PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTERY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
UNIVERSITY OFMOHAMED BOUDIAF - M'SILA

FACULTY: Mathematics and Computer Science
DEPARTMENT: Computer Science
N°:……

DOMAIN: Mathematics and Computer Science
BRANCH: Computer Science
OPTION: RTIC

Dissertation submitted in partial fulfillment of the requirements for The degree of MASTER

By: KADDOUR MOUSTAFA

SUBJECT

Implementation of Web Browser Extension For Mitigating CSRF Attack

Publicly defended before the jury composed of:

TAHRI ZOHIR	University of M'sila	Chair
SAOUDI LALIA	University of M'sila	Supervisor
OULDMOHAMEDI NAJIB	University of M'sila	Examiner

Academic Year: 2017/2018

ACKNOWLEDGEMENTS

First and Foremost praise is to ALLAH, the Almighty, the greatest of all, on whom ultimately we depend for sustenance and guidance. I would like to thank Almighty Allah for giving me opportunity, determination and strength to do my research. His continuous grace and mercy was with me throughout my life and ever more during the tenure of my research.

Writing this thesis has been fascinating and extremely rewarding. I would like to thank a number of people who have contributed to the final result in many different ways:

To commence with, I express my sincere and deepest gratitude to my supervisor, MS.Saoudi Lalia, who ploughed through several preliminary versions of my text, making critical suggestions and posing challenging questions. Her expertise, invaluable guidance, constant encouragement, affectionate attitude, understanding, patience and healthy criticism added considerably to my experience. Without her continual inspiration, it would have not been possible to complete this study.

I owe everything to my family who encouraged and helped me at every stage of my personal and academic life and longed to see this achievement come true. I dedicate this work to my pious grandfather, my sincere and generous father and my loving mother. Every breath of my life and drop of blood in my body is dedicated to my family

Last, but not the least, I gratefully acknowledge all my relatives for providing me constant encouragement, divine presence and supporting me spiritually throughout.

KADDOUR MOUSTAFA

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

List of figures:

List of tables:

ABBREVIATIONS LIST

OWASP	Open Web Application Security Project
HTTP	Hyper Text Transfer Protocol
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
XSS	Cross-site Scripting
CSRF	Cross-Site Request Forgery
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SOP	Same Origin Policy
XML	Extensible Markup Language
DOM	Document Object Model
OTP	One-Time Password
SSO	Single Sign On

GENERAL INTRODUCTION

GENERAL INTRODUCTION

Context of the study:

The Web has been an essential part of business all over the world and the browsers became the backbones of today's systems and applications. Nowadays, web applications are being developed in several fields and manage users' personal, confidential, and financial data, causing the security of such applications to become increasingly important. Back to the earlier Web and its first appearance, the websites were contained only a set of static web pages interconnected via hyperlinks. Then, images were added to web content. A request to a webpage might cascade other requests to different multiple websites content. Thus, cross-site or cross-origin requests triggered without user interaction. With the coming of the web notions JavaScript and forms, a cross-site interaction has become a true security threat to web applications. Typically, today's websites implement cookies as session management mechanism. The mechanism gave a browser an identity login cookie after user is successfully authenticated. Later, when the user visits the web pages of the target website, the browser will automatically put the identity login cookies in the request. This may not be removed till the browser is closed or the user logged out. CSRF vulnerabilities arise as a result of the browsers send the cookies back to the web server automatically with every request. If web application relied on cookies as a mechanism to stay tracking the user sessions, they're going to be at risk for this type of attack. The attackers will exploit this period of authentication to make the user's browser perform authenticated cross-origin requests probably without their knowledge and causes what's known as CSRF.

Motivation:

CSRF is one of the most serious attacks and has been recognized among the major threats to web applications and among the top ten worst vulnerabilities for web applications. Nowadays, many countermeasures and a set of techniques are available that does a reasonable job at defending against common CSRF attacks, however attackers still find vulnerabilities in websites and exploit them to carry out their attacks against servers and clients. With regard to CSRF protections, the existing countermeasures cannot defend against 100% of the scenarios encountered in the real world. Recently, many security assessment and bug bounty considering CSRF vulnerability have been reported. Also, a new experiment [33] analysis 300 web sites belonging to 3 different rank ranges of the Alexa global top 1500. The results of these experiments are alarming: out of the 300 web sites considered, 133 of these suffered

from at least one vulnerability enabling CSRF. The experiment also tested 132 additional web sites (again from the Alexa global top1500) and discovered that 95 of them were vulnerable to CSRF (i.e. 72%). these findings include serious vulnerabilities among the web sites of Microsoft, Google, eBay… etc.

The seriousness of a CSRF attack depends on the consequences of the state-changing action that was initiated by the attacker. For instance, in [25], it was shown that while being logged in on prominent web sites like *nytimes.com* (an online newspaper web site), or *INGdirect.com* (a famous banking web site) , if a victim user loads an attacker controlled web page in his/her web browser, the attacker can send forged HTTP requests to these web sites and steal the victim's personal email address stored at nytimes, or transfer money from the victim's ING bank account to the attacker's account etc.

Statement of the Problem:

The prevention from CSRF attacks is still a topic of active researches in the industry and academia. To protect users and servers application from csrf attack and overcome the attack, a set of techniques that do not require user interaction are available. The existing countermeasures can be classified into two main categories:

1. **Server-side:** The web application can use a three common mechanisms known today on the web to defend itself against cross-site request forgery attacks: validating a secret token, verifying the Same Origin with Standard Headers**,** and including additional headers with XMLHttpRequest**.** Unfortunately, not any of the proposed mechanisms is fully capable of carrying out this task.
2. **Client-Side:** This category let researchers and developers to implement new mitigation techniques to combat CSRF attacks. The client-side protection techniques are divided into two categories:
   - **Browser Extensions / add-ons:** It refers to mitigation techniques that are implemented based upon the browser extensions and used by the users. Example of such extensions is CsFire.

     The majority of these techniques have a small amount of attack vectors, with a high false positive.
   - **Proxy level framework:** This category represents client-side techniques that implemented on proxy level. Example: RequestRodeo.

     This technique requires a proxy installed on local host or on intranet, which affect the performance of the browser.

Objective:

In order to combat CSRF attacks, we proposed a novel technique for protecting users from basic and advanced CSRF attacks at client-side. Our proposed tool 'CSRF Detector' is a web browser extension implemented for Google Chrome.

Contributions:

In order to achieve the above objectives, our work makes the following contributions:

- Our proposed tool 'CSRF Detector' detect CSRF attacks with a hybrid mechanism based on the notion of content checking and request analysis.
- CSRF detector can detect reflected XSS attack in web site for the purpose of preventing attackers from using such vulnerability to carried out the CSRF attacks.
- CSRF Detector also has the ability to detect cross and sub-domain CSRF attack.
- CSRF Detector adopts **Google Safe Browsing** service to identify unsafe websites and notify users to protect themselves from harm websites (malware, phishing …).
- CSRF Detector also serves as a tool for web developers to detect CSRF vulnerability. It allows developers to quickly discover forms that are likely unprotected against CSRF attacks.

Report outline:

This report divided in four chapters:

- **The first chapter** provides two main sections, in the first one we explored specific topics related to web browsers, web servers, client/server communication and session management. In the second section of this chapter we discussed commons web application vulnerabilities.
- **The second chapter** provides a general overview of many defenses techniques that have been suggested on both server and client side to combat CSRF attacks.
- **The third chapter** presents our solution approach and the conceptual design of our Browser extension (CSRF Detector) with its different modules and methods for detecting basic and advanced CSRF attacks.
- **The fourth chapter** presents the implementation and experimentation steps of our CSRF Detector to ensure the efficiency of our tool.

Finally, we conclude this project by a general conclusion, and different perspectives.

CHAPTER 01

WEB APPLICATIONS AND VULNERABILITIES

CHAPTER 1
WEB APPLICATIONS AND VULNERABILITIES

1. Introduction

The phenomenal growth of the Web has made it a dominant platform for new software application. As a result, new web applications are being developed in several fields and manage users' personal, confidential, and financial data, causing the security of such applications to become increasingly important. Vulnerabilities in web applications can provide an unauthorized gain access to critical and sensitive information of users, increases in required technical support and costs may includes direct financial losses.

2. Web application

Web applications can be considered as a specific variant of client-server software as it is software that is typically distributed over a Web server and a Web browser. The Web server implements the application's logic. The client-side of the web application implements the user interface. It consists of HTML, JavaScript and CSS components that are received and interpreted by the Web browser [1].

2.1. Web browser

A Web browser is an important component of every computer system as it provides the interface to the Internet world. The browser allows user to view and interact with content on the web pages. It provides user the interface to perform the wide range of activities, such as personal financial management, online shopping, social networking and professional business. All major web browsers consist of various core components that interact in intricate ways to provide the web browsing experience [8]. Figure 1.1 illustrated a simplified view of the major browser components. The Networking component handles fetching HTML documents and implements the file transfer protocols such as FTP. The HTML document is parsed using the HTML Parser and XML Parser into a Document Object Model (DOM) tree. This DOM tree used by the Rendering Engine to display HTML and Extensible Markup Language (XML). The Browser Engine is the component that provides controls over the events, loads the given URI and manages the session. The JavaScript engine used to evaluate and execute the JavaScript code present on web pages. The User Interface component is the "frontend" of the Browser Engine, it displays HTML

documents based on the layout information received from the Rendering Engine. It provides features such as toolbars, navigation buttons and menus.

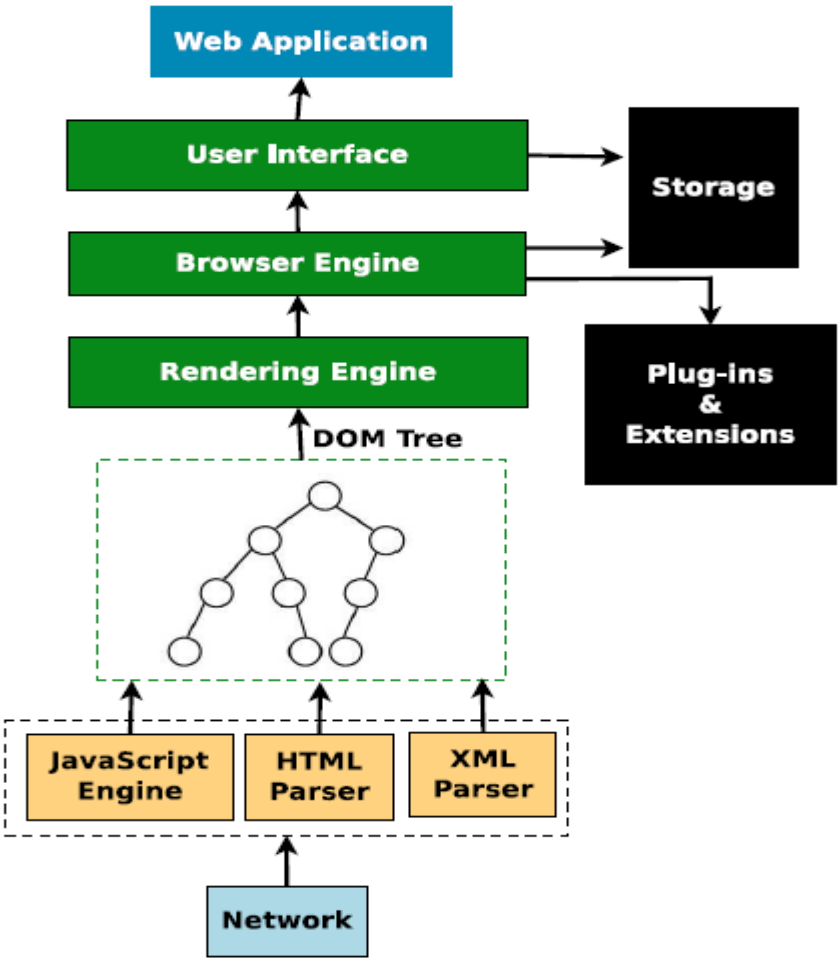

Figure 1.1: Major components of a web browser [8].

2.2. Web Server

Web content lives on web servers. Web servers speak the HTTP protocol, so they are often called HTTP servers. These HTTP servers store the Internet's data and provide the data when it is requested by HTTP clients [3].

Web servers enable HTTP access to a 'Web site' which is simply a collection of documents and other information organized into a tree structure. In addition to providing access to static documents, modern Web servers implement a variety of protocols for passing requests to custom software applications that provide access to dynamic content. Dynamic content can come from a variety of sources. Search engines and databases can be queried to retrieve and present data that satisfies the selection criteria specified by a user.

Measuring instruments can be probed to present their current readings (e.g. temperature, humidity). News feeds and wire services can provide access to up-to-the-minute headlines, stock quotes, and sports scores [2].

