A COMPARATIVE STUDY OF SECURITY VULNERABILITIES IN RESPONSIVE WEB DESIGN FRAMEWORKS

Pakorn Ieamsirinoppakun
pin14002@student.mdh.se

Examiner: Jan Carlson
Mälardalen University, Västerås, Sweden

Supervisor: Irfan Sljivo
Mälardalen University, Västerås, Sweden

June 10, 2015
Abstract

With a fast growing of Internet, users spent more time accessing websites through mobile devices. Traditional websites that are not designed to support multi screen sizes typically have a lower number of users accessing the website. Responsive Web Design (RWD) is a web design technique that solves this limitation of traditional websites. RWD can dynamically adapt and fluidly display a webpage in different sizes and resolution. Many RWD frameworks are available in the market to facilitate the responsive webpage layout. Currently, RWD frameworks are widely used and discussed on its features only, while security vulnerabilities are usually not considered.

In this thesis, we aim to study the security vulnerabilities of RWD frameworks to identify the most secure RWD framework. To contribute, we first select RWD frameworks that rely on JavaScript as JavaScript vulnerabilities are commonly used to attack a webpage. Then, we identify the common security vulnerability in JavaScript that RWD frameworks could enable. Next, we test selected frameworks based on the identified vulnerability. The results are evaluated and compared by using established security metrics and comparison criteria. Finally, we identify the most secure RWD framework from the considered RWD frameworks.
# Contents

1 Introduction
   1.1 Problem Formulation .............................................. 4
   1.2 Research Methodology ........................................... 5

2 Background
   2.1 JavaScript .......................................................... 6
   2.2 Responsive Web Design ........................................... 7
   2.3 Security Vulnerability .......................................... 8
      2.3.1 Persistent XSS .................................................. 9
      2.3.2 Non-Persistent XSS ........................................... 9
      2.3.3 DOM-based XSS ................................................. 10
      2.3.4 Sources ......................................................... 11
      2.3.5 Sinks ........................................................... 11
      2.3.6 An Example of Vulnerable Code in DOM-based XSS ....... 12
   2.4 Detection of Vulnerabilities in Web Applications ............. 13
      2.4.1 White Box Testing ............................................. 13
      2.4.2 Black Box Testing ............................................. 13
   2.5 DOM-based XSS Detection Tools ................................ 13
      2.5.1 Xenotix XSS Exploit Framework ............................... 13
      2.5.2 IronWASP ........................................................ 14
   2.6 Regular Expression ................................................ 14
   2.7 Line of Code ......................................................... 15

3 Method
   3.1 Revisiting the Research Subgoals ............................... 18
   3.2 Data Gathering ...................................................... 18
   3.3 Data Interpretation ................................................ 19
   3.4 Study Setup ........................................................ 20

4 Result
   4.1 The Result of Source Density ................................... 21
   4.2 The Result of Sink Density ...................................... 21
   4.3 Interpretation of the results ................................... 21
   4.4 Discussion .......................................................... 22

5 Related Work .......................................................... 23

6 Conclusion and Future Work .......................................... 24

References ............................................................. 27
List of Figures

1. Responsive Web Design site on Smartphone, Tablet and Desktop .......................... 7
2. Storing Malicious Script in Server ................................................................. 9
3. Stealing Credential Information .................................................................. 9
4. Attacker detects vulnerability ...................................................................... 10
5. A successful attack in Non-Persistent XSS ....................................................... 10
6. DOM-based XSS ......................................................................................... 11
7. OWASP Xenotix XSS Exploit Framework ......................................................... 14

List of Tables

1. Source Name & Property ........................................................................ 12
2. Sink Name & Property ........................................................................ 12
3. A Subset of Special Characters in Regular Expression ............................... 15
4. Type of Sources & JavaScript Object Name in Regular Expression Pattern ................ 16
5. Type of Sinks & Function Name in Regular Expression Pattern ..................... 17
6. Size and number of JavaScript files .......................................................... 19
7. Information of RWD framework .................................................................. 20
8. Path of RWD Framework ......................................................................... 20
9. Number of Sources & Source Density ....................................................... 21
10. Number of Sinks & Sink Density ............................................................. 21
11. Vulnerability density ranking .................................................................... 22

Listings

1. Inline JavaScript ......................................................................................... 6
2. Example of External JavaScript .................................................................... 6
3. Well-Crafted JavaScript Tag with Hyperlink ................................................... 10
4. Example of well-crafted URL in DOM-Based XSS ........................................ 11
5. Example of URI fragment ......................................................................... 11
6. Exploitation of Sources and Sinks in DOM-based XSS ................................. 12
7. Regular Expression Pattern of Sources ....................................................... 15
8. Regular Expression Pattern of Sinks .......................................................... 15
1 Introduction

With the expansion of Internet technologies, an online connection is not limited only in Desktop anymore. Users can connect online through numerous devices, for example, mobile phones and tablets. According to the report on Internet trends in 2013 [1], 30 percent of mobile phones were smartphones. The percent of page visit from mobile devices grew rapidly at 25 percent of all global web usage in 2014. Moreover, consumers spent more time on tablets and smartphones in 2013 at 1040 and 237 percent respectively compared to 2010. On the other hand, consumers spent less time on desktop [2]. Traditional websites are not designed to adapt the page layout to different screen sizes and resolutions. This limitation of the traditional websites can lead to the loss of potential customers [3]. The presentation of the modern websites needs to be optimized to provide customized webpage layout on all mobile devices.

At the present time, many devices with different sizes can connect online, website developers have to consider support for many different resolutions. Responsive Web Design (RWD) is a web design technique that provides an optimal page layout on any device regardless of the display resolution. RWD is used to solve the limitation of traditional websites that cannot dynamically adapt and fluidly fit to display with different sizes and resolutions.

Currently, there are two main approaches in designing and developing the mobile web application namely Adaptive Web Design (AWD) and Responsive Web Design (RWD) [4]. AWD is a technique which uses the device information from users to detect and deliver the display via m dot URL [5]. Contents are optimized for each platform, for instance, m.yoursite.com on iOS may design all images align left while Android may design all images align right. Developers use the benefit of device information to control a set of scripts (HTML, JavaScript and CSS). On the other hand, RWD is a technique where developers use media queries, HTML5, JavaScript and CSS set to control and deliver content that automatically adapts on all screens, resolutions and devices.

RWD is commonly used technique to achieve responsive layout as it requires less effort than AWD to provide support for additional devices. Moreover, it helps both consumers and website’s owners as consumers need a better user experience to access the contents and website’s owners also need to increase a higher number of visitors to visit website. As a consequence, there are various RWD frameworks released in the market to facilitate and shorten the development process. Many popular websites and communities have discussed and compared popular RWD frameworks only based on the features each framework offers. While a substantial number of popular websites and communities present the information by comparing only the features that offered by frameworks, they usually do not consider security vulnerability in their comparisons.

