. In *IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pages 403–410. IEEE, June 2011.

[3] Sara Baghbanzadeh, Salman Hooshmand, Gregor v. Bochmann, Guy-Vincent Jourdan, Seyed M. Mirtaheri, Muhammad Faheem, and Iosif Viorel Onut. Forenria: The reconstruction of user-interactions from http traces for rich internet applications. In *12th Annual IFIP WG 11.9 International Conference on Digital Forensics, New Dehli, India*, 2015.

[4] Bettina Berendt, Bamshad Mobasher, Myra Spiliopoulou, and Jim Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. In *Workshop on Web Mining at the First SIAM International Conference on Data Mining*, pages 7–14, 2001.

[5] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal. Adaptive neuro-fuzzy intrusion detection systems. In *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004*, volume 1, pages 70–74, April 2004.

[6] Jim Conallen. *Building Web Applications with Uml*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. ISBN 0-201-73038-3.

[7] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns. *Knowledge and Information Systems*, 1(1):5–32, July 2013.

[8] E. Dincturk, G.-V. Jourdan, G. V. Bochmann, and V. Onut. A model-based approach for crawling rich internet applications. In *ACM Transactions on the Web (TWEB), 8(3)*, 2014.

[9] Mustafa Emre Dincturk. *Model-based Crawling - An Approach to Design Efficient Crawling Strategies for Rich Internet Applications*. PhD thesis, University of Ottawa, Ottawa, Canada, 2013. `https://www.ruor.uottawa.ca/bitstream/10393/24375/3/Dincturk_Mustafa_Emre_2013_thesis.pdf`, visited 2015-12-07.

[10] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A Scalable Comparison-shopping Agent for the World-Wide Web. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pages 39–48, New York, NY, USA, 1997. ACM.

[11] Fabien Duchene, Roland Groz, Sanjay Rawat, and Jean-Luc Richier. XSS Vulnerability Detection Using Model Inference Assisted Evolutionary Fuzzing. In *SECTEST 2012-3rd International Workshop on Security Testing (affiliated with ICST)*, pages 815–817. IEEE Computer Society, April 2012.

[12] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, Marc Fisher, et al. Leveraging user-session data to support web application testing. *Software Engineering, IEEE Transactions on*, 31(3):187–202, 2005.

[13] K. Etminani, A.R. Delui, N.R. Yanehsari, and M. Rouhani. Web usage mining: Discovery of the users' navigational patterns using SOM. In *First International Conference on Networked Digital Technologies. NDT '09*, pages 224–249, July 2009.

[14] Jonathan Fielding. Optimizing your responsive site. In *Beginning Responsive Web Design with HTML5 and CSS3*. Apress, 2014. `http://link.springer.com/chapter/10.1007/978-1-4302-6695-2_10`, visited 2015-12-07.

[15] David Flanagan. *JavaScript: The Definitive Guide*. O'Reilly, 2011. ISBN 978-0-594-80552-4.

[16] Rob Friesel. *PhantomJS Cookbook*. Packt Publishing Ltd, 2014. ISBN 978-1-78398-192-2.

[17] Jesse James Garrett. Ajax: A new approach to web applications. February 2005. http://adaptivepath.org/ideas/ajax-new-approach-web-applications/, visited 2015-12-07.

[18] Kirk Glerum, Kinshuman Kinshumann, Steve Greenberg, Gabriel Aul, Vince Orgovan, Greg Nichols, David Grant, Gretchen Loihle, and Galen Hunt. Debugging in the (Very) Large: Ten Years of Implementation and Experience. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 103–116, New York, NY, USA, 2009. ACM.

[19] Achyut S. Godbole. *Data Communications and Networks*. Tata McGraw-Hill Publishing Company Limited, 7th edition, 2007. ISBN 0-07-047297-1.

[20] Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, and Edoardo M. Airoldi. A Survey of Statistical Network Models. *Found. Trends Mach. Learn.*, 2(2):129–233, February 2010.

[21] David Gourley, Brian Totty with Marjorie Sayer, Sailu Reddy, and Anshu Aggarwal. *HTTP: The Definitive Guide*. O'Reilly, 2002. ISBN 978-1-56592-509-0.

[22] M. Grill and M. Rehak. Malware detection using HTTP user-agent discrepancy identification. In *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 221–226, December 2014.

[23] Seung-Sun Hong and Shyhtsun Felix Wu. On interactive internet traffic replay. In *Recent Advances in Intrusion Detection (RAID)*, volume 3858, pages 247–264. Springer, 2005.

[24] Pieter Hooimeijer and Westley Weimer. Modeling Bug Report Quality. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 34–43, New York, NY, USA, 2007. ACM.