2.3. Structure

The structure of web applications has evolved over time. While traditionally web applications consisted of only a presentation layer which resides on the client machine, nowadays web applications commonly employ a three tier approach shown in figure 1.2. These three tiers are called: presentation, application and storage. The first tier is the Front End which is the HTML, JavaScript and CSS content rendered by the web browser. The Front End web server serves the static content and cached dynamic content. The application tier is implemented by an application server (ASP.NET, CGI, ColdFusion, Java EE, Perl, and PHP) [1].

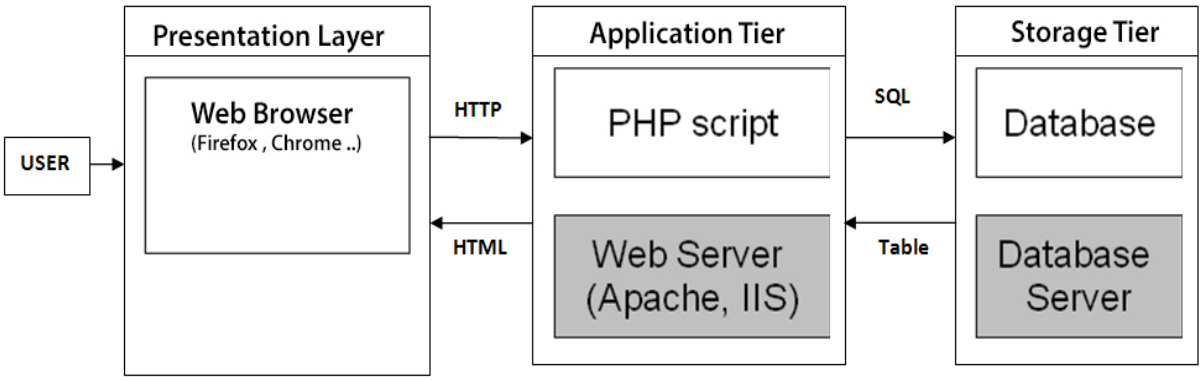

Figure 1.2: 3-tier web application structure.

2.4. Communication

Web servers, browsers, and proxies communicate by exchanging HTTP messages. The server receives and interprets HTTP requests, locates and accesses requested resources, and generates responses, which it sends back to the originators of the requests [2].

2.4.1 Uniform Resource Locators (URLs)

Uniform Resource Locators *(URLs)* provide the means to identify and locate a resource. HTTP URLs (RFC 1738) are a specific type of URLs and, as they are part of HTTP requests, they are the key to the communication between browser and server. A web application exposes one or more resources and all those resources can be identified and located through HTTP URLs [1].

2.4.2 The HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is the protocol programs use to communicate over the World Wide Web. There are many applications of HTTP, but HTTP is most famous for two-way conversation between web browsers and web servers [3].

The HTTP protocol is a stateless protocol which means that each request is handled as an isolated transaction and is independent from any previous request. As a consequence, web application developers need to implement session management mechanisms to manage state in web applications [1].

Once a server-authenticated secure channel has been established with the target website, the next step is to request the server for the path and query string specified in the URL. Versions 1.0 and 1.1 of HTTP are extremely straightforward: the client sends its request which consists of headers (one per line, consisting of the name of the header, followed by a colon and the value of the header) and a (possibly empty) body. The header and body are separated by an empty line. The very first header of the request is special: it contains an action (e.g. GET to retrieve a document, POST to submit a form), followed by a path and query string, and protocol version. The server reply also consists of a body and headers. The first reply header is also special: it contains the protocol version, status code, and status message of the response. For instance, a server may reply with HTTP/1.1 404 Not found, a well-known message on the Web [7].

2.4.3 HTTP version

HTTP uses a “<major>.<minor>” numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication [9].

The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message.

```
HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

2.4.4 HTTP Message

HTTP messages consist of requests from client to server and responses from server to client [9].

Both types of message consist of a start-line, zero or more header fields (also known as “headers”), an empty line indicating the end of the header fields, and possibly a message body.

```
HTTP-message = Request | Response; HTTP/1.1
```

2.4.5 HTTP Requests

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use [9].

- An HTTP request message has the following structure:

```
METHOD /path-to-resource HTTP/version-number
Header-Name-1: value
Header-Name-2: value
[ optional request body ]
```

2.4.6 HTTP Response

After receiving and interpreting a request message, a server responds with an HTTP response message. The first line of an HTTP response message is the *status line*. This line contains the version of HTTP being used, followed by a three-digit *status code*, and followed by a brief human-readable explanation of the status code [2].

- An HTTP response message has the following structure:

```
HTTP/version-number status-code message
Header-Name-1: value
Header-Name-2: value
[ response body ]
```

2.4.7 HTTP Request methods

There are varieties of request methods specified in the HTTP protocol. The most basic ones defined in HTTP/1.1 are GET, HEAD, and POST. In addition, there are the less commonly used PUT, DELETE, TRACE, OPTIONS and CONNECT [2].

2.4.8 HTTP Cookies

HTTP Cookies or simply cookies are used to exchange state information between web browser and web server. The exchange of cookies between web browser and web server happens as part of standard HTTP request and response messages [1].

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value1
```

Figure 1.3: HTTP response message with cookie.

```
GET /loans.php HTTP/1.1
Host: library.eurecom.fr
Cookie: name=value2
Accept: */*
```

Figure 1.4: HTTP request message with cookie.

Cookies have several different purposes including personalization, tracking of users on the web and session management. A cookie may be used to remember the information of a user that visited a particular website and show relevant content to the user in the future. For example, a cookie can be used to store search queries. Another application for cookies is the tracking of users. A website may store a unique identifier in a cookie. Then, each time the user visits the website, the cookie is stored along with the URL and date/time of the request in a logfile. By analyzing the logfile, the website's owner can obtain detailed information about the user including the sequence of webpage views. Cookies may contain sensitive information and, for security reasons, web browsers implement the Same Origin Policy. This prevents web pages from accessing cookies that belong to other domains than the origin domain to which the cookies correspond to. Hence, cookies cannot be shared across multiple domains [1].

Finally, cookies can be used by a web application to implement session management mechanisms which we will show in the next section.

2.5 Session Management

A web session is a sequence of network HTTP request and response transactions associated to the same user. Modern and complex web applications require the retaining of information or status about each user for the duration of multiple requests. Therefore, sessions provide the ability to establish variables such as access rights and localization settings which will apply to each and every interaction a user has with the web application for the duration of the session [11].

Session information is identified and stored on the web server by a session identifier which is generated as a result of a request to a resource on the web server. The web server stores the session information such as session identifier, username, account number, shopping basket content in memory, in flat-files on local disk or in a database [1].

2.6 HTTP authentication

Http authentication enables the web server to request authentication credentials from the browser in order to restrict access to certain WebPages. There are three methods frequently used: Basic, digest and NTLM [9].

In the mechanism defined in the HTTP protocol, when accessing a webpage that requires authentication, the browser will pop up a dialog asking for the username and password. After entering the information, the credential is encoded and sent to the web server via the Authorization request header. The browser remembers the credential until the browser is closed. When later the user visiting the web pages in the same authentication realm, the browser automatically includes the credential in the request via the Authorization header.

3. Web Vulnerabilities

Web applications have become a popular way to provide access to services and information. Millions of people critically depend on web applications in their daily lives. Over the years, web applications have evolved towards complex software systems that exhibit critical vulnerabilities [1].

An application that uses JavaScript and cookie-based sessions is exposed to, and must protect against a broad range of web attack vectors [7]. A website vulnerability is a weakness or misconfiguration in a website or web application code that allows an attacker to gain some level of control of the site, and possibly the hosting server. On average, websites experience 22 attacks per day, that’s over 8,000 attacks per year [31].

A rich set of tools and techniques have been developed to mitigate a subset of vulnerabilities present in web-based programs such as SQL injection (SQLI) and cross site scripting (XSS). The OWASP Top 10 is a powerful awareness document for web application security. It represents a broad consensus about the most critical security risks to web applications. Table 1.1 shows the ten most critical web application security risks for 2017 reported by OWASP.

OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

Table 1.1: The ten most critical web application security risks 2017 (OWASP) [30].

The OWASP Top 10 2017 introduced a new security issues as well as removed familiar attacks from the previous reported on top 10 2013 (**table1.2**) . Based on the data analyzed during the 2017 Top 10 and combined with better framework protections, CSRF has now fallen out of the Top 10 entirely [32]. CSRF is one of the top threats to web applications and has been continuously ranked in the OWASP Top Ten, it is well present in the OWASP TOP 10 since 2007, being ranked 5th in 2007 and 2010, and 8th in 2013.

The post-authentication CSRF is known at least since 2001 and has received a lot of attention from the web community. For instance, the OWASP Testing Guide devotes an entire section to this vulnerability .On the contrary, pre-authentication CSRF is not mentioned in the OWASP Testing Guide [33], although severe exploits have been reported in the literature, including:

- The execution of malicious JavaScript code at the victim's web browser [19].
- The association of the victim's financial details (or Google Search History) with the attacker's PayPal account or Google account, respectively [19].
- The tracking of the videos watched by the victim by the attacker on YouTube [35].

OWASP Top 10 – 2013
A1 – Injection
A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References
A5 – Security Misconfiguration
A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control
A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards

Table 1.2 : The ten most critical web application security risks 2013 (OWASP)

With regard to CSRF protections, many countermeasures and a set of techniques are available that does a reasonable job at defending against common CSRF attacks, but that cannot defend against 100% of the scenarios encountered in the real world .After a decade fighting the CSRF attacks. Most developers have at least heard the term, even thought, they don't fully understand it [32]. Recently, many security assessment and bug bounty have been reported. Also, a new experiment analysis considering 300 web sites belonging to 3 different rank ranges of the Alexa global top 1500. The results of this experiments are alarming and shows that there are 318 exploitable CSRF vulnerabilities affecting 185 web sites from the Alexa global top 1500 [33] .

4. Conclusion

We divide this chapter into two main sections, at the first one we explore specific topics related to web browsers, web servers, the client/server communication and session management. For the second section of this chapter we discuss commons web application vulnerabilities. Finally, we show an important web vulnerability that exploit user session information stored in browsers known as CSRF attack which will be considered and discussed in our work .

CHAPTER 02

CSRF ATTACK AND PROTECTION TECHNIQUES

CHAPTER 2

CSRF ATTACK AND PROTECTION TECHNIQUES

1. introduction

CSRF has been recognized for several years as one of the most important web vulnerabilities. Many countermeasures have been proposed in both server and client side. In this chapter we will see a detailed description of CSRF attack and the previous countermeasure against Cross-Site Request Forgery (CSRF) and its limitations.

2. Description

Cross-Site Request Forgery (CSRF) is an attack technique that exploits implicit authentication. The attack is executed by causing the victim's web browsers to create hidden HTTP requests to restricted resources. In the case that a successful request for such a resource, causes the web application to commit further persistent actions, these actions are done using the victim's authentication tokens [16].

CSRF attacks target functionality that causes a state change on the server, such as changing the victim's email address or password or purchasing something. Forcing the victim to retrieve data doesn't benefit an attacker because the attacker doesn't receive the response, the victim does. As such, CSRF attacks target state-changing requests [11].

CSRF attacks are also known by a number of other names, including XSRF, "Sea Surf", Session Riding, Cross-Site Reference Forgery, and Hostile Linking. Microsoft refers to this type of attack as a One-Click attack in their threat modeling process and many places in their online documentation [11].

3. Understanding the CSRF attack

A CSRF attack involves three parties, a victim, a trusted website where the victim is currently authenticated and an attacker [21]. CSRF is an attack that consists in tricking the victim to send an authenticated HTTP request to a target Web server, e.g., via visiting a malicious web page that creates such a request in the background. In cases that the user is currently in an authenticated state with the targeted Web site, the browser will automatically attach the session cookies to such requests, making the server believe that this request is a legitimate one and, thus, may cause state-changing actions on the server-side. **Figure 2.1** shows an overall diagram that gives an idea of how CSRF attack is performed by an attacker.