In this thesis, we focus on comparing the RWD frameworks based on their security vulnerability. Security vulnerabilities affect not only user experiences but also related to a reputation of business. Hence, security vulnerabilities is a crucial factor that should be considered. Today, attackers perform the malicious action against systems or websites. A degree of severity can be defined from low to high which depends on a particular security policy. Attackers can use different components of web application to attack a website, for instance, client-side script, HTML, HTTP and server-side script. RWD framework can also be source of these threats because RWD framework consists of HTML, JavaScript and CSS. If the attack is successful, the whole system or website is completely controlled by attackers. Upon a successful attack, attackers can endangered the system or perform any desirable action based on their purpose. As a result, users are affected by this threat while business might lose their reputation.

1.1 Problem Formulation

The aim of this thesis is to evaluate and compare RWD frameworks with respect to security vulnerability. In RWD, JavaScript plays an important role that drives features and interaction of Responsive sites. Moreover, JavaScript is selected as a main element by various RWD frameworks that provide plug-ins to enhance the interaction and features. Therefore, RWD frameworks which use JavaScript to drive features and interactions are more popular and the most widely used in the responsive design world. Because of the extensive usage in Responsive site, JavaScript attracts attackers to penetrate the system. Hence, JavaScript in RWD frameworks needs to be examined for presence of security vulnerabilities.
**Main Goal:** The main goal of this thesis is to determine the most secure of Responsive Web Design framework.

To achieve, we decomposed the main goal into 4 subgoals. At present, there are various Responsive Web Design frameworks released in a market daily. To perform the study, we have to select Responsive Web Design frameworks. This leads to the first research subgoal.

**Research Subgoal 1:** Identify the most commonly used frameworks that rely on JavaScript technologies to achieve the responsive layout.

At present, attackers try to perform an unauthorized access against the system. As a result, it leads to the security threat. There are many security vulnerabilities can be found based on each type of attack. This leads to the second research subgoal.

**Research Subgoal 2:** Identify the common security vulnerability that affects the selected RWD frameworks.

After the research subgoal 2 is answered, there might be many properties or components which lead the security vulnerability that can be found in RWD framework. This problem leads to the third research subgoal.

**Research Subgoal 3:** Test the identified vulnerability and gather the data.

After the research subgoal 3 is answered, the data from research subgoal 3 is required an interpretation before performing the evaluation and comparison. This problem leads to the final research subgoal.

**Research Subgoal 4:** Establish security metrics and comparison criteria

### 1.2 Research Methodology

In our thesis, we selected comparative study as our research methodology. The comparative study is used to observe some groups or domain of interest in order to compare the differences between the target groups [6]. The approach of comparative study provides an insight of the project based on the observation. As a result, the association between cause and effect or consequences are explored.

In general, case and comparative studies are more appropriate to use in software engineering than experiments because of the lack of control over the environment [7].

The research methodology consists of six steps as following:

1. Perform a literature reviews and review state-of-the-art of Responsive Web Design frameworks.

The current technology of Responsive Web Design, system requirements and its limitation will be conducted and use to support the research subgoal 1.

2. Identify four RWD frameworks that rely on JavaScript framework from the most commonly used frameworks.

3. Define the problem based on state-of-the-art Responsive Web Design framework.

4. Develop the prototype website based on selected Responsive Web Design framework.

The prototype website will be developed on the current version of each framework. Each prototype is constructed under provided resources from each vendor without any modification. After completion of prototype websites, all of them will be stored and performed under the same environment.

5. Test each selected framework on dedicated tools based on defined security metrics to find the vulnerabilities.

Each prototype website will be tested by dedicated tools. Data will be collected based on each security metric which identify to perform the evaluation and comparison.

6. Interpret the data and compare each framework based on each security metrics and defined criteria.

The evaluation and comparison of each RWD frameworks use the data which derive from step 5. These results will be analyzed and validated to guarantee the validity of conclusion.

The data from this study will be conducted and interpreted, where the interpreting results on each security metrics allow us to draw a conclusion.
2 Background

In this chapter, we present essential information about Responsive Web Design and the related security issues. More specifically, in Section 2.1, we provide an introduction to JavaScript. In Section 2.2, we provide general concept of Responsive Web Design and introduce each Responsive Web Design framework. In Section 2.3, we provide essential security vulnerability information. In Section 2.4, we present a background in detection vulnerabilities in web applications. In Section 2.5, we introduce the DOM-Based XSS detection tools. In Section 2.6, we introduce the regular expression. Finally, we introduce the line of code in Section 2.7.

2.1 JavaScript

JavaScript is a scripting language that execute and interpreted by interpreter instead of compiler on client-side[8]. With the advantage of executing on client-side, JavaScript is used to alleviate a duty of server-side when applying JavaScript on web application. Whereas other scripting languages, for example, Visual Basic Script, is also known as VB Script, requires Microsoft platform to execute and operate, JavaScript is platform independent. JavaScript has a syntax style similar to C and Java. The advantages of JavaScript are not limited to use only in web application. Moreover, JavaScript is used in the other applications, for example, Window Runtime app [9] and Adobe Acrobat PDF [10].

In web application, JavaScript is used for interaction by controlling a set of behaviors and event of elements. JavaScript is supported by several modern browsers. To use, JavaScript source code must be inserted between the middle of opening script tag `<script>` and closing script tag `</script>`. There are two ways to insert JavaScript code in website which are inline JavaScript and external JavaScript.

### Inline JavaScript

```html
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function convertToCelsius(fahrenheit) {
  return (5/9) * (f-32);
}
</script>
<body>
<p id="demo">The weather is 105 Fahrenheit. To convert to celsius click the button below.</p>
<button type="button" onclick="convertToCelsius()">Convert</button>
</body>
</html>
```

Listing 1: Inline JavaScript

Listing 1 shows an example of inline JavaScript. The JavaScript function convertToCelsius() is included within the `<body>` tag. The call function is invoked by using HTML event attribute onclick. It is quick and easy way for developers to include the JavaScript code. However, it is not the best practice because embedded JavaScript source code in HTML file is hard to manage and maintain. For example, If the function convertToCelsius() requires by several other pages, developers need to place this function in every page.

### External JavaScript

```html
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript" src="/js/convertToCelsius.js"></script>
</head>
<body>
</body>
```

Listing 2: External JavaScript
The weather is 105 Fahrenheit. To convert to Celsius click the button below.

```
<button type="button" onclick="convertToCelsius()">Convert</button>
```

Listing 2: Example of External JavaScript

Listing 2 shows an example of external JavaScript. External JavaScript is the most proper and common way to include the JavaScript in web page. JavaScript file and its path are included in src attribute inside the `<script>` tag separately. In contrast to inline JavaScript, developers just only update and maintain a script from a particular file instead of update every page. Therefore, developers can save their development time.