[25] Jeffrey C. Jackson. *Web Technologies: a computer science perspective*. ISBN 978-0-13185-603-5.

[26] Jerald B. Johnson and Kristian S. Omland. Model selection in ecology and evolution. *Trends in Ecology & Evolution*, 19(2):101–108, February 2004.

[27] Guy-Vincent Jourdan. WWW structures, techniques and standards. University Lecture, University of Ottawa, `http://www.site.uottawa.ca/~gvj/Courses/CSI3140/lectures/WebEssentials.pdf`, visited 2015-12-07, 2010.

[28] Jeremy Keith. *DOM Scripting: Web Design with JavaScript and the Document Object Model*. Springer-Verlag New York Inc., 2005. ISBN 1-59059-533-5.

[29] N. Kiatkumjounwong, S. Ngamsuriyaroj, A. Plangprasopchok, and A. Hoonlor. Analysis and classification of web proxy logs based on patterns of traffic rates. In *TENCON - IEEE Region 10 Conference*, pages 1–5, October 2014.

[30] Tammo Krueger, Christian Gehl, Konrad Rieck, and Pavel Laskov. TokDoc: A Self-healing Web Application Firewall. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1846–1853, New York, NY, USA, 2010. ACM.

[31] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Rec.*, 31(2):84–93, June 2002.

[32] Eric Lawrence. *Debugging with Fiddler: The complete reference from the creator of the Fiddler Web Debugger*. CreateSpace Independent Publishing Platform, 2015.

[33] G. Le Breton, N. Bergeron, and S. Halle. A Reference Framework for the Automated Exploration of Web Applications. In *2014 19th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 81–90. IEEE, August 2014.

[34] John Zhong Lei and Ali Ghorbani. The Reconstruction of the Interleaved Sessions from a Server Log. In Ahmed Y. Tawfik and Scott D. Goodwin, editors, *Advances in Artificial Intelligence*, number 3060 in Lecture Notes in Computer Science, pages 133–145. Springer Berlin Heidelberg, 2004.

[35] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, January 2013.

[36] Stephen W. Liddle, David W. Embley, Del T. Scott, and Sai Ho Yau. Extracting Data behind Web Forms. In Antoni Oliv, Masatoshi Yoshikawa, and Eric S. K. Yu, editors, *Advanced Conceptual Modeling Techniques*, number 2784 in Lecture Notes in Computer Science, pages 402–413. Springer, October 2003.

[37] Xinghua Lou, M. Schiegg, and F.A. Hamprecht. Active Structured Learning for Cell Tracking: Algorithm, Framework, and Usability. *IEEE Transactions on Medical Imaging*, 33(4):849–860, April 2014.

[38] Jayant Madhavan, David Ko, \Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's Deep Web Crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, August 2008.

[39] James Mickens, Jeremy Elson, and Jon Howell. Mugshot: Deterministic capture and replay for javascript applications. In *NSDI*, volume 10, pages 159–174, Berkeley, CA, USA, 2010. USENIX Association.

[40] Seyed M. Mirtaheri, G.v. Bochmann, G.-V. Jourdan, and I.V. Onut. Pdist-ria crawler: A peer-to-peer distributed crawler for rich internet applications. In *Proceedings of the WISE*, 2014.

[41] A. Moosavi, S. Hooshmand, S. Baghbanzadeh, G.-V. Jourdan, and I.V Onut. Indexing rich internet applications using components-based crawling. In *Proceedings of the ICWE 2014, Toulouse, France. 18 pages*, 2014.

[42] Ali Moosavi. Component-based crawling of complex rich internet applications. Master's thesis, University of Ottawa, Ottawa, Canada, 2013. `http://www.ruor.uottawa.ca/bitstream/10393/30636/3/Moosavi_Byooki_Seyed_Ali_2014_thesis.pdf`, visited 2015-12-07.

[43] M. Nadjarbashi-Noghani and A.A. Ghorbani. Improving the referrer-based Web log session reconstruction. In *Second Annual Conference on Communication Networks and Services Research, 2004. Proceedings*, pages 286–292. IEEE, May 2004.

[44] Christopher Neasbitt, Roberto Perdisci, Kang Li, and Terry Nelms. ClickMiner: Towards Forensic Reconstruction of User-Browser Interactions from Network Traces. In

*Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1244–1255, New York, NY, USA, 2014. ACM.

[45] Bill Pea Nick Heinle. *Designing with Javascript: Creating Dynamic Web Pages*. O'Reilly, 2nd edition, 2002. ISBN 1-56592-360-X.

[46] I.J. Nino, B. de la Ossa, J.A. Gil, J. Sahuquillo, and A. Pont. CARENA: a tool to capture and replay Web navigation sessions. In *Workshop on End-to-End Monitoring Techniques and Services*, pages 127–141. IEEE, May 2005.