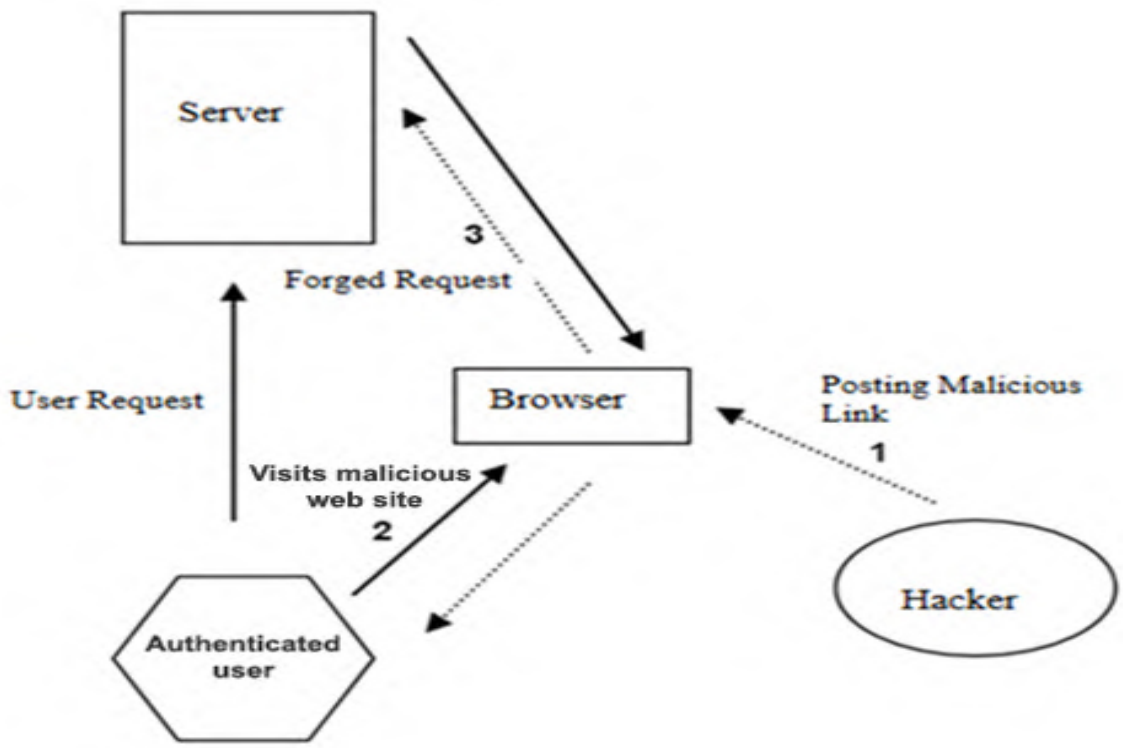

Figure 2.1: Overview of CSRF attack.

4. CSRF types

CSRF is one of the powerful vulnerabilities where an attacker performs the unauthorized activities without the knowledge of the user. The launching of CSRF attack may be carried out in different steps depending on the type of CSRF attack. CSRF can mainly be classified depending on the way in which the attacker makes the victim send the forged HTTP request (stored and reflected CSRF) and also depending on the state of CSRF victims with the trusted web application (pre-authenticated and post authenticated CSRF).

4.1. Stored and Reflected CSRF

A stored CSRF attack is one where the attacker can use the application itself to provide the victim the exploit link [23] or other content which directs the victim's browser back into the application and causes attacker controlled actions to be executed as the victim. Any application that uses HTML is vulnerable to CSRF attack where the attacker stores the malicious code using the IMG, IFRAME tag or XSS [18].

In Reflected CSRF attack, the attacker performs the attack outside the system application, by exposing the victim to exploit link [12] or content which is malicious. This can be done using a blog, an email message, an instant message, a message board posting, or even an advertisement posted in a public place with an URL that a victim types in. Reflected CSRF attacks will frequently fail, as users may not be currently logged into the target system when the exploits are tried. The trail from a reflected CSRF attack may be under the control of the attacker, however, and could be deleted once the exploit was completed [18].

4.2. Pre- and Post-authentication CSRF

From 2001 (first reporting of CSRF) to 2008, it was widely considered that only the state changing actions that can be caused by authenticated users need to be protected from CSRF attacks. This is mainly due to the assumption that only authenticated users can execute actions having high-impacts (e.g. transferring money from one account to another). However, in 2008, Login CSRF attack was introduced [19]. Unlike previously classic CSRF attacks, the victims of Login CSRF are note authenticated users. Depending on this, CSRF attacks can be classified into two categories: CSRF attacks that do not require the victim to have an authenticated session with the vulnerable website (refer to as pre-authentication CSRF attacks) and the type of CSRF attacks that require the victim to have an authenticated session (refer to as post-authentication CSRF attacks).

4.2.1. Post-authentication CSRF (Classic CSRF)

The goal of a CSRF attack is to forge a request from a victim's browser to the target application, triggering state-changing effects in the target application. A classic CSRF attack can only be successful if the forged request happens within a previously authenticated session between the victim's browser and the target application. Unfortunately, the design of current session management mechanisms in the browser attaches session information to any outgoing request, fostering the prevalence of the classic CSRF attacks [17]. **Figure 2.2** shows different steps in classic CSRF attack. The user establishes an authenticated session with website 'A' as illustrated in steps 1–4. The attacker triggers a request from origin E to the target application at origin 'A' (step 13), to which the browser attaches the cookies from the existing session with origin A. If origin 'A' does not have CSRF protection, this request will be executed as if it was generated by the user.

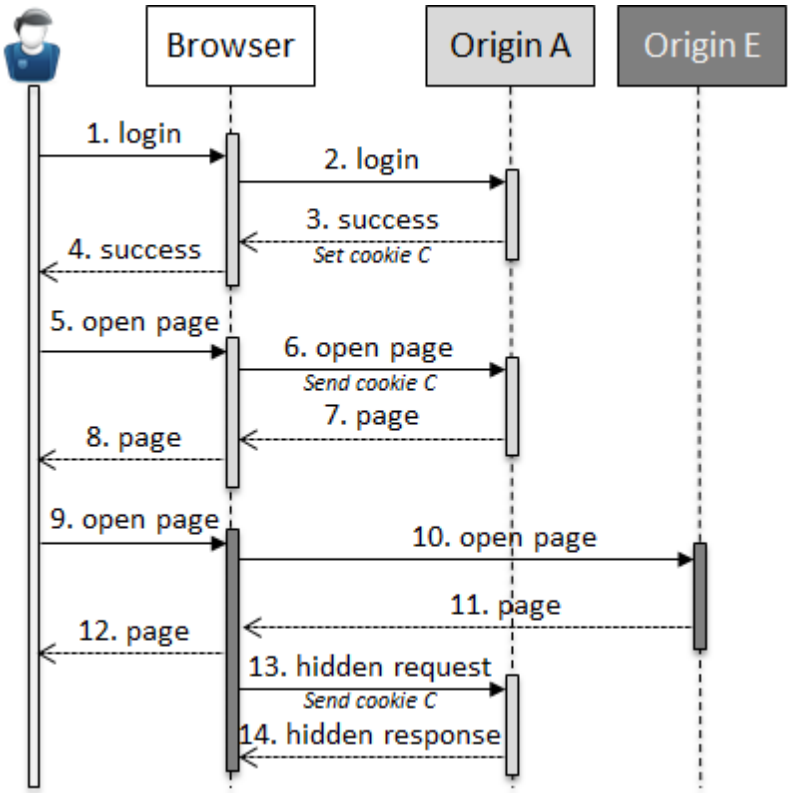

Figure 2.2: Steps in Classic CSRF [17].

4.2.2. Pre-authentication CSRF (Login CSRF)

In a login CSRF attack, the attacker forges a login request to an honest site using the attacker's username and password at that site. If the forgery succeeds, the honest server responds with a Set-Cookie header that instructs the browser to mutate its state by storing a session cookie, logging the user into the honest site as the attacker. This session cookie is used to bind subsequent requests to the user's session and hence to the attacker's authentication credentials. Login CSRF attacks can have serious consequences, depending on other site behavior [18].

The main objective of this type of attack is the attacker and the users are both authorized users. But the attacker will send his identity to the user by means of some link. The user will now enter using the attacker's identity. Then all information related to the user will get stored in the user browser, but actually in the attacker identity. So, the attacker will now see this information of the user. Therefore, they input personal data to the web server such as credit card numbers or search keywords. Moreover, by using a Gadget, adversaries can execute malicious gadgets registered to the adversaries' accounts within the victim's local machine. Both Classic CSRF and Login CSRF are

subtle exploitations based on the common belief that every web request made by a browser is initiated under users' supervision [19].

```
<form action=https://www.google.com/login method=POST target=invisibleframe>
<input name='username' value= 'attacker'>
<input name='password' value='xyz'>
</form>
<script>document.forms[0].submit()
</script>
```

Figure 2.3: Sample script for login CSRF.

5. Basic CSRF attacks techniques

CSRF attacks specifically target state-changing requests, not theft of data. And this can be done in several ways as presented in the following sections.

5.1. Image-based CSRF attacks

Image-based CSRF attack works by including HTML image elements, attributes or image objects in a page that accesses a site to which the user is known (or is supposed) to have authenticated [23]. The attacker can use image elements when the action requires an HTTP GET request. A sample image tag containing a malicious code is shown in the figure below.

```
<IMG
src="http://bank.com/transfer.php?account=Alice&amount=100000&for=Bob"
width="0" height="0"
/>
```

Figure 2.4: Sample image tag containing a malicious code.

The **Figure 2.5** shows how an attacker can use image element to perform the CSRF attack. Firstly, the victim is currently authenticated on the trusted website and visits a web page controlled by the attacker. When visiting the attacker controlled web page, the web page tells the victim's web browser to send a request to the trusted web application by inserting the malicious link of the trusted site into an image tag. The victim browser will try to fetch automatically the image and will send an unwanted request to the URL of the image. Because the image is located at the victim browser, the victim's web browser will include all the information that the web browser has associated with that domain. This includes the cookies which the web application uses to keep the victim authenticated.

Therefore, when the trusted web application receives the request from the victim, the session will be authenticated to the trusted web application and the request will look like a normal request from the victim's web browser.

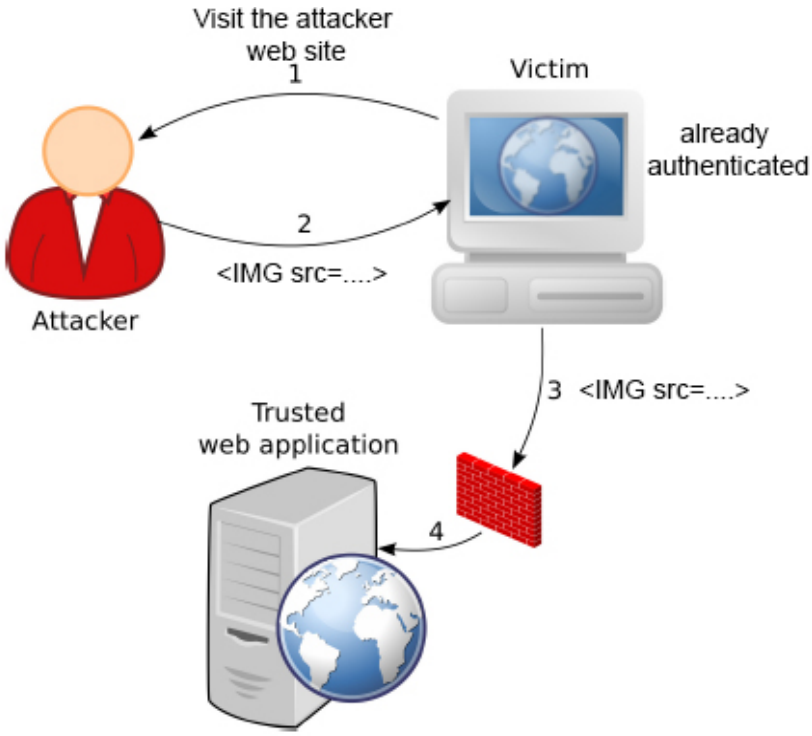

Figure 2.5: CSRF attack using IMG element.

5.2. Invisible iframe-based CSRF attack

If the action requires HTTP POST, The attacker can create a form using HTML or JavaScript. These require the attacker to have some degree of control over the CSRF site, as they will have to embed their own HTML in the site. They can gain this control either by being the site owner or by finding some sort of cross-site scripting vulnerability in the site.

```
<html>
<body onload="document.frames[0].submit()">
<form action="http://target.example.com/action.html" method="POST">
<input name="field1" value="value1">
<input name="field2" value="value2">
</form>
</body>
</html>
```

Figure 2.6: Example of Auto-submitting form on the attacker site.

- The attacker would inject the following code into an invisible iframe:

```
<iframe width="0" height="0" style="visibility: hidden;"
src="http://attacker.example.com/CSRF.html"/>
```

6. CSRF protection techniques

To protect users and servers application from csrf attack and overcome the attack, a set of techniques that do not require user interaction are available. The existing countermeasures can be classified into two main categories:

- Server-side protection techniques.
- Client-side protection techniques.