### 2.2 Responsive Web Design

Responsive Web Design (RWD) is a design approach to construct a display of contents and elements on website that fit in all screens and sizes. Traditional websites face a problem when displaying on multiple hand held devices. By fixed-width of the display in traditional website, a display cannot fit in small screen sizes especially on smartphones and tablets. Moreover, the technology of hand held devices change rapidly. Hand held devices, for example, smartphones and tablets, can display in both page orientation (landscape and portrait). Therefore, traditional websites cannot support both page orientation. The term of Responsive Web Design was introduced by Ethan Marcotte [11]. RWD was introduced to overcome the barrier of traditional websites that display cannot dynamically adapt on different hand held devices. Without tap or pinch to zoom in, a display of web page can display fluidly when viewing on both page orientation. Responsive Web Design uses Media Queries in CSS to define the styles when displaying in different media types and devices. However, CSS has its own limitation in RWD because some devices are not support the Media Queries. Moreover, size of CSS files become larger and larger because of included CSS rules which use to support in various devices. Therefore, JavaScript also has an important role to manage dynamic CSS rules. Nowadays, various RWD frameworks use JavaScript to manage and support the rendering process when display in multiple screens. Contents of web page including texts and images are defined into a single column instead of multiple column like traditional websites. Each block of contents stacks vertically and prioritize [12]. The most important contents are displayed first. Example of Responsive Web Design site¹ is presented in Figure 1.

![Responsive Web Design site](http://www.time.com)

Figure 1: Responsive Web Design site on Smartphone, Tablet and Desktop

Currently, there are many RWD frameworks released in the market. These frameworks reduce work load of developers and shorten a front-end development process. Moreover, first version of web prototype can be delivered to client much faster by supportable components in framework. In the reminder of this section we provide additional information about the frameworks considered in this thesis.

¹http://www.time.com
Bootstrap: Bootstrap has been developed by group of designers and developers at Twitter in 2010. This framework has become one of the most popular framework and used by popular website, for example, Newsweek.com and Vogue.com. Bootstrap uses fluid layout as a grid system for development. A set of CSS, fonts and JavaScript libraries are provided in the download package. Optimizing web site can be done rapidly by UI elements and widgets which developed by Bootstrap working team. The current version is v3.3.2 [13]. Bootstrap supports many modern browsers, for example, Chrome, Safari, Firefox, Opera and IE8-11 on four different operating system (Android, iOS, Mac OS X and Windows)[13]. Bootstrap has a large number of documentation and community support.

Foundation: Foundation has been developed by ZURB in 2008. Foundation uses fluid layout similar to Bootstrap. Foundation 5 is the current version [14]. In download package, Foundation provides basic components similar to other frameworks. However, Foundation is not support LESS which is a CSS pre-processor that allows developers to code less and shorten development time. However, User Interface(UI) elements and Widgets are also provided which help to shape a rapid prototype.

Gumby: Gumby is an easy customization framework which developed by Primavision Inc. [15]. The current version is Gumby 2. Multiple types of grid with various combination of column allow developers to develop and refine the prototype much easier. Therefore, the first prototype can be launched much faster. With a power of Grid system, developers can provide more options for the layout design to customers ranging from one to multiple column as desired. Download package’s Gumby Framework includes templates and User Interface(UI) kits. According to this benefits, it reduces the development time. Moreover, the large community with a subscriber around 2,000 members are ready to support the growth of this framework. Therefore, this framework can support and provides a complete documents of each templates and UI kits that they launched.

UIkit: UIkit is developed by YOOtheme. The current version is UIkit 2.19.0. This framework is a lightweight and modular front-end framework [16]. It provides a powerful web interfaces for a fast development. A comprehensive collection of HTML, CSS, and JavaScript components provide inside the download package. The provided components are well-structured which is easy to maintain. Hence, code is possible to extend and customize.

2.3 Security Vulnerability

The number of websites is increasing dramatically every year, which increases the information exchange over the network. As a result, attackers take advantages of this benefit and try to perform unauthorized action while researchers and security practitioners find the ways to defend the attacks. More than a hundred different types of threats on web application flaws are recognized nowadays. The Open Web Application Security Project(OWASP) has announced the most critical web application flaws in 2013 as follows [17]:

- A1 - Injection
- A2 - Broken Authentication and Session Management
- A3 - Cross-Site Scripting (XSS)
- A4 - Insecure Direct Object References
- A5 - Security Misconfiguration
- A6 - Sensitive Data Exposure
- A7 - Missing Function Level Access Control
- A8 - Cross-Site Request Forgery (CSRF)
- A9 - Using Components with Known Vulnerabilities
- A10 - Unvalidated Redirects and Forwards

One of the most commonly found vulnerabilities in web application is Cross-Site Scripting (XSS). In 2011, XSS ranked the fourth of top 25 most dangerous software errors in the Common Weakness Enumeration (CWE) community. Moreover, in 2013, OWASP was defined the XSS as the third of top 10 application security risks. Regarding to the ranking from both sources, it can be clearly seen that XSS is one of the most dangerous security threats in web applications.

XSS occurs when the input from users are not validated and sanitized. Attackers use this vulnerability to perform unauthorized action, for example, steal personal information from visitors’
trusted website or destruct web application. Preventing this flaw can be done from both client-side and server-side. XSS can be classified in 3 types:

- Persistent XSS
- Non-Persistent XSS
- DOM-based XSS

In the reminder of this section we describe in more details the three XSS types.

### 2.3.1 Persistent XSS

Persistent XSS is also known as Stored XSS or “TYPE-I XSS”. This type of XSS occurs on the target site which users input is not properly validated or sanitized. As illustrated in Figure 2, at first, the weakness of the target site are searched by attackers to identify the vulnerability [18]. Frequently, vulnerability can be found at the HTML input form which is not properly validated or sanitized. Once the attackers find the vulnerability. They perform a malicious action by injecting the malicious script through the form. Then, the malicious script permanently stores in the server or the database of target site. As illustrated in Figure 3, whenever a victim visits the site which infected by malicious script and browser executes the malicious script. The malicious script is command to steal the credential information, for example, cookies ID or session ID from victim. These credential information will be sent back to the attackers for further purpose. The most frequent example of Persistent XSS can be found on blogs, forums and message board [18].