[47] Benjamin Livshits Paruj Ratanaworabhan and Benjamin G. Zorn. Jsmeter: Comparing the behavior of javascript benchmarks with realweb applications. In *USENIX Conference on Web Application Development (WebApps)*, June 2010.

[48] K. Pattabiraman and B. Zorn. DoDOM: Leveraging DOM Invariants for Web 2.0 Application Robustness Testing. In *2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, pages 191–200, November 2010.

[49] Linda Dailey Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

[50] Roberto Perdisci, Davide Ariu, and Giorgio Giacinto. Scalable fine-grained behavioral clustering of HTTP-based malware. *Computer Networks*, 57(2):487–500, February 2013.

[51] Emmanuel S. Pilli, R. C. Joshi, and Rajdeep Niyogi. Network forensic frameworks: Survey and research challenges. *Digital Investigation*, 7(12):14–27, October 2010.

[52] Dragos-Paul Pop and Adam Altar. A Model For The Automation Of Html Form Creation And Validation. *Journal of Information Systems & Operations Management*, 6(1):119–130, 2012.

[53] Mahendra Pratap, Singh Dohare, Premnarayan Arya, and Arun Bajpai. Novel web usage mining for web mining techniques. *International Journal of Emerging Technology and Advanced Engineering*, 2(1):253–262, 2012.

[54] Sriram Raghavan and Hector Garcia-Molina. Crawling the Hidden Web. In *Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy*. Morgan Kaufmann, 2001.

[55] M. Rebhan, V. Chalifa-Caspi, J. Prilusky, and D. Lancet. GeneCards: a novel functional genomics compendium with automated data mining and query reformulation support. *Bioinformatics*, 14(8):656–664, January 1998.

[56] Wei Ren. Modeling Network Forensics Behavior. *Journal of Digital Forensic Practice*, 1(1):57–65, March 2006.

[57] William O. Scheeren. *The Hidden Web: A Sourcebook*. Libraries Unlimited, 2012. ISBN 978-1-59884-627-0.

[58] Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs. Jalangi: A selective record-replay and dynamic analysis framework for javascript. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 488–498, New York, NY, USA, 2013. ACM.

[59] Chris Shiflett. *HTTP Developer's Handbook*. Sams Publishing, 1st edition, 2003. ISBN 0-672-32454-7.

[60] V. Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed Web crawler. In *Proceedings of the 18th International Conference on Data Engineering*, pages 357–368, 2002.

[61] Steve Souders. High-performance Web Sites. *Commun. ACM*, 51(12):36–41, December 2008.

[62] Myra Spiliopoulou and Lukas C. Faulstich. WUM: A Tool for Web Utilization Analysis. In Paolo Atzeni, Alberto Mendelzon, and Giansalvatore Mecca, editors, *The World Wide Web and Databases*, number 1590 in Lecture Notes in Computer Science, pages 184–203. Springer Berlin Heidelberg, March 1998.

[63] Myra Spiliopoulou, Bamshad Mobasher, Bettina Berendt, and Miki Nakagawa. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *Informs journal on computing*, 15(2):171–190, 2003.

[64] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. *SIGKDD Explor. Newsl.*, 1(2):12–23, January 2000.

[65] Seyed M. Mir Taheri. *Distributed Crawling of Rich Internet Applications*. PhD thesis, University of Ottawa, Ottawa, Canada, 2015. `https://www.ruor.uottawa.ca/bitstream/10393/32089/3/Mir_Taheri_Seyed_Mohammad_2015_thesis.pdf`, visited 2015-12-07.

[66] Martijn Terpstra. Whatsapp and privacy. Master's thesis, Radboud University Nijmegen, Netherlands, 2013. `http://cs.ru.nl/bachelorscripties/2013/Martijn_Terpstra___0814962___WhatsApp_and_privacy.pdf`, visited 2015-12-07.

[67] K. Vikram, Abhishek Prateek, and Benjamin Livshits. Ripley: Automatically Securing Web 2.0 Applications Through Replicated Execution. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 173–186, New York, NY, USA, 2009. ACM.

[68] K.-L. Wu, P.S. Yu, and A. Ballman. SpeedTracer: A Web usage mining and analysis tool. *IBM Systems Journal*, 37(1):89–105, 1998.

[69] Guowu Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Yaohui Jin. ReSurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference*, pages 1–9, May 2013.

[70] Nancy J. Yeager and Robert E. McGrath. *Web Server Technology*. Morgan Kaufmann Publishers, Inc., 1st edition, 1996. ISBN 1-55860-376-X.