6.1. Server-side protection techniques

The web application can use a three common mechanisms known today on the web to defend itself against cross-site request forgery attacks: validating a secret token, verifying the Same Origin with Standard Headers**,** and including additional headers with XMLHttpRequest. Unfortunately, not any of the proposed mechanisms is fully capable of carrying out this task.

6.1.1. Secret validation token

Secret Validation Token is a well known server-side protection scheme against CSRF attacks. Not like other verification, the token approach does not require user intervention, so the users will not know that something has been used to protect them. This scheme sends additional information with each HTTP request to determine whether the request came from an authorized user. To apply token, web applications must first create a “pre-session”, and then proceed forward a real session after successful authentication. Validation token should be hard to guess for an attacker who does not already have access to the user’s account. If a request is missing a token or the token does not match the expected value, the web application should reject the request and prompt the user [24].

One disadvantage of using the Secret Validation Token is that, occasionally, some users disclose the contents of web pages they view to third parties, for example via email. If the page contains the user’s Validation Token, anyone who views the contents of the page can impersonate the user to the web application until the session expires [25].

6.1.2. Verifying Same Origin with Standard Headers

There are two steps to this check:

- Determining the origin of a request is coming from (source origin).
- Determining the target of a request is going to (target origin).

Both of these steps rely on examining an HTTP request header value. Although it is usually trivial to spoof any header from a browser using JavaScript, it is generally impossible to do so in the victim's browser during a CSRF attack, except via an XSS vulnerability in the site being attacked with CSRF. More importantly for this recommended Same Origin check, a number of HTTP request headers can't be set by JavaScript because they are on the 'forbidden' headers list. Only the browsers themselves can set values for these headers, making them more trustworthy because not even an XSS vulnerability can be used to modify them [11].

To identify the source origin, web application server can use one of these two standard headers that almost all requests include one or both of:

- Origin Header.
- Referer Header.

6.1.2.1. Checking the Origin Header

The Origin HTTP Header standard was introduced as a method of defending against CSRF and other Cross-Domain attacks. The Origin header will be present in HTTP requests that originate from an HTTPS URL. If the Origin header is present, then it should be checked to make sure it matches the target origin [26].

- The general format of the Origin header:

```
Origin: <origin> [<origin>]*
```

An <origin> is a combination of scheme, host and port. Unlike HTTP Referer, no path data or query string will be provided in the origin. The first origin value will be the initial source of the request, and any remaining values will be origins of any redirects that changed the target of the request [26].

6.1.2.2. Checking the Referer Header

An HTTP request's referrer [7] indicates the URL of the webpage that contained the HTML link or form that was responsible for the request's creation. The referrer is

communicated via an HTTP header field. To protect against CSRF, web application checks if a request's referrer matches the web applications domain. If this is not the case, the request is usually rejected [9].

The web server can verify the hostname in the Referer header matches the target origin. Checking the Referer is a commonly used method of preventing CSRF on embedded network devices because it does not require any per-user state. This makes Referer a useful method of CSRF prevention when memory is scarce or server-side state doesn't exist. This method of CSRF mitigation is also commonly used with unauthenticated requests, such as requests made prior to establishing a session state which is required to keep track of a synchronization token [11].

- The general format of Referer Header:

```
Referer: <url>
```

6.1.3. Custom request headers

Custom HTTP headers can be used to prevent CSRF because the browser prevents sites from sending custom HTTP headers to another site but allows sites to send custom HTTP headers to themselves using XMLHttpRequest [19].

XMLHttpRequest’s popularity has increased recently with more sites implementing AJAX interfaces. Sites can defend against CSRF by setting a custom header via XMLHttpRequest and validating that, the header is present before processing state modifying requests. Although effective, this defense requires sites to make all-state modifying requests via XMLHttpRequest, a requirement that prevents many natural site designs [19].

6.1.4. CSRF Specific Defense

There are numerous ways that web servers can specifically defend against CSRF. The flowing three mechanisms are the most recommended by OWASP as a second check and an additional precaution.

6.1.4.1. Synchronizer CSRF Tokens

The most popular CSRF defense is to include a secret token with each request and to validate that the received token is correctly bound to the user’s session, preventing CSRF by forcing the attacker to guess the session’s token. There are a number of variations on this approach, each fraught with pitfalls, and even sites that implement

the technique correctly often overlook their login requests because login request lacks a session to which to bind the token [19].

Many implementations of synchronizer tokens include the challenge token in GET (URL) requests as well as POST requests. This is often implemented as a result of sensitive server-side operations being invoked as a result of embedded links in the page or other general design patterns. These patterns are often implemented without knowledge of CSRF and an understanding of CSRF prevention design strategies. While this control does help mitigate the risk of CSRF attacks, the unique per-session token is being exposed for GET requests. CSRF tokens in GET requests are potentially leaked at several locations: browser history, HTTP log files, network appliances that make a point to log the first line of an HTTP request, and Referer headers if the protected site links to an external site [11].

6.1.4.2. Double submit cookie

Storing the CSRF token in session is problematic, an alternative defense is the use of a double submit cookie. The double-submit cookie method offers a protection against CSRF that can be implemented without the need for a server-side state. Thereby, the CSRF token is submitted twice. Once in the cookie and simultaneously within the actual request. As the cookie is protected by the Same-Origin Policy only same-domain scripts can access the cookie and thus write or receive the respective value. Hence, if an identical value is stored within the cookie and the form, the server side can be sure that the request was conducted by a same-domain resource. As a result, cross-domain resources are not able to create a valid request and therefore CSRF attacks are rendered void [42].

6.1.4.3. Encrypted token pattern

The Encrypted Token Pattern leverages an encryption, rather than comparison, method of Token-validation. After successful authentication, the server generates a unique Token comprised of the user's ID, a timestamp value, and a nonce, using a unique key available only on the server. This Token is returned to the client and embedded in a hidden field. Subsequent AJAX requests include this Token in the request-header, in a similar manner to the Double-Submit pattern. Non-AJAX form-based requests will implicitly persist the Token in its hidden field. On receipt of this request, the server reads and decrypts the Token value with the same key used to create the Token. Inability to correctly decrypt suggest an intrusion attempt. Once

decrypted, the UserId and timestamp contained within the token are validated to ensure validity; the UserId is compared against the currently logged in user, and the timestamp is compared against the current time [11].

6.2. Client-side protection techniques

6.2.1. The Same Origin Policy (SOP)

The most widespread countermeasure is the Same Origin Policy (SOP), implemented in most browsers. This policy limits access to DOM properties and methods to scripts from the same 'origin', where the origin is usually defined as the triple <domain name, protocol, TCP port>. This prevents for example, a malicious script on the website evil.com from reading out session identifiers stored in a cookie from homebanking.com. Unfortunately, the protection offered by the SOP is insufficient. Although the SOP prevents the requesting script from accessing the cookies or DOM properties of a page from another origin, it does not prevent an attacker from making requests to other origins. The attacker can still trigger new requests and use cached credentials, even though the SOP prevents the attacker from processing responses sent back from the server [29].

6.2.2. client-side proxy and web extension

On top of SOP, a client-side protection technique can be used to protect users from CSRF attacks by monitoring outgoing requests and incoming responses. This countermeasure can be implemented as a browser proxy or extension to web browsers.

6.2.2.1. Requestrodeo

Johns and Winter [16] proposed RequestRodeo as a client-side protection proxy against CSRF. RequestRodeo lies, next to cookie-based HTTP authentication. This technique offers protection via detecting cross-domain requests and then removal of cookie values from these requests (stripping of implicit authentication). A request is authenticated when it satisfies the Same Origin Policy (SOP) and it initiated as a result of an interaction with the currently viewed browser tab [16].

Limits:

RequestRodeo is limited to only certain HTTP requests and not for HTTPS requests, so it does not scale well to web 2.0 applications. RequestRodeo fails to detect all JavaScript dynamic links in the responses, since this dynamic content has come after passing through the proxy. Also, RequestRodeo does not differentiate

between malicious and genuine cross-origin requests, so it provides very poor protection against CSRF [26].

6.2.2.2. CSFire

CsFire is integrated extension into Mozilla browser to mitigate CSRF attacks, it extends the work of Maes and al [27]. CsFire is the only system that provides formal validation through bounded model checking to defend against CSRF in the formal model of the web developed by Akhawe and al [27]. CsFire strips cookies and HTTP authorization headers from a cross-origin request. The advantage of stripping cookies and HTTP authorization headers is that there are no side-effects for cross-origin requests that do not require credentials in the first place.

Additionally, CsFire supports users for creating custom policy rules, which use user supplied whitelist and blacklist to certain traffic patterns.

Limits:

The Limits of CsFire is that approach relies only on examining the cross-domain request and protect users against one specific type of CSRF attack which is Reflected CSRF.

6.2.2.3. NoScript ABE

ABE is an application Boundary Enforcer, restricts an application within its origin, which effectively strips credentials from cross-origin requests unless specified otherwise. The default ABE policy only prevents CSRF attacks from the internet to an intranet page. The user can add specific policies, such as a CsFire-alike stripping policy, or a site specific blacklist or white list. If configured with, ABE successfully blocks the CSRF attack scenarios, but disables the payment and central authentication scenario [27].

6.2.2.4. RequestPolicy

RequestPolicy protects against CSRF by blocking all cross-origin requests. In contrast to stripping credentials, blocking a request can have a very noticeable effect on the user experience. When detecting a cross-origin redirect, RequestPolicy injects an intermediate page where the user can explicitly allow the redirect. RequestPolicy also includes a predefined whitelist of hosts that are allowed to send cross-origin requests to each other. Users can add exceptions to the policy using a whitelist. RequestPolicy successfully blocks the three attack scenarios (by blocking instead of

stripping all cross-origin requests) and requires interactive end-user feedback to enable the payment and central authentication scenario [37].

7. CSRF protection: Require User Interaction

Sometimes it is easier or more appropriate to involve the user in the transaction in order to prevent unauthorized transactions. Challenge-response is a very strong defense against CSRF; it does not impact user experience [11]. The following are some examples of challenge-response options:

- **Re-Authentication:** some websites enforce users to re-enter credential information such as password when action require state changing at a web server.
- **One-time Password Token (OTP):** a password that is valid for only one login session or transaction. The use of one-time password tokens hardens a traditional ID and password system by adding another dynamic credential. OTP can be used to mitigate the CSRF attack by simply asking the user to enter OTP when action requires state changing at a web server.
- **CAPTCHA:** a computer program or system intended to distinguish human from machine input. This challenge-response interact with users and prevent the executed of unknown actions behind users.

8. Advanced CSRF attacks

The following advanced techniques are used by attackers to bypass CSRF protection and exploit the user session.

8.1. Using XSS to bypass CSRF protection

XSS is well-known web vulnerability.XSS attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it [36].

According to OWASP, Attackers can use XSS to defeat any automated cross-site request forgery (CSRF) defense the application might employ [30]. The anti-CSRF can be stolen accordingly. This is what called vulnerability chaining.

For example, a web application that can defend itself against cross-site request forgery attacks and have a form which is not vulnerable to CSRF by implementing the CSRF Token protection (example shown in the box below). The token will be verified before the action is executed.

```
<input type="hidden" name="token" value="<?php print$_SESSION['token']; ?>"/>
```

- The script shown below adds new administrator when the button is clicked

```
<form method="get" action="add_admin.php">
<input type="text" name="name" value="new_admin" />
<input type="hidden" name="token" value="<?php print $_SESSION['token']; ?>" />
<input type="submit" name="submit" value="add admin" />
</form>
```

- The request generated will be such as:

```
http://www.site.com/add_admin.php?name=Nytro&token=1htFI0iA9s&submit
```

The token sent from the form will be verified with the token of session stored on web server. If it was the same, new admin will be added. Although, in case of the web site thathas a vulnerable to XSS. The attacker can bypass CSRF protection easily by using JavaScript code.

At first, the attackeruses JavaScript codeto create iframe which will open the administration page and put everything in a function and will be called at the onload event of the iframe. The js file will be considered uploaded on:

« **http://www.attacker.com/script.js** » and contains the following JavaScript Code:

```
document.writeln('<iframe id="iframe" src="/admin/admin.php?action=add_admin"
width="0" height="0" onload="read()"></iframe>');
functionread()
{
varname = 'administrator_name ';
vartoken=document.getElementById("iframe").contentDocument.forms[0].token.value
;
document.location.href= 'http://127.0.0.1/admin/add_admin.php?name=' + name+
'&token=' + token+ '&submit';
}
```

If the victim makes a request to:

http://site_victim.com/index.php?name=<script src="http://127.0.0.1/script.js></script>

It will be redirected to:

http://site_victim.com/admin/add_admin.php?name=administrator_name&token=aH52G7jtC3&submit

The forged request looks the same as the normal request. The server will check the validity of the token and execute the requested action. The victim has been exploited to add new admin without knowledge.