![Figure 2: Storing Malicious Script in Server](image)

![Figure 3: Stealing Credential Information](image)

### 2.3.2 Non-Persistent XSS

As opposed to the Persistent XSS, in Non-Persistent XSS, the malicious script is injected but not stored on the target site or web server [18]. Attackers use the malicious script and send through the HTML input form. The malicious script is dynamically executed on the server-side and the result is sent back to the client-side. According to its behavior, the malicious action reflects and responses
directly to the attacker. Therefore, this type of XSS is also known as Reflected XSS. In Figure 4 and 5, we demonstrate an example of Reflect XSS. First, attacker goes to the target website. Attacker notices that the search feature uses the method “GET” to pass the keyword search as a parameter via the URL. Next, attacker traces the vulnerability by input keyword “Smith” to the search box. Search feature returns the result “There is no Smith in the search result” as figure 4.

![Figure 4: Attacker detects vulnerability](attachment:figure4.png)

Now, attacker performs the malicious action by attachment the well-crafted JavaScript tag within the hyperlink to inject the malicious script into the website. In Listing 3, it is an example of URL with well-crafted JavaScript.

```python
1 http://domainname.site/search.php?search=<script>alert(document.cookie)</script>
```

Listing 3: Well-Crafted JavaScript Tag with Hyperlink

This injection is successful because this search form does not validate and sanitize the user input. This malicious script is dynamically executed on the server side and the result reflects to the client side as pop-up message with victim’s cookies. An example of successful Non-Persistent XSS is illustrated in Figure 5.

![Figure 5: A successful attack in Non-Persistent XSS](attachment:figure5.png)

2.3.3 DOM-based XSS

This type of XSS is different from both Persistent XSS and Non-Persistent XSS as both of them are executed at server-side but DOM-based XSS is executed only on client-side. Attackers use Document Object Model (DOM) which is the root of the HTML document to perform this type of attack. DOM-based XSS occurs when JavaScript payload exploits by the modification of DOM-based on the original client-side script in the victim’s browser. To retrieve the response from the server, the JavaScript payload does not have to reach the server-side like Non-Persistent XSS [19]. As a result, the client-side script performs and interpret with unexpected manner. The DOM-based XSS is hard to detect because this attack happens during a browser rendering process. Therefore, general tools for XSS cannot detect this type of XSS because most of them detect XSS based on
In order to detect DOM-based XSS, we need to detect from the client-side environment. In Figure 6, we present the example of DOM-Based XSS.

In Listing 4, we present an example of well-crafted URL that attacker can use to perform DOM-based XSS. First, attackers use well-crafted URL that contains a malicious script to complete the mission.

```markdown
1 http://domainname.site/search.php#search=<script>alert(document.cookie)</script>
```

Listing 4: Example of well-crafted URL in DOM-Based XSS

Next, the browser executes URI fragment in Listing 5 and sends the request to a website. In listing 5, we present the URI fragment.

```markdown
1 /search.php#search=<script>alert(document.cookie)</script>
```

Listing 5: Example of URI fragment

The hash sign (#) in the URL commands the victim's browser that everything after # is not a part of query. The browser treats it as the fragment of URL. This fragment never delivers to the server-side via HTTP Request Messages which causing the malicious script to be inserted into the page while DOM manipulate. Finally, attacker receives the credential information. Based on the example of DOM-based XSS, we can clearly see that the DOM-Based XSS takes place only on client-side. However, this is just only a simple example of DOM-based XSS. Imagine that attackers use a complex malicious script and this script is widespread and cascaded from one popular site to the other related site. Credential information, for example, credit card number and personal information from million users leak and use by the attacker. This can cause damage and business reputation.

In an occurrence of DOM-based XSS, the untrusted data flows from one point in program and gets executed by some function in a program. In Section 2.3.4 and 2.3.5, we introduce the starting point of DOM-based XSS called sources and the execution part called sinks.

### 2.3.4 Sources

Sources are the DOM properties or any input that attackers can use to control and perform the DOM-based XSS [22]. The most popular sources and its properties in DOM-based XSS are presented in Table 1 [23].

### 2.3.5 Sinks

Sinks are a method or function in JavaScript that lead to DOM-based XSS [22]. The most popular sinks and its properties in DOM-based XSS are presented in Table 2 [23].
### Source Name & Property

<table>
<thead>
<tr>
<th>Source Name / Document URL / URL</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location / Document URL / URL</td>
<td>document.URL</td>
</tr>
<tr>
<td></td>
<td>document.documentURI</td>
</tr>
<tr>
<td></td>
<td>location</td>
</tr>
<tr>
<td></td>
<td>location.href</td>
</tr>
<tr>
<td></td>
<td>location.search</td>
</tr>
<tr>
<td></td>
<td>location.pathname</td>
</tr>
<tr>
<td></td>
<td>location.hash</td>
</tr>
<tr>
<td>Referrer</td>
<td>document.referrer</td>
</tr>
<tr>
<td>Window Name</td>
<td>window.name</td>
</tr>
</tbody>
</table>

### Sink Name & Property

<table>
<thead>
<tr>
<th>Sink Name</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Sink</td>
<td>eval</td>
</tr>
<tr>
<td></td>
<td>setTimeout</td>
</tr>
<tr>
<td></td>
<td>setInterval</td>
</tr>
<tr>
<td>HTML Element Sink</td>
<td>document.write</td>
</tr>
<tr>
<td></td>
<td>document.writeIn</td>
</tr>
<tr>
<td></td>
<td>(Element).innerHTML</td>
</tr>
<tr>
<td></td>
<td>outerHTML</td>
</tr>
<tr>
<td>Set Location Sink</td>
<td>location</td>
</tr>
<tr>
<td></td>
<td>location.href</td>
</tr>
</tbody>
</table>

### Table 1: Source Name & Property

### Table 2: Sink Name & Property

### 2.3.6 An Example of Vulnerable Code in DOM-based XSS

In this example, we selected some popular sources and sinks with its properties to demonstrate how sources can be influenced by attackers which lead to the execution in sinks to cause the DOM-based XSS.

1. `<DOCTYPE html>`
2. `<html>`
3. `<body>`
4. `<p>Dear Customers,</p>`
5. `<p>You have received 10 percent discount from our program.</p>`
6. `Now, you can use your discount at `<span id="seller">`</span>`
7. `<script>`
8. `var path = location.pathname.substring(1);`
9. `document.getElementById("seller").innerHTML = path;`
10. `</script>`
11. `</body>`
12. `</html>`

Listing 6: Exploitation of Sources and Sinks in DOM-based XSS

This page uses a path from URL to show where users can use their discount for shopping. The examples of URL uses in this example is `http://www.domainname.site/shopA`. At line 12, the sources “location” and its property “pathname” are used to retrieve the shop name from path in URL. At line 13, variable path stored the value of “shopA” and assign to the HTML element with id attribute of “seller” to display the name of shop. Attackers can simply exploit by sending HTTP request with attached malicious script like

```
http://www.domainname.site/shopA#<script>(document.cookie)</script>
```
This malicious script flows from sources name “Location” with its property “Pathname”. Then, the execution is taken place by HTML manipulation sinks name “.innerHTML”.