8.2. Bypass Double-Submit Cookie protection

The attackers can defeat the double-submit cookie protection and set the CSRF cookie by simply exploit subdomains. For example, an attacker controls and own a malicious sub-domain '*https://evil.example.com/*'. In this case, the attacker can set the cookie for the parent domain '*https://example.com*'. **Figure 2.7** shows an example of set cookie for parent domain by sub-domain response.

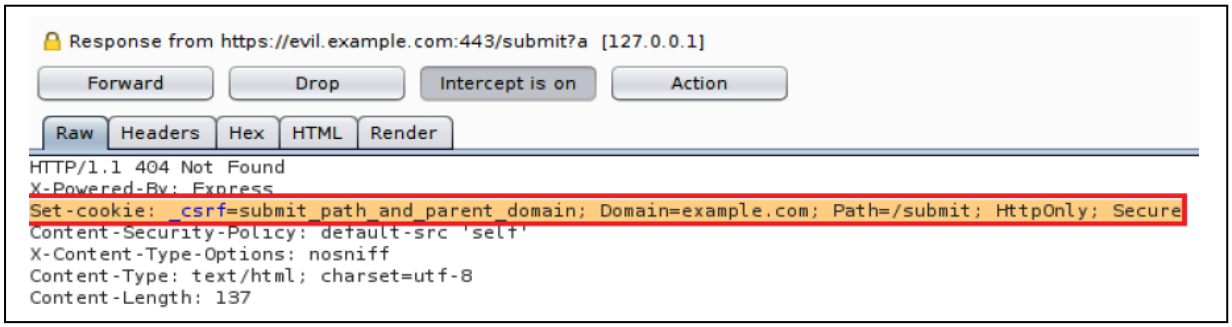

Figure 2.7: Example of set cookie for parent domain by Sub-domain response.

Now, the attacker can use this to controls cookies sent to *https://www.example.com/submit*. When user submits a request, the browser will have two values to the same cookie name (the original cookie and the attacker cookie). The two values will be sent with that request and according to RFC, the cookie with longer path should be the first. The server at this point has no way to figure out which is the right cookie and which is the attacker cookie because no information is sent with cookies. In this case, the server side will look for the first value of the cookie name which is the attacker cookie. The **Figure 2.8** below shows how an attacker can send the CSRF cookie to the parent domain.

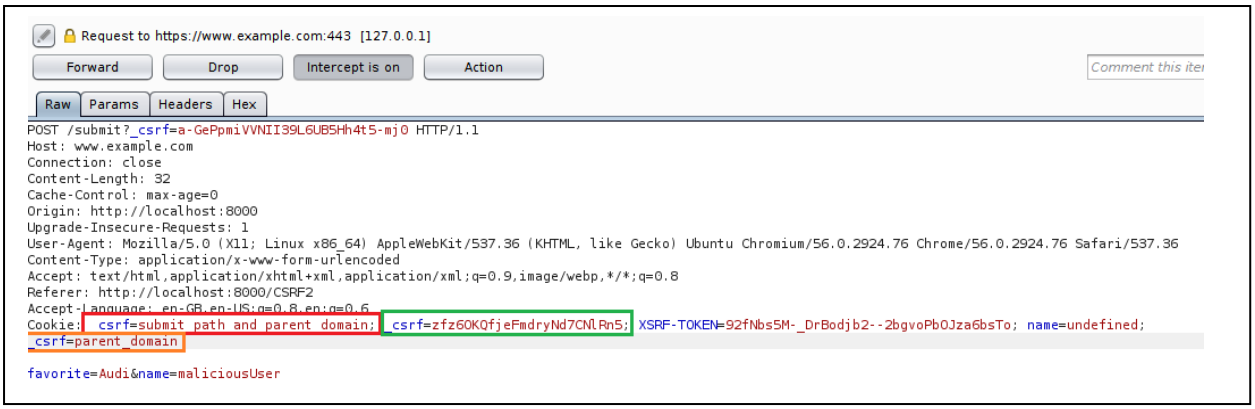

Figure 2.8: Example of CSRF cookie sent to the parent domain.

9. CSRF countermeasures discussion

As we have presented various defensive mechanisms against CSRF, now we discuss those mechanisms. The basic idea behind the Server side protection techniques is that server can strip authentication credential and session information from suspected requests or it can refuse such requests. A validation token is the most straight forward protection technique used by web server developers. In this approach, web forms are augmented with a server generated, unique validation token (e.g. embedded as a hidden field in the form), and at form submission the server checks the validity of the token before executing the requested action [27]. At the client-side, validation tokens are protected from cross-origin attackers by the same origin policy, distinguishing them from session cookies or authentication credentials that are automatically attached to any outgoing request. Such a token-based approach can be offered as part of the web application framework [38], as a server-side library or filter, or as a server-side proxy. Recently, the Origin header has been proposed as a new server-side countermeasure. With the Origin header, clients unambiguously inform the server about the origin of the request (or the absence of it) in a more privacy friendly way than the Referer header. Based on this origin information, the server can safely decide whether or not to accept the request. In follow-up work, the Origin header has been improved, after a formal evaluation revealed a previously unknown vulnerability [19]. Although, the disadvantage of server-side countermeasures is that they require modifications of server-side programs and making the adoption rate of these protection mechanisms slow. As result, client-side mitigation techniques can be important solution. As stated before, the client-side countermeasures are implemented in form of browser proxy or extension to web browsers. This mechanism makes the web browser at first line defense and reduces the overtime efforts of web servers. A policy used by the client side protection techniques is to block all cross-

domain traffic; this is undoubtedly the most secure policy. But unfortunately, many websites need cross domain operation to perform. So, the key challenge in those mechanisms is to distinguish malicious from non-malicious cross-origin requests. As result, the most existing countermeasures do not handle current technologies well, such as single sign-on (SSO). Furthermore, the HTTP protocol specification states that GET requests should not have state-altering effects. Even though, many HTTP traffic analysis has shown that cross-domain GET requests are very common, this means that the attack vector is quite large too. However, they do not have much importance in client side protections. Besides to this and as we mentioned before, CSRF attack also can be carried out using XSS attacks and the maximum protection mechanism suggested cannot protect users against CSRF in this case. The existing client countermeasures also they don't take this fact in consideration, which makes web applications and users not completely protected from CSRF. For those reasons, we will implement in our work a ’CSRF Detector’, a browser extension to defeat attackers from attempt to perform CSRF attacks by detecting all basic and advanced attacks techniques.

10. Conclusion

In this chapter, we talked about many defense techniques that have been proposed to protect web users against CSRF attacks in both server and client-side. We have seen several client-side mitigations techniques to combat CSRF attacks and we discussed their limits. In the next chapter, we will see our solution to defeat attackers from the attempt to perform CSRF attacks by detecting all basic and advanced CSRF attacks techniques.

CHAPTER 03

CSRF DETECTOR EXTENSION

CHAPTER 3
CSRF DETECTOR EXTENSION

1. Introduction

In the previous chapter we discussed the previous works that have been done to defend against CSRF attacks and their limits. In this chapter we will talk about our proposed tool to defeat attackers from attempt to perform basic and advanced CSRF attacks.

2. The proposed scheme

For the client-side protection, the web browser is the right place to apply appropriate protection mechanism against CSRF. We have seen in the previous chapter the existing client-side countermeasures and we have cited two main approaches used to protect users from CSRF attacks. The first one relies on examining and analyzing the request to identify the cross-domain requests. The cross-domain request will be either blocked or stripped from authentication credential and session information. But blocking all cross domain requests in contrast to stripping credentials can have a very noticeable effect on the user experience. The general limit of this approach is that can only deal with one type of CSRF attack which is reflected CSRF. The second approach can protect users from CSRF by checking the content of the response web page in order to detect the symptoms of CSRF attack. In this approach, the malicious and benign cross-domain requests are distinguished from each other based on the existence and visibility of the submitted form or link on the originating page. The advantage of this approach is that can protect users from both types of CSRF attack (reflected and stored). But unfortunately, the approach suffers from more false positive, the genuine cross-domain requests will be falsely detected as CSRF attacks.

In order to combat CSRF attacks, we proposed a novel technique for protecting users from basic and advanced CSRF attacks at client-side.

Our CSRF detector has the following characteristics:

- Our proposed tool ‘CSRF Detector’ detect CSRF attacks with a hybrid mechanism based on the notion of content checking and request analysis. This approach does not rely on server side program states and it does not require storing URLs or tokens to be matched at a later stage for attack detection. Our CSRF detector analysis the HTML code before each page loads and detects potential CSRF attack, and it intercepts every HTTP request and decides

whether it should be allowed or not. The suspicious request event will strip from the authentication credential and session information.

- In order to combat Image based CSRF Attacks we propose a mechanism to identify of such suspicious image elements by obtaining all IMG elements from the response page and matched the image URL linked with the image file formats (JPG, Gif …etc).
- CSRF detector can detect XSS attack in web site for the purpose of preventing attackers from using such vulnerability to carried out the CSRF attacks.
- Our proposed solution could defend against login CSRF attack because all processes of checking are done before the login process.
- CSRF Detector has the ability to detect both types of CSRF attack (reflected and stored).
- CSRF Detector has the ability to detect cross and sub-domain CSRF attack.
- CSRF Detector adopts **Google Safe Browsing** service to identify unsafe websites and notify users to protect themselves from harm websites (malware, phishing …).
- Finally, CSRF Detector also serves as a tool for web developers to detect CSRF vulnerability. It allows developers to quickly discover forms that are likely unprotected against CSRF attacks.

2.1. Considerations of web 2.0

The difficulty of preventing CSRF cannot necessarily be contributed to the nature of the attack, but more to the complex traffic patterns that are present in the modern web 2.0 contexts. Especially sites extensively using AJAX and single sign-on (SSO) mechanisms, which combines content of multiple different Websites, make it hard to distinguish intended user traffic from unintended or malicious traffic. A SSO session typically uses multiple redirects to go from the site the user is visiting to an SSO service. During these redirects, authentication tokens are exchanged. When the original site receives a valid authentication token, the user is authenticated. Since all these redirects are usually cross-domain, no cookies or HTTP authentication headers can be used anyway, due to the SOP restrictions implemented in browsers. The authentication tokens are typically encoded in the redirection URLs [27]. Web 2.0 techniques such as AJAX and SSO can be dealt with appropriately since our solution does not block any cross-domain request. By stripping the suspicious cross-domain request from the authentication credential and session information, CSRF Detector is able to deal with these multiple redirects and does not break SSO sessions and has no degrading effect on websites using AJAX and SSO.

3. CSRF Detector architecture

The figure 3.1 shows the flowchart of the act of CSRF Detector, which is a web extension, locates on Google chrome browser, to monitor any interaction between web sites and web client. CSRF Detector **analysis web requests** and **web pages content** to detect all basic and advanced CSRF attacks.

CSRF detector also adopted **Google Safe Browsing** service to check in really big global database offered by Google that contains a reported malicious website (Malware, fishing, social engineering).

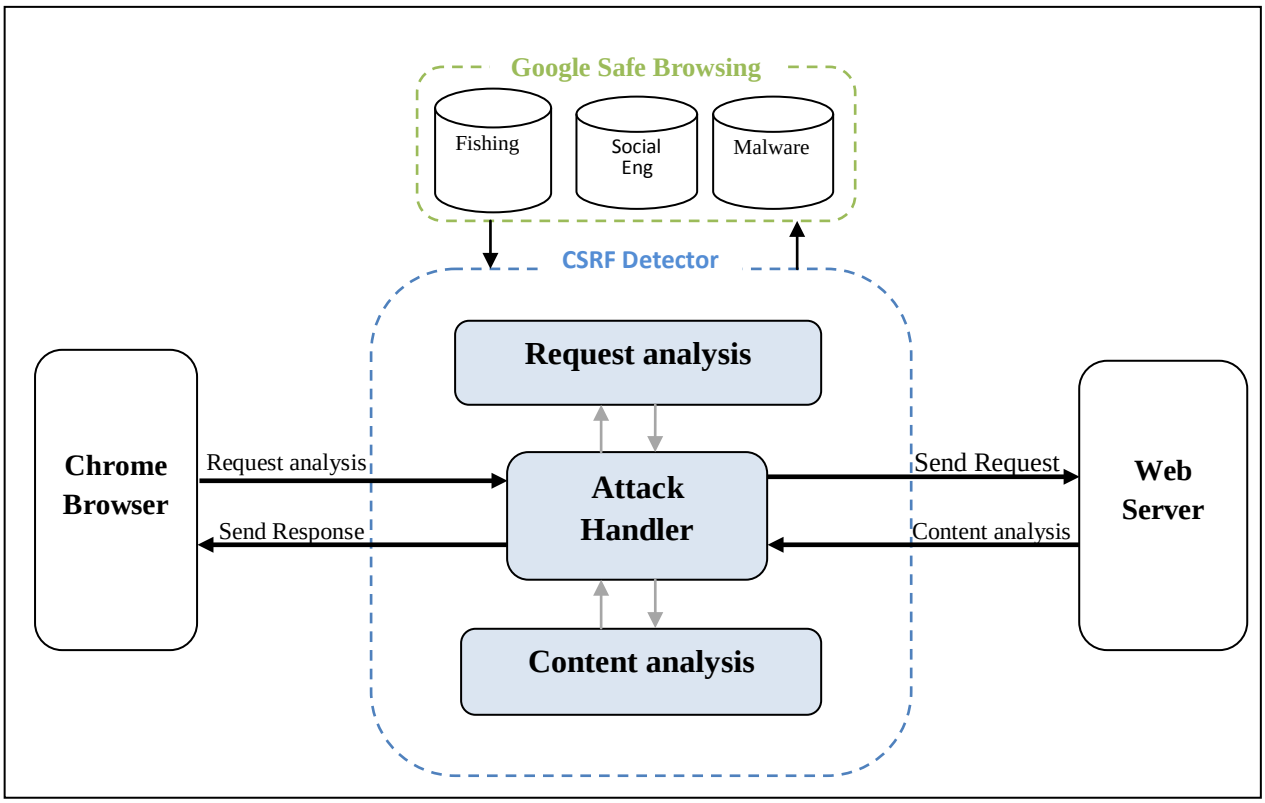

Figure 3.1: CSRF Detector mechanism.

CSRF Detector has three modules:

1- **Request analysis Module:** CSRF Detector will intercept the outgoing request sent by the web browser and check for :
   a. Suspicious Cross-domain requests.
   b. Suspicious sub-domain requests.
   c. Suspicious reflected XSS (Cross-site scripting) requests.
   d. Check URL in Google Safe Browsing.

For the first and the second type of requests (Cross domain requests and Suspicious sub-domain requests), CSRF Detector intercepts the outgoing requests sent by the web browser and determines the target and the origin domain. If the origin domain does not match the destination of the request, the request is considered suspicious. Suspicious requests will be stripped of authentication credentials in the form of cookies or HTTP authorization headers. There are two steps to this check:

1. Determining the origin of the request is coming from (**Origin domain**).
2. Determining the target of request is going to (**Destination domain**).

For the third type of requests (reflected XSS requests), our CSRF Detector intercepts all requests and looks for cross-site scripting signatures within parameter values. To look for signatures in parameters values, our CSRF detector must parse the URL correctly and retrieve the value part and then search for the signature on the value while overcoming encoding issues.

2- Content analysis Module: Before and after the requested web page is loaded, CSRF Detector will check for:

a. Invisible iframes.
b. Suspicious image elements.
c. Unprotected forms.

After webpage is loaded, users can checks for any CSRF vulnerabilities. The module will check for HTML forms that are likely unprotected against CSRF attacks by examining forms on a webpage and matching the form elements with user-specified CSRF Token names and formats.

3- Attack Handler Module: This module sanitizes all suspicious requests, and generates an alert message to the user about all different types of CSRF attack. The module will strip the suspicious request from authentication credential and session information, or any suspicious XSS code.

4. CSRF Detector mechanism

In this section we will explain how our CSRF Detector works in its different stage:

4.1. Request analysis

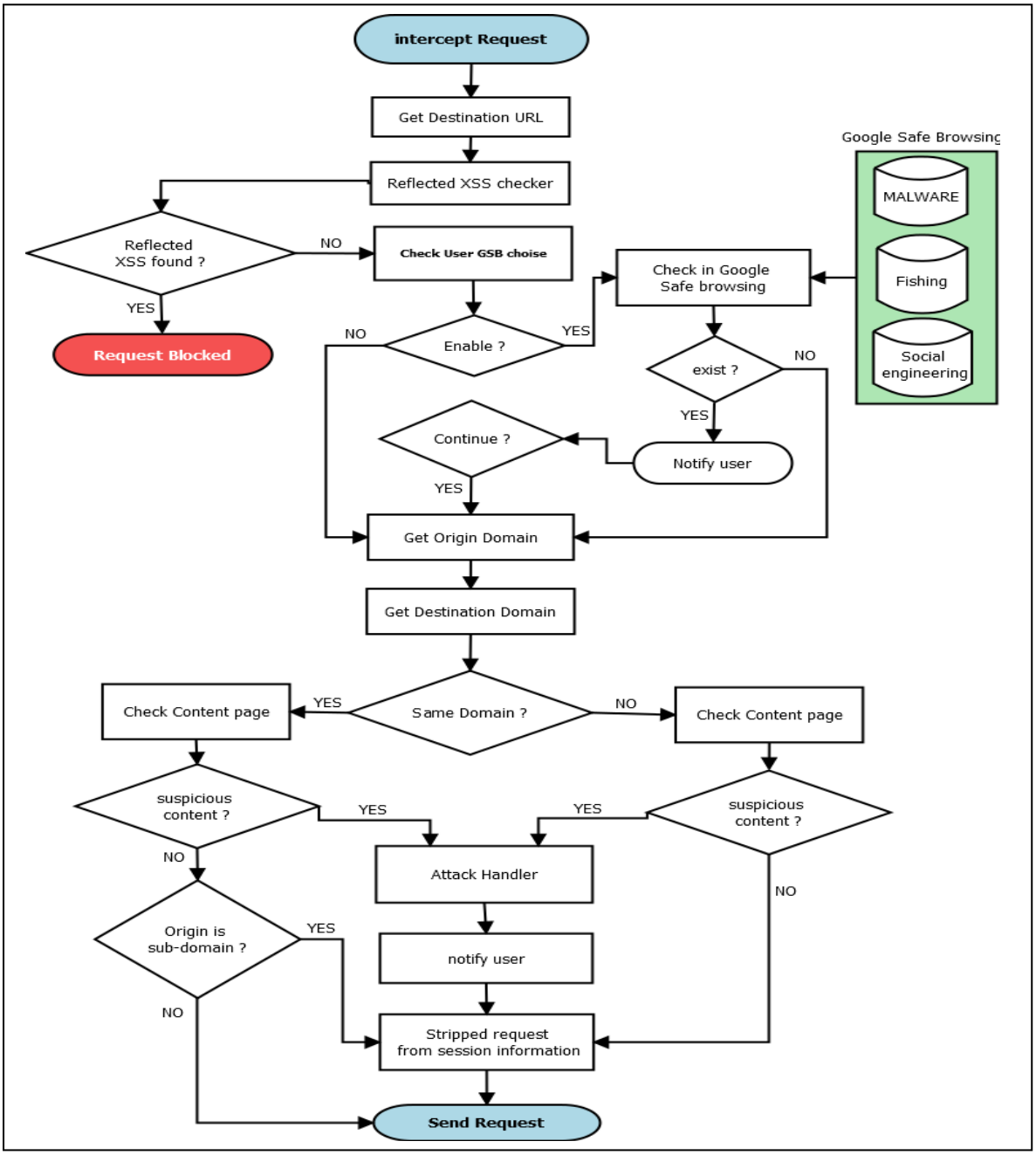

Figure 3.2: Request analysis flowchart.

In **Figure 3.2**, we illustrate how CSRF Detector intercepts every HTTP request sent by the browser and decides whether it should be allowed, blocked or stripped from authentication credential and session information. First of all, CSRF Detector will get the

requested URL and check for reflected XSS (Cross-site scripting attack) using reflected XSS checker.

In our case with Cross-site scripting ('XSS') attack, the hacker places a hyperlink with an embedded malicious script into an URL address. The purpose of the malicious script is to steal user session token and then send it to the attacker. For this reason our detector module matches any XSS signature (for example keyword: <script> ,"%3Cscript%3E"…) in the requested url, blocks any suspecious request and notify the user (**Figure 3.2**).

In the second phase, the requested URL is checked in **Google safe browsing** databases if the user was enabling this service. Enabling Google Safe Browsing will offer users more protections. After that, CSRF Detector will check for cross-domain requests and requests event that originating from sub-domain to parent domain, this two events are considered suspicious and will be stripped from session information.

4.1.1. Reflected XSS checker

This module checks for reflected XSS attack in the requested URL. CSRF detector can detect reflected XSS for the purpose of preventing attackers from using such vulnerability to carried out the CSRF attacks. XSS exploit is carried out using of <script> tags and using of characters such as “<>”. In some cases, it is possible that signature-based filters can be simply defeated by obfuscating the attack. Typically the attacker can do this through the insertion of unexpected variations in the syntax or in the encoding. These variations are tolerated by browsers as valid HTML when the code is returned [43]. Example of different syntax or encoding is shown in the boxes below:

```
<ScRiPt>alert(document.cookie)</ScRiPt>
```

```
<script >alert(document.cookie)</script >
```

```
%3cscript%3ealert(document.cookie)%3c/script%3e
```

Our module of reflected XSS checker will check for a different character that can be used to carry out the reflected XSS attack. Here are the characters it may exist in URL to detect as reflective XSS attack:

{ '%3c', '%3e', '<', '>' }

In case the module match this characters in the requested URL, the request will be blocked and send to attack handler to notify the user.

4.1.2. Google Safe Browsing checker

Safe Browsing is a Google service that lets client applications check URLs against Google's constantly updated lists of unsafe web resources. Unsafe web resources are social engineering websites (phishing and deceptive sites) and websites that host malware or unwanted software.

CSRF detector helps protect users by showing warnings to users when they attempt to navigate to dangerous sites. With Safe Browsing we can:

- Check web pages against Google Safe Browsing lists based on platform and threat types.
- Warn users before they navigate a harm websites.

4.1.3. Cross-domain request checker

To check for cross-domain request, CSRF Detector will determine the origin domain and destination domain by:

- First step, Get all existing tabs. Then, identify the valid tab ID of the triggered request and extract the URL of the considered tab ID to determine the origin domain of the request. This step is important to specify the request source whether comes from different tab or from the same one of a valid user.
- Second step, is to determine the destination domain by extracting the value of 'HOST' header of the triggered request.

After CSRF Detector determine the target and the origin domain, we will check if the request is under the same domain. when the origin domain does not match the destination of the request, the request will be stripped from authentication credentials in the form of cookies or HTTP authorization headers. The advantage of stripping is that there are no side-effects for cross-domain requests that do not require credentials in the first place.

4.2. Content analysis

Figure 3.3 shows the flowchart of web page response analysis, and how it works when the response webpage is loaded. This module will check for invisible iframes and suspicious image elements.

We integrate in this module CSRF vulnerability checker to notify users if any unprotected forms are found, and help developers to check their web sites for CSRF vulnerabilities.

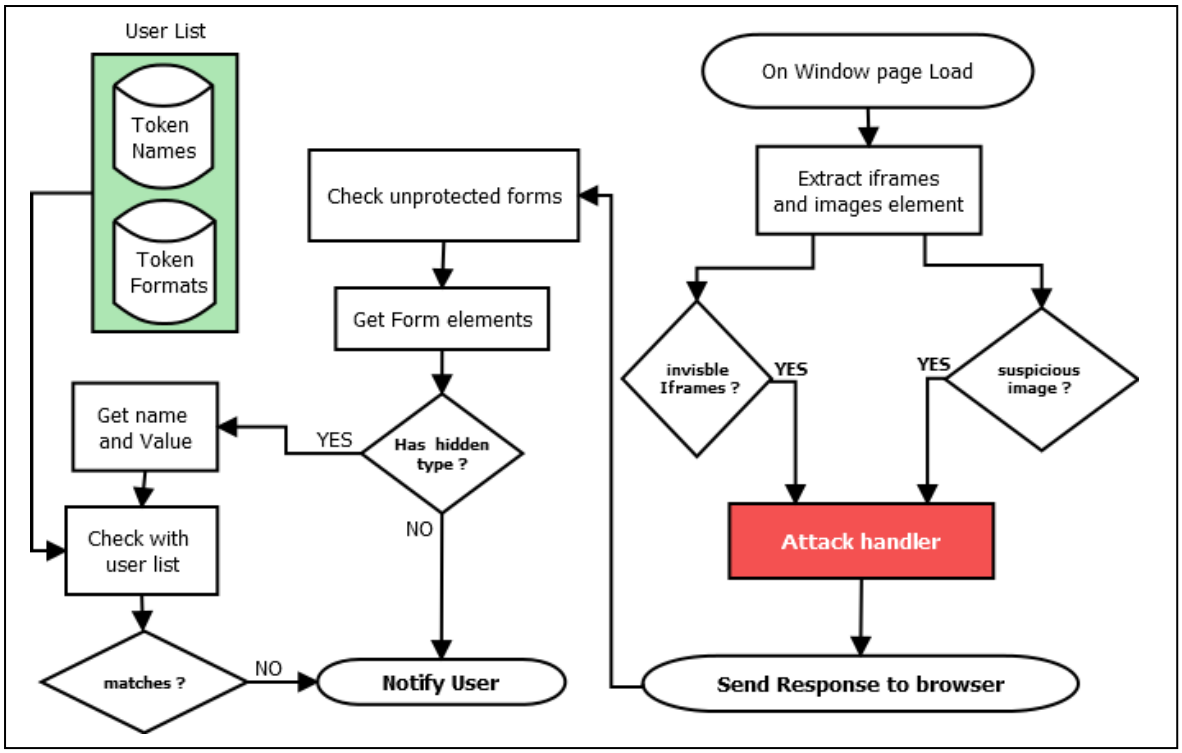

Figure 3.3: Flowchart of content analysis.