In some DOM properties in sources can be found in sinks as a format of JavaScript statement under the function in sinks. Moreover, some DOM properties are defined in both sources and sinks such as location.href. Attackers exploit the vulnerability of the program that can be found in sources and sinks to attack the system.

2.4 Detection of Vulnerabilities in Web Applications

The detection of vulnerabilities in web applications can be classified into two testing methods. The two testing methods are “White Box” and “Black Box” testing [24], [25].

2.4.1 White Box Testing

The white box or structure-based testing is the testing approach based on the internal structure of the program or software [26]. We can compare this testing technique with the transparent box. In this transparent box, source code are examined. White box testing can be used in several level of testing, for example, unit testing, integration testing [26]. In white box testing, static analysis is used to analysis the source code without any execution of the program. The static analysis can proceed by human to perform a manual review or use source code analysis [25]. However, manual review process requires experiences, expertise of reviewer and time. The manual review can lead a high number of error prone due to the human error [27]. This technique can be used in the early stage of software development life cycle. Therefore, defects and vulnerabilities can be found at the early stage of the software development life cycle [28].

2.4.2 Black Box Testing

The black box testing is the testing technique based on the software’s specification or requirement without examining the internal structure of the software [24], [29]. The purpose of black box testing is to dynamic analysis and examine on what software or function of software suppose to do based on the software specification [30]. We can compare this testing method in software security as the way that attackers perform the attack. To attack, attackers find the software vulnerabilities without having any information of the internal structure, architecture and source code behind the software. To detect vulnerabilities in Web application, fuzzing technique is the testing design technique in black box testing that use to detect vulnerabilities [25]. In fuzzing technique, a large number of random data are generated as an input to penetrate the system over HTTP protocol [25], [31]. The weakness in the system is identified by the penetration testing tools at runtime. The testing can be done automatically by the penetration testing tools.

2.5 DOM-based XSS Detection Tools

There are several vulnerability detection tools that can be used to detect the DOM-based XSS. In 2011, Wichers recommended DOM-based XSS detection tools on both commercial and open source [32].

Nowadays, several tools can be used to detect vulnerability of JavaScript. However, these tools need to be updated often to cover the pattern of penetration that happens daily. OWASP Xenotix XSS, IronWASP and regular expression pattern are the tools that can be used to identify the vulnerability.

2.5.1 Xenotix XSS Exploit Framework

This tool provides an advanced Cross Site Scripting (XSS) vulnerability detection. OWASP Xenotix XSS Exploit Framework is an open source tool. The most interesting features related to XSS are as follows:

- DOM Scanner
- DOM XSS Analyzer
• Local DOM XSS Analyzer

This framework claims that it provides a zero false positive in XSS result. A false positive is incorrectly identified defect or error in a program. The latest version of this tool is version 6.1. In Figure 7, we present the user interface of OWASP Xenotix XSS Exploit framework.

![Figure 7: OWASP Xenotix XSS Exploit Framework](image)

2.5.2 IronWASP

Iron Web Application Advanced Security Testing Platform (IronWASP) is an open source framework for web application vulnerability testing. It was developed by Lavakumar Kuppan. The most interesting features related to XSS are as follows:

• DOM XSS Checking

• Editable Source & Sink Lists

The latest version is IronWASP 2015 beta. In Figure 8, we present the user interface of IronWASP. Both tools report the number of sources and number of sinks for each JavaScript file.

2.6 Regular Expression

Regular Expression is a combination of character, operators and number in a pattern [33]. This pattern is used to match the input data. The input data can be a single or an array of characters. In modern web site, Regular Expression is widely used on both server-side and client-side script, for example, PHP: Hypertext Preprocessor (PHP), Application Service Provider (ASP), JavaScript which are server-side and client-side respectively. Frequently, we can find the regular expression in form validation and search filtering in web application.

Typically, a pattern is located between the delimiters, which is a pair of forward slashes. The pattern consists of characters and number. For example, /Regular Expression Pattern/, this regular expression is used to match string “Regular Expression Pattern”. In Table 3, we present some of the elements in the regular expression and its description [34]:

Regular expression is used to identify sources and sinks in various static analysis tools such as IronWASP and Xenotix tools. Regular expression in static analysis tools analyze the source code against the list of all defined sources, sinks and its properties. The list of all matches in statement will be used to improve in a particular section for securing the application.
The list of all sources with JavaScript object name and sinks with function name for each of the patterns are indicated in Table 4 and 5 respectively. This regular expression pattern is introduced in domxsswiki [35] which is the knowledge base community. The purpose of this project is to provide a knowledge of DOM-based XSS. The regular expression pattern covers a list of all sources and sinks which widely used in various JavaScript frameworks.

In listing 7 and 8, we present the regular expression patterns of both sources and sinks.

### Listing 7: Regular Expression Pattern of Sources

```
1 //\{location\s*\[\.:\]\s*\[\.:\]\s*\[\s*:*\]\s*\}\s*\{arguments\|dialogArguments\|innerHTML\|write\(\ln\)\|open\(\text{Dialog}\)\|cookie\|URL\|document\#URI\|base\#URI\|referrer\|name\|opener\|parent\|top\|content\|self\|frames\)\}\W)\s*/
```

### Listing 8: Regular Expression Pattern of Sinks

```
1 //\{(src\|href\|data\|location\|code\|value\|action\)\s*\[\s*:*\]\s*\s*\}\s*\}\s*/
```

2.7 Line of Code

Line of Code (LOC) is a software metric that use to to measure the size of the source file. Line of code is widely used as size metrics in software engineering [36]. To measure the line of code, white space, blank line, comments and non-executable statements are not considered since it is
<table>
<thead>
<tr>
<th>Type of Sources</th>
<th>JavaScript Object Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>location/documentURI/URL Sources</td>
<td>document.URL&lt;br&gt;document.documentURI&lt;br&gt;document.URLUnencoded (IE 5.5 or later Only)&lt;br&gt;document.baseURI&lt;br&gt;location&lt;br&gt;location.href&lt;br&gt;location.search&lt;br&gt;location.hash&lt;br&gt;location.pathname</td>
</tr>
<tr>
<td>Cookies Sources</td>
<td>document.cookie</td>
</tr>
<tr>
<td>Referrer Source</td>
<td>document.referrer</td>
</tr>
<tr>
<td>Window Name Source</td>
<td>window.name</td>
</tr>
<tr>
<td>History Sources</td>
<td>history.pushState()&lt;br&gt;history.replaceState()</td>
</tr>
<tr>
<td>Indirect Sources</td>
<td>localStorage&lt;br&gt;sessionStorage&lt;br&gt;IndexedDB  (mozIndexedDB, webkitIndexedDB, msIndexedDB)&lt;br&gt;Database (Safari Only)</td>
</tr>
<tr>
<td>Other Objects Sources (Opener)</td>
<td>opener&lt;br&gt;parent/top/frames[i].obj&lt;br&gt;event.data</td>
</tr>
</tbody>
</table>