4.2.1. Invisible iframes checker

The **Figure 3.4** shows the flowchart and demonstrates the working module for this stage of checking the content.

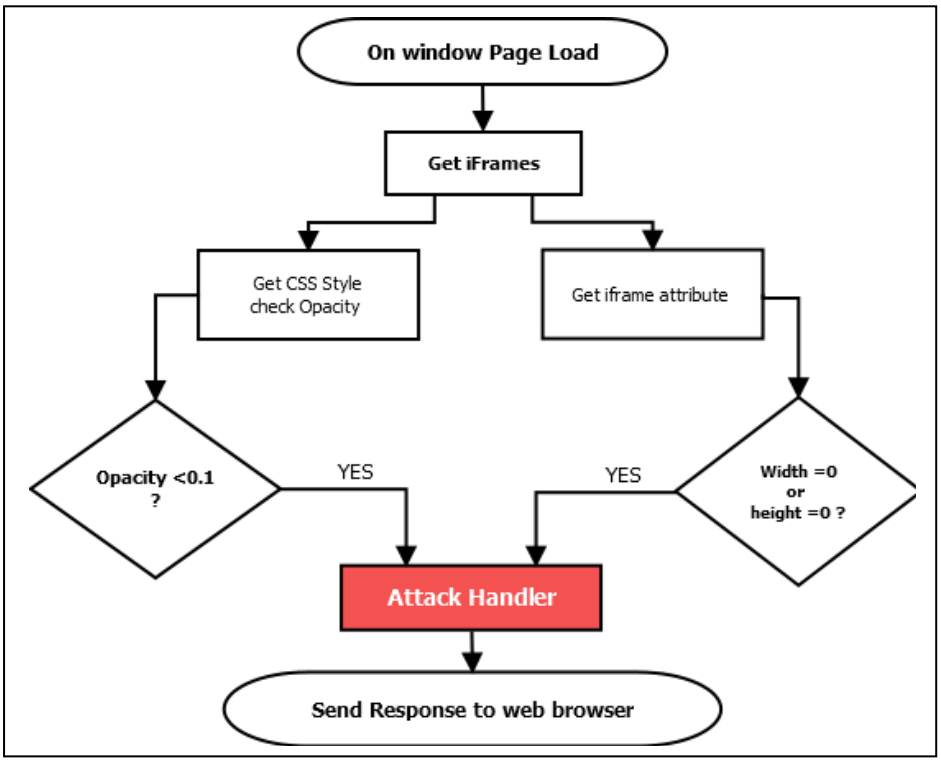

Figure 3.4: Flowchart of invisible iFrames checker.

CSRF detector will check for invisible iframes when the webpage is loaded in order to prevent attackers to perform the invisible iframe-based CSRF attack. The module will extract all iframes exist in a webpage, check CSS style and the attributes of HTML element iframe (“width” and “height”). If (opacity<0.1) or one of the iframe attribute has the value equal to zero. The web page will be consider having the symptoms of CSRF attack. The attack handler will strip the next requests event from authentication credential and session information.

4.2.2. Suspicious image elements checker

In order to combat Image-based CSRF Attacks, we propose a mechanism to identify of such suspicious image elements by obtaining all IMG elements from the response page and matched the image URL linked with the image file formats (JPG, Gif …etc).

The major of the websites use the static image URLs to obtain the images in their web pages. The widely recognized image formats include GIF, JPEG, and PNG. The other formats supported by popular browsers are jpg, bmp, xbm. So if the website uses a static image URL then the URL must end with gif, jpeg, jpg, png, bmp or xbm extension. We can successfully mitigate the CSRF attacks using static image URL by checking the URL extensions and validating the URL specified in all images element before the browser renders the webpage.

4.2.3. Unprotected forms checker

CSRF Detectorcan help web developers to quickly discover forms that are unprotected against CSRF attacks. The CSRF token is added as a hidden field to forms. The hidden field input is a parameter with a common name such as "CSRF_Token". The value of this token is randomly generated by the server, such that it cannot be guessed by an attacker. The box below shows an example of how CSRF Token added in forms.

```
<form action="/transfer.php " method="post">
<input type="hidden" name="CSRFToken" value="OWY4NmQwO"
</form>
```

CSRF detector can check for unprotected forms by examining forms on a webpage and matching the form element of the hidden type against a list of user-specified token names and formats. Regular expressions are supported to specify both token names and formats.

- A **token name** is the name of a hidden element that contains a CSRF security token.
- A **token format** is the format of the value of a CSRF security token.

5. Conclusion

In this chapter, we have shown the architecture of our CSRF Detector extension and its techniques to defeat and prevent the attackers to perform the CSRF attacks, in the next chapter we will show how we will implement and experiment our CSRF Detector.

CHAPTER 04

IMPLEMENTATION AND EXPERIMENTATION

CHAPTER 4

IMPLEMENTATION AND EXPERIMENTATION

1. introduction

This chapter describes the experimental study of our implemented extension. In which our goal is implementing a browser extension to detect basic and advanced CSRF attacks.

We conducted a number of experiments to study and measure the performance of our extension, to achieve this objective, our extension CSRF is tested with 300 pages of white list Websites, extracted from Alexa top 1500 websites to identify false positive generating by CSRF Detector,

We also developed a vulnerable web application and malicious websites which perform CSRF attacks in order to measure the false negative detection rate of our extension. We also have discovered CSRF vulnerabilities on 3 real life Algerian famous web sites; we apply our CSRF detector to identify its detection rate.

2. implementation tools

2.1. Google Chrome Browser

Chrome is freeware web browser developed by Google in 2008. Released for windows then it ported to Linux, macOS, ISO, and Android.

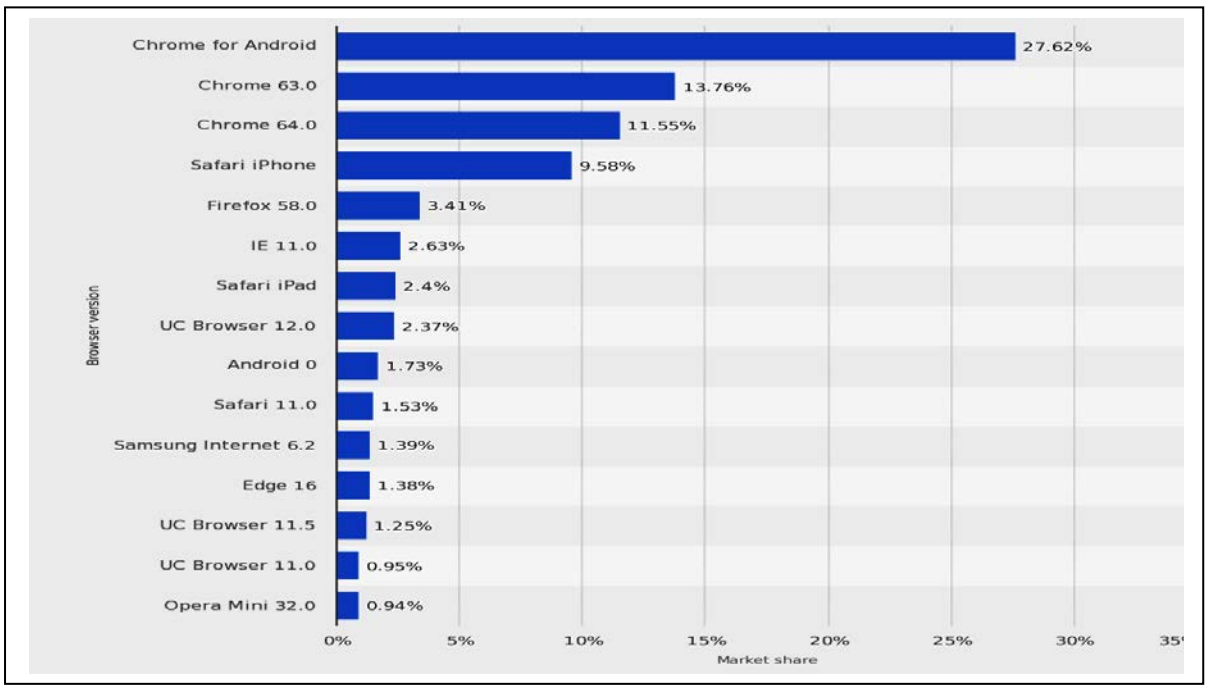

Figure 4.1: Most popular web browser (February 2018) [39].

Currently, Google Chrome is the most popular and widely used browser. As is illustrated in **figure 4.1**, this statistic ranks the most popular web browser versions as of February 2018, ranked by their share of total page views. Google's Chrome 63 was the second most popular browser with a market share of 13.76 percent, behind Chrome for Android at 27.62 percent.

2.2. Web browser extension or add-on

Extensions are small software programs that customize the browsing experience [40]. They enable users to tailor Chrome functionality and behavior to individual needs or preferences. They are built on web technologies such as HTML, JavaScript, and CSS. Chrome extensions have a strong focus on extending the browser's functionality. It is used for different reasons and extension types are assigned different tasks.

2.3. Extension Architecture

Extensions are made of different components. Components can include background scripts, content scripts, an options page, UI elements and various logic files. Extension components are created with web development technologies: HTML, CSS, and JavaScript. **Figure 4.2** shows the different parts of Chrome extension.

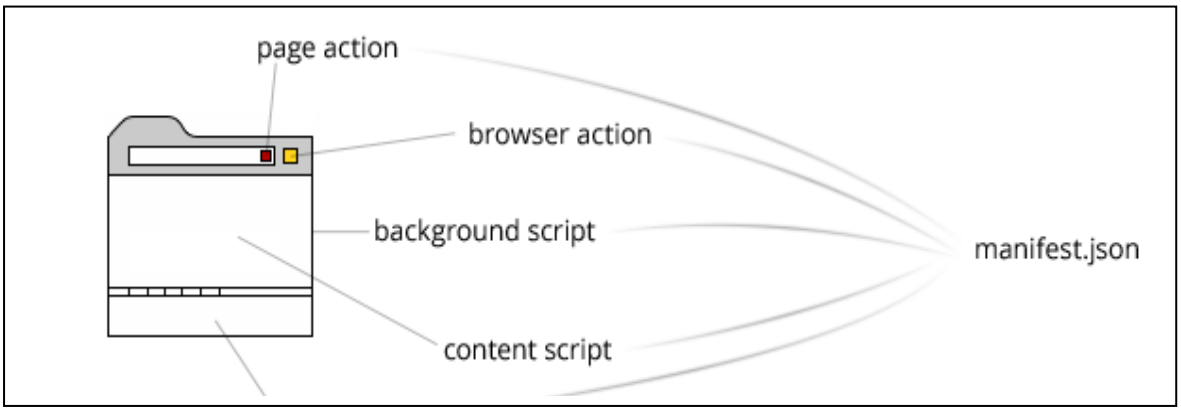

Figure 4.2 Google chrome extension architecture system [40].

2.3.1. Manifest file

Every extension has a JSON-formatted manifest file, named **manifest.json**. The manifest file gives information about the extension, such as the most important files and the capabilities that the extension might use. It should at least contain the bellow JSON Objects shown in **figure 4.3**.The file could also contain other properties depending on what kind of extension.

```
{
   "name": "Extensionname",
   "version": "0.0.1",
   "manifest_version": 2,

"browser_action": {
      "default_title": "That's it",
      "default_popup": "popup.html"
   }
   "background": {
   "scripts": ["background.js"],
   "persistent": false/true
   },
   "content_scripts": [
   {
      "matches": ["http://*/*", "https://*/*"],
      "js": ["content.js"]
   }
   ]
}|
```

Figure 4.3: Manifest file format.

2.3.2. Background Script

Every extension has an invisible background page which is run by the browser. It plays the role of a bridge between the other parts of the extension.