Table 4: Type of Sources & JavaScript Object Name in Regular Expression Pattern

not the actual part of the source code. Currently, many counting tools are available in the market. These counting tools can solve repetitive and tedious task from counting a thousand line of code. Moreover, it can reduce the error prone causes by human. Universal Code Lines Counter is an example of the counting tools which are used to solve repetitive task of human. This tool supports many programming languages.
<table>
<thead>
<tr>
<th>Type of Sinks</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Sinks</td>
<td>eval, Function, setTimeout, setInterval, setImmediate, execScript, crypto.generateCRMFRRequest, ScriptElement.src, ScriptElement.textContent, ScriptElement.innerText, anyTag.onEventName</td>
</tr>
<tr>
<td>HTML Element Sinks</td>
<td>document.write, document.writeln, anyElement.innerHTML, Range.createContextualFragment, HTMLButton.value</td>
</tr>
<tr>
<td>Set Location Sink</td>
<td>location, location.href, location.pathname, location.search, location.protocol, location.hostname, location.assign, location.replace</td>
</tr>
</tbody>
</table>

Table 5: Type of Sinks & Function Name in Regular Expression Pattern
3 Method

In this section, we first revisit the research subgoals stated in Section 1.1. Next, we describe the approach for gathering the data to use for RWD comparison. Finally, we describe the approach for interpreting the data to use for RWD frameworks comparison.

3.1 Revisiting the Research Subgoals

According to the main goal of this thesis, we use a comparative study to determine the most secure of RWD frameworks. To achieve the goal we stated four research questions detailed in Section 1.1. The first question deals with selection of the RWD frameworks. We have selected four frameworks that are described in section 2.2. By performing the literature reviews and current state-of-the-art of Responsive Web Design, we found that these RWD frameworks contain the same criteria which use the JavaScript to manipulate the Responsive layout and its related components. These frameworks are the most commonly used JavaScript-based RWD frameworks according to different sources [37], [38], [39]. Moreover, the amount and the role of JavaScript in each of the frameworks is approximately the same.

The second question deals with identifying a common security vulnerability in the selected RWD frameworks. According to the selected frameworks in the first research subgoal, we found that DOM-based XSS is the security threat which the most related to the web application [17], [18], [23]. As described in Section 2.3, DOM-based XSS attacks affect RWD frameworks differently than other two XSS types. While the DOM-based XSS is executed on the client-side, the other two are executed on the server-side. DOM-based XSS exploits the vulnerabilities in both sources and sinks which can be found as a common in JavaScript files, as mentioned in Sections 2.3.4 and 2.3.5. Therefore, we focus on testing the selected RWD frameworks for the presence of DOM-based XSS security vulnerabilities.

We developed the prototype website of each RWD framework. In a download package, there are several JavaScript files contain and some of them are used as a JavaScript library for a plug-ins. Therefore, all JavaScript files may not be used at once due to purpose of usage. However, in order to test the identified vulnerability of each framework, all JavaScript files in each of the framework must be tested and called while performing the testing. Hence, in each framework, all JavaScript files were included in a single file. In general, static analysis provide results with a lot of false positive [40], [41]. This can cause an experimental bias. Therefore, we proposed to use two DOM-based XSS tools and one regular expression pattern to identify sources and sinks and cross check our results based on the implementation. Based on this proposal, we can mitigate the error prone from the tools. With the ease of usage, reliable results, execution time and community supports, both Xenotix and IronWASP are two of the best two among the open-sources type of DOM-based XSS tools and the regular expression pattern in domxsswiki is also widely used as a pattern in various static analyzer tool such as domxssscanner.com. These tools gave the vulnerabilities found on each framework were given as a number of sources and sinks. We use Xenotix, IronWASP and regular expression pattern to test the frameworks and gather the data.

3.2 Data Gathering

Most of the tools which use static analysis approach to identify DOM-based XSS applied different regular expression patterns which define according to the specific criteria by security experts. This regular expression patterns are run against the JavaScript source code to detect all possible sources and sinks. The report gave a number of all possible sources and sinks which can cause DOM-based XSS. In both Xenotix and IronWASP, we do not know exactly which sources and sinks were identified in its regular expression. We can only speculate which sources and sinks the tools detect. As a result, the number of sources and sinks were given differently in each tool. For example, the number of sources in Bootstrap detected by Xenotix and IronWASP were 2539 and 772 respectively. Therefore, we cannot conclude the result based on these tools to identify which RWD framework has the lowest vulnerability. Hence, we decide to use the regular expression pattern presented in Section 2.6 since in that way we know exactly which sources and sinks are included in the results.
According to this problem, we applied the regular expression patterns as described in Section 2.6 to detect the list of all possible sources and sinks because we know the identification of sources and sinks in regular expression pattern. This regular expression is introduced in domxsswiki [35]. Now, we can identify the number of sources and sinks in each of the selected RWD frameworks. The result is presented in Section 4.1 and 4.2.

### 3.3 Data Interpretation

To detect DOM-Based XSS, we will use static analysis technique. As mentioned in Section 2.3.3, DOM-based XSS exploits vulnerability due to the modification of DOM. Therefore, the RWD frameworks should be analyzed to find the vulnerability.

The vulnerabilities found in each framework need to be quantified in order to compare the vulnerability of RWD frameworks. Therefore, the software metric is required in this approach. We use the software metric Static Analysis Vulnerability Density (SAVD).

The Static Analysis Vulnerability Density metric can be measured of total number of vulnerability detected divided by size of source file (Measuring in per thousand lines of code (KLOC)) [42]. We used Line of Code (LOC) to measure the size of the source file. Line of code is widely used as size metrics in software engineering. By counting LOC, white space, blank line, comments and non-executable statements are not considered.

\[
\text{VulnerabilityDensity} = \frac{\text{TotalNumberofVulnerabilityDetected}}{\text{SizeofSourceFile}}
\]  

(1)

JavaScript file was analyzed by the tool and uses the static analysis approach to detect the vulnerability of each file. The tools provided the results as the number of sources and sinks. These two types of vulnerabilities which are sources and sinks were identified separately by using Static Analysis Vulnerability Density formula in below.

\[
\text{SourceDensity} = \frac{\text{TotalNumberofSourcesDetected}}{\text{SizeofSourceFile}}
\]  

(2)

\[
\text{SinkDensity} = \frac{\text{TotalNumberofSinksDetected}}{\text{SizeofSourceFile}}
\]  

(3)

The lower numbers in each vulnerability density formula mean the lower number of vulnerability in RWD framework.

Size of source file is a measure of the actual size of the source code data. In order to measure the size of the source file, whitespace, blank lines and comments are not included in the measurement because whitespace, blank lines and comments depend on the coding style of developers and its purpose of usage which affected the size of source file and they are not the part of the actual source code. There are several JavaScript files including in each framework. Therefore, our study used Universal Code Lines Counter version 1.1.6 to measure the size of the JavaScript file included in each package of the framework. Whitespace and comments of each JavaScript file were removed from the measurement. The size of source files was measured in thousand lines of code.

The size and number of JavaScript files after calculated by program are presented in Table 6.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Number of JavaScript files</th>
<th>Size (KLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootstrap 3.3.4</td>
<td>57</td>
<td>29.05</td>
</tr>
<tr>
<td>Foundation 5.5.1</td>
<td>23</td>
<td>9.45</td>
</tr>
<tr>
<td>Gumby 2.6</td>
<td>19</td>
<td>1.01</td>
</tr>
<tr>
<td>UIkit 2.19.0</td>
<td>78</td>
<td>11.59</td>
</tr>
</tbody>
</table>

Table 6: Size and number of JavaScript files

In calculation, we used Static Analysis Vulnerability Density (SAVD) equation as describe in Section 3.3. The results are represented as a ratio. This ratio was used to compare against the number of vulnerability found per framework. After comparison, we concluded which RWD framework has the lowest security vulnerability and the most secure in Section 4.3.
3.4 Study Setup

1) We downloaded Responsive Web Design package of each framework. The information of each framework are described in Table 7.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Source Code Version</th>
<th>Download Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootstrap</td>
<td>3.3.4</td>
<td><a href="http://getbootstrap.com/">http://getbootstrap.com/</a></td>
</tr>
<tr>
<td>Foundation</td>
<td>5.5.1</td>
<td><a href="http://foundation.zurb.com/">http://foundation.zurb.com/</a></td>
</tr>
<tr>
<td>Gumby</td>
<td>2.6</td>
<td><a href="http://gumbyframework.com/">http://gumbyframework.com/</a></td>
</tr>
<tr>
<td>UIKit</td>
<td>2.19.0</td>
<td><a href="http://getuikit.com/">http://getuikit.com/</a></td>
</tr>
</tbody>
</table>

Table 7: Information of RWD framework

2) For each framework, we included all JavaScript files at the index file (index.html) by using external JavaScript method with a full path reference to include files. The reason to include all JavaScript files at index.html files is for ease of testing by one URL per framework.

3) We uploaded all files including JavaScript, HTML, CSS which provided by each framework to the prepared host.

4) We set path of each framework to perform the experiment as following:

<table>
<thead>
<tr>
<th>Framework</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootstrap</td>
<td>testingLocation/bootstrap/index.html</td>
</tr>
<tr>
<td>Foundation</td>
<td>testingLocation/foundation/index.html</td>
</tr>
<tr>
<td>Gumby</td>
<td>testingLocation/gumby/index.html</td>
</tr>
<tr>
<td>UIKit</td>
<td>testingLocation/uikit/index.html</td>
</tr>
</tbody>
</table>

Table 8: Path of RWD Framework
4 Result

In this section, we first present the result of vulnerability density 1 & 2 detected by regular expression pattern in Section 4.1 and 4.2. Next, we describe the interpretation of the results in Section 4.3. In Section 4.4, we provide the discussion.

4.1 The Result of Source Density

The total number of sources derived from the static analysis approach by using regular expression pattern as presented in Listing 7. The source density derived from the total number of sources divided by the size of source file measured in KLOC as presented in Table 6. The total number of sinks and Sinks Density from each framework are presented in order from the lowest to the highest source density in Table 9.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Total Number of Sources</th>
<th>Source Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foundation</td>
<td>395</td>
<td>15.67</td>
</tr>
<tr>
<td>UIKit</td>
<td>354</td>
<td>30.54</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>1561</td>
<td>53.73</td>
</tr>
<tr>
<td>Gumby</td>
<td>148</td>
<td>389.93</td>
</tr>
</tbody>
</table>

Table 9: Number of Sources & Source Density

4.2 The Result of Sink Density

The total number of sinks and Sinks Density from each framework are presented in Table 10.

The total number of sinks derived from the static analysis approach by using regular expression pattern as presented in Listing 8. The sink density derived from the total number of sinks divided by the size of source file measured in KLOC as presented in Table 6. The total number of sinks and sinks density from each framework are presented in order from the lowest to the highest sink density in Table 10.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Total Number of Sinks</th>
<th>Sink Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foundation</td>
<td>107</td>
<td>4.98</td>
</tr>
<tr>
<td>UIKit</td>
<td>169</td>
<td>14.58</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>836</td>
<td>28.78</td>
</tr>
<tr>
<td>Gumby</td>
<td>47</td>
<td>105.63</td>
</tr>
</tbody>
</table>

Table 10: Number of Sinks & Sink Density

4.3 Interpretation of the results

As mentioned in Section 2.3.3, the connection between the sources and sinks are strong to each other. In an occurrence of DOM-based XSS, the untrusted action starts from sources as inputs and finishes at sinks where it gets executed. Therefore, we consider both sources and sinks as equally important aspects of a DOM-based XSS attack because of the strong correlation between each of them.

The result of Sources Density has shown that Foundation ranks the first when it comes to the density of sources in the Foundations’ JavaScript files. It has the lowest vulnerability in sources at 15.67 by comparing to other selected frameworks. In contrast, Gumby has the highest source density at 389.93 and stands at the last position.

The result of Sinks Density has shown that Foundation has the best ranking. It has the lowest vulnerability in sinks at 4.98. Gumby ranks the last position. It has the most Vulnerability in sinks at 105.63.

According to the results of both Vulnerability Density, we can conclude that Foundation ranks the first position. It has the lowest number of vulnerability found in the framework. Ulkit and
Bootstrap rank the second and the third position respectively. Gumby ranks the last position since it has the most vulnerability in both sources and sinks. Table 11 presents the rank of Vulnerability Density of RWD frameworks from the lowest to the highest.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foundation</td>
<td>1</td>
</tr>
<tr>
<td>UIKit</td>
<td>2</td>
</tr>
<tr>
<td>Bootstrap</td>
<td>3</td>
</tr>
<tr>
<td>Gumby</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 11: Vulnerability density ranking

4.4 Discussion

As described in Section 2.3.3, DOM-based XSS executes on the client-side and DOM is exploited by attackers to perform the malicious action. Apart from HTML and CSS, RWD framework typically uses JavaScript in order to control an interaction of Responsive site, for example, auto-resize images and Fade in/out images. According to the list of sources and sinks in Section 2.3.4 and 2.3.5, some properties or methods in JavaScript are used to manage the behavior of the elements in the HTML document. These sources and sinks are widely used in several JavaScript libraries and plug-ins in RWD frameworks. As a result, these JavaScript files may contain vulnerabilities that can lead to the DOM-based XSS. Therefore, in our study, we focus investigating security vulnerability of RWD frameworks that rely on JavaScript.

To study the DOM-based XSS, we first find an approach to identify the DOM-based XSS in JavaScript files. There are two main approaches in detecting DOM-based XSS tools, which are static analysis and dynamic analysis. The static analysis uses regular expression pattern which contains a list of sources and sinks in a defined library files to detect vulnerability but it has its own limitation with the false positives. On the other hand, dynamic analysis tests based on the implementation which depends on the skills of the developer using the framework. To identify vulnerabilities, we selected the static analysis approach since the testing in dynamic analysis rely on the implementation. Therefore, limitations of static analysis are limitations of this work, hence issue with false positives is inherited as well.

Regarding to the results in Section 4.1 and 4.2, we noticed that if size of framework is larger, the more number of vulnerability can be found. Therefore, the normalization in form of vulnerability density is used. However, the normalization relies on the line of code metric which depends on the structuring of the code. The better the structure of the code, the more lines there is, hence the vulnerability density is lower. Despite the limitation of the line of code metric, we still use it since it is widely used and there are no conclusive evidence of better metrics for JavaScript size of code.
5 Related Work

Amatya and Kurti [4] presented the literature survey on the challenge of cross-platform mobile development. The survey showed an essence of development technologies (HTML5) cross-platform development tools which are used in both web-based approach and hybrid approach to bridge the gap between platform. However, none of the literature survey mentioned the responsive design which is one of the key challenges on the usability of the cross-platform mobile development technology. In addition, there is no discussion on the security vulnerability on the client-side which is the one of the issues of the cross-platform technology that we address in this thesis.

Klein [21] introduced the DOM-based XSS as the third type of XSS. In the contribution, Klein described the scenario of DOM-based XSS and showed the difference between Non-Persistent XSS and Persistent XSS. Klein described an example of an execution of DOM-based XSS that occur on client-side and cannot detect by traditional security mechanism. The list of sources and sinks of the JavaScript code that may be exploited to by the attackers to perform the malicious action were discussed. Finally, the effective solution to detect the vulnerability were provided. In contrast, in our work we focus on detecting sinks and sources in a set of RWD frameworks by using a static analysis method. More specifically, we presented a completed list of sources and sinks in regular expression pattern that can be used to detect the vulnerability.

Malviya et al. [18] consolidated the body of knowledge on XSS in order to study the classification of XSS and approaches to mitigate the issues. In their study, they described the type of XSS, tools and its limitation that used to mitigate and protect the danger of XSS. However, the most of the approaches and tools were mainly supported the classic type of XSS which are Persistent and Non-Persistent XSS. An approaches or tools that can be used to solve the problem of DOM-based XSS were not discussed.

Fonseca et al. [25] presented an approach to evaluate and benchmark web vulnerability scanners. In their study, three commercial scanners were used to identify the potential vulnerabilities in two web applications. The results from the scanners were used to compare, evaluate and consider the percentage of false positive. The performance of the scanners due to the ability to detect the potential vulnerability can be used to improve the limitation and quality of the scanners. However, the case study conducted which refer to XSS did not identify the type of XSS found by each scanner. Moreover, this case study did not include the client-side in their experiment to evaluate the false positive on each of the scanner in order to identify the DOM-based XSS.

Muscat [23] and Di Paola [35] presented the general concept of DOM-based XSS and the process to identify the DOM-based XSS. They identified the list of sources and sinks which attackers can exploit to attack the system. The list of sources and sinks were identified based on the widely used JavaScript framework. Muscat provided an advance example of DOM-based XSS and the detecting result by the tool.

Stock et al. [20] proposed an approach to identify DOM-based XSS precisely with the lower false positive rate. To propose, they demonstrated a practical experiment on DOM-based XSS by combining the runtime taint tracking and taint-aware JavaScript and HTML parser. Stock, Ben, et al also discussed the current approach to detect the DOM-based XSS and State-of-the-Art of XSS auditor including string-based XSS filters and string-matching. However, a complete list of criteria on both string-based XSS and string-matching were not completely described. Moreover, pros and cons of the current XSS auditor are not available to reference the selection of XSS auditor.
6 Conclusion and Future Work

In this thesis we performed a comparative study on security vulnerability of a set of the most commonly used RWD frameworks. We first focused our study on frameworks that rely on JavaScript to achieve the responsive layout. Since JavaScript can be exploited in different ways to affect security of web applications, we focus on DOM-based XSS because is one of the most common security vulnerability in web application [17]. To identify vulnerabilities that can lead to DOM-based XSS, we use the regular expression pattern to list the amount of vulnerability which is sources and sinks from each framework. Since the frameworks differ in size, we use Vulnerability Density formula to normalize the results on the number of detected vulnerabilities divided by size of source file. Then, we calculated the size of the source file in all JavaScript files of each framework. The number of sources and sinks and the size of source files were used to measure Source Density and Sink Density respectively. Vulnerability density is the comparison criteria for comparing the RWD frameworks. We found that Foundation ranked the most secure amongst the selected RWD frameworks. It has the lowest number of vulnerability in both sources and sinks. UIKit and Bootstrap ranked the second and the third position. Finally, Gumby ranked the last position.

In our study, we had learned several lessons from the limitation of the experimental tools. The results of Xenotix and IronWASP tools were significantly different. Therefore, we could not use these results to conclude because we cannot identify the regular expression pattern that use as a criteria to identify the potential vulnerabilities. In addition, we used static analysis over dynamic analysis despite the limitations because dynamic analysis depends on the skills of the developer using the framework. Regarding to the limitations in each approach, we can use the combination between static and dynamic analysis as a complement to the solution for solving the limitation.

For future work, we intend to extend our comparative study on identifying more vulnerabilities other than DOM-based XSS that may exist in cross-platform mobile development. To explore our study, we will include more frameworks in our study because other cross-platform mobile development technologies may lead other type of attack but they still lack of identifying. Our future work can be used as a key decision to support a platform selection.
References