The Background Script has two types defined by the persistent attribute:

1. **Persistent background pages:** active all of the time (persistent attribute is true).
2. **Event page:** active only when it needed (persistent attribute should be false in that case).

Background scripts are registered in the manifest under the "background" field. They are listed in an array after the "scripts" key, and the attribute "persistent" should be specified as illustrated in **figure 4.4** below.

```
{
  "name": "Awesome Test Extension",
  ...
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  ...
}
```

Figure 4.4: Example of Background scripts registered in a manifest file.

2.3.3. Content Script

The Content script is the only script could access and made changes to the current page's DOM. The code is run on the current page and executed with every refresh.

2.4. Manage Events

Extensions are event-based programs used to modify or enhance the Chrome browsing experience. Events are browser triggers [41]. Such as navigating to a new page, removing a bookmark, or closing a tab. Extensions monitor these events in their background script and then react with specified instructions.

2.4.1. The Life cycle of requests

The web request API defines a set of events that follow the life cycle of a web request. These events can be used to observe and analyze traffic. Certain synchronous events will allow us to intercept, block, or modify a request. The event life cycle for successful requests is illustrated in **figure 4.5** below.

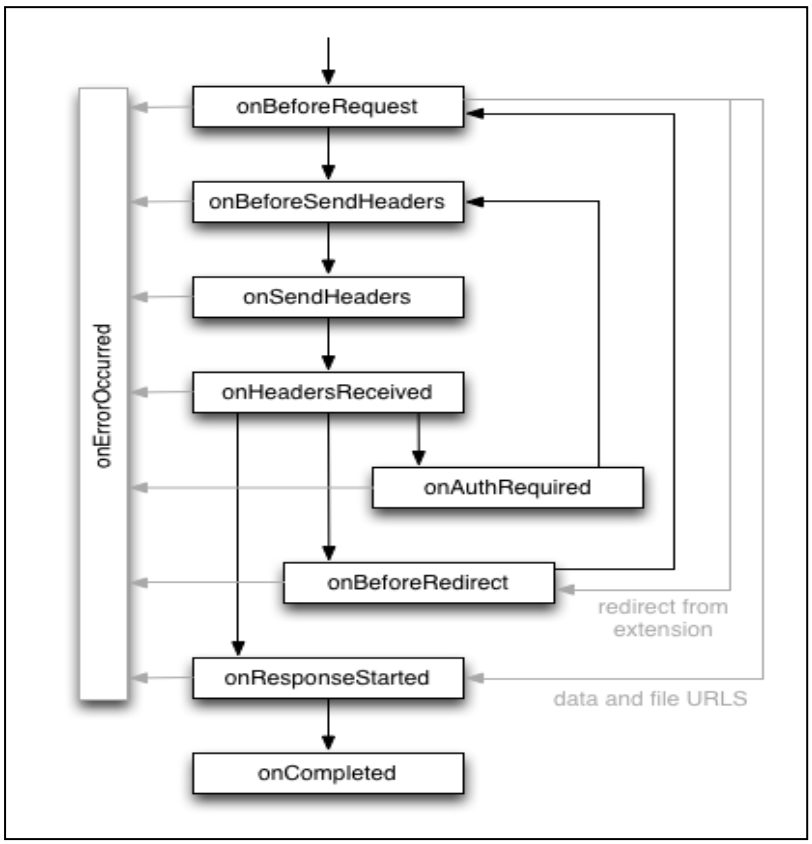

Figure 4.5: The event lifecycle for request [41].

2.4.2. onBeforeSendHeaders event

onBeforeSendHeaders event Fires when a request is about to occur and the initial headers have been prepared [41]. The event is intended to allow extensions to add, modify and delete request headers. CSRF Detector uses this event to strip cookies from the suspicious request.

2.5. Google Safe Browsing API

The Safe Browsing APIs (v4) let our extension CSRF Detector to check URLs against Google's constantly updated lists of unsafe web resources. Any URL found on a Safe Browsing list is considered unsafe. To determine if a URL is on any of the Safe Browsing lists, clients can use either the **Update API (v4)** or **Lookup API (v4)**.

2.5.1. Update API (v4)

The Update API lets client applications download encrypted versions of the Safe Browsing lists for local, client-side checks of URLs. The Update API is designed for clients that require high frequency, low-latency verdicts. Several web browsers and software platforms use this API to protect large sets of users.

2.5.2. Lookup API (v4)

The Lookup API lets client applications send URLs to the Google Safe Browsing server to check their status [44]. The API is simple and easy to use, as it avoids the complexities of the Update API.

In CSRF Detector we have used Lookup API (v4), which implements a simple URL check, by sending an HTTP POST request with the actual URLs, and the server responds with the state of the URLs (safe or unsafe).

To use the Lookup API we need a Google Account, a Google Developer Console project, and an API key. Finally, we also need to activate the Safe Browsing APIs for use with our project.

2.5.3. Set up an API key

To access to the safe browser APIs we set up and activate our own API key to authenticate as an API user to allow us to interact with the APIs. The API key is passed as a URL parameter in the HTTP request to the Safe Browsing service. Here the URL with our API key:

```
https://safebrowsing.googleapis.com/v4/threatMatches:find?key=AI**HC8
```

2.5.4. Checking URLs

To check if a URL is on a Safe Browsing list, we send an HTTP POST request to the **Google Safe Browsing** server. The requestis an XMLHttpRequestand has the flowing structure :

- The request header includes the request URL and the content type.
- The request body includes the client information (ID and version) and the threat information (the list names and the URLs).

Figure 4.6 below shows the code of HTTP post request:

Figure 4.6: Lookup API HTTP post request code.

The HTTP POST response will include the response header and a body in case matches. If there are no matches (that is, if none of the URLs specified in the request are found on any of the lists specified in a request), the HTTP POST response simply returns an empty object in the response body.

- The response header includes the HTTP status code and the content type.

```
HTTP/1.1200 OK
Content-Type:application/json
```

- The response body includes the match information (the list names and the URLs found on those lists, the metadata, if available, and the cache durations).

```
{
  "matches":[{
    "threatType":      "MALWARE",
    "platformType":    "WINDOWS",
    "threatEntryType":"URL",
    "threat":          {"url":"http://www.urltocheck1.org/"}
,
    "threatEntryMetadata":{
      "entries":[{
        "key":"malware_threat_type",
        "value":"landing"
     }]
    },
    "cacheDuration":"300.000s"}
```

Figure 4.7: example code of HTTP post response body

3. Interfaces

3.1. CSRF Detector user interface

CSRF Detector is an extension that works on browser background with a simple interface. The user can decide whether to use or not the extension by simply check the button Disable. The suspicious request which stripped from cookies will be displayed in the middle gray box. Web developers also can check for unprotected forms against CSRF by simply click on the button of checking.

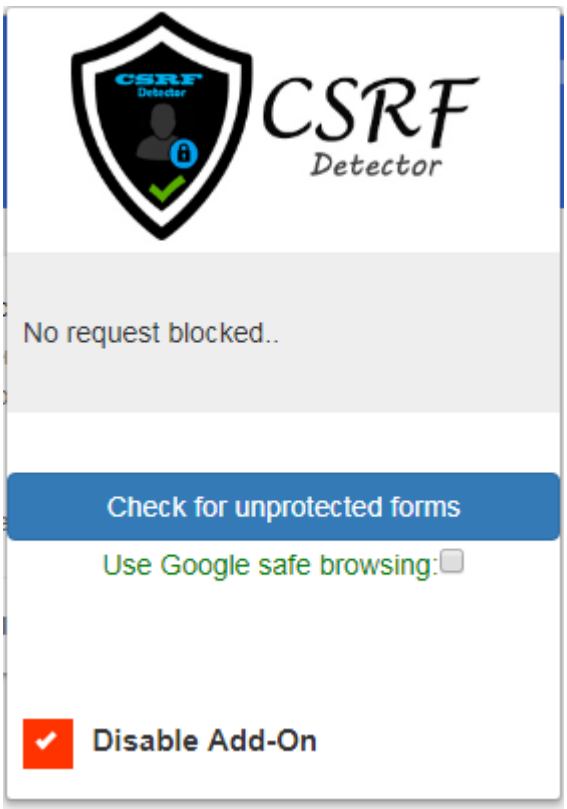

Figure 4.8: CSRF Detector user interface.

3.2. CSRF Detector alerts

CSRF Detector generates alerts to notify users about all types of attack that our tool can identify. The flowing alerts illustrate cases when the users alerted:

- **Image-Based CSRF attack:** italerts a user when suspicious image found **.**

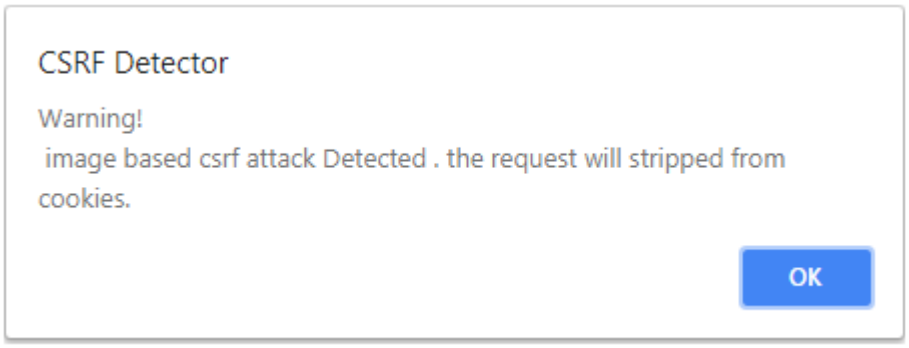

Figure 4.9: alert onImage-based CSRF attack.

- **Invisible-iframe Based CSRF attack :** italerts a user when invisible iframe found.

Figure 4.10: alert on the invisible iframe.

- **Reflected XSS alert:** it alerts a user while the requested URL contains symptom of reflected XSS attack

Figure 4.11: Alert on reflected XSS attack.

4. Test Methodology

In order to ensure the efficiency of our solution, we conducted some tests to evaluate the efficiency and the capability of the CSRF Detector against basic and advanced CSRF attacks. To make sure that its results match with what is predetermined we developed a small web application which implements cookies as session management mechanism and simulates the CSRF attacks with it. We also tested our solution in Real-life with vulnerable and white list websites.

4.1. Testing with basic CSRF attacks

E-Money transfer application is small web application written in PHP, HTML, JavaScript, and MySQL, we developed this web application to simulate the basic CSRF attacks and to evaluate the ability of CSRF Detector to protect vulnerable applications. The users of this application can login to the application and send money to each other. The form of sending money contains information (Receiver and amount), the user had to login and fill the form to send money. In purpose to simulate the CSRF attack we deliberately left this form unprotected and use it later at different stages of CSRF attack. We checked the form by CSRF Detector and we get the result as illustrated in the figure below.

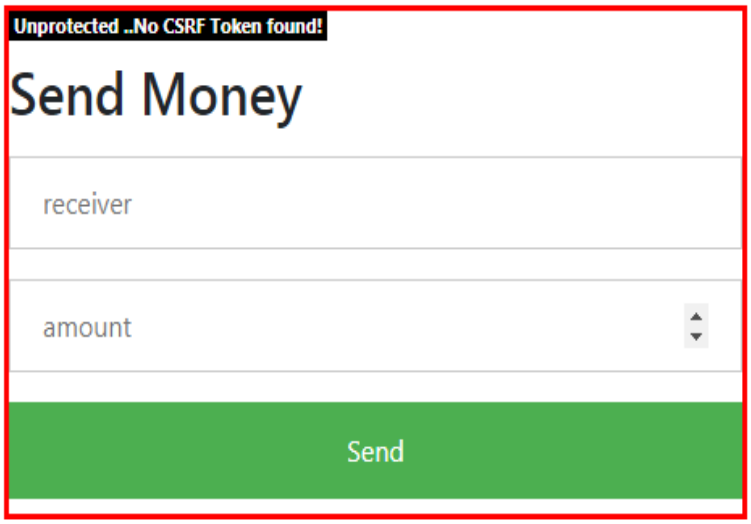

Figure 4.12: Unprotected form checked by CSRF Detector.

4.1.1. Test scenario

- **Case 1:** Without CSRF Detector

The victim login to E-money transfer website. After that, he clicks on a malicious link sent by an attacker or malicious link stored in attacker website. Because there is no detection against CSRF attack in this website, the attackers can easily trick the

user through the malicious link and send money to his account without knowledge of the victim.

- **Case 2:** With CSRF Detector Extension

  We enable CSRF detector. Then, we repeated the previous attacks. Our tool detected and rejected all CSRF attacks attempted from different sources correctly, which means that the **false negative** rate is zero.

4.2. Testing with advanced CSRF attacks

We conducted some tests to evaluate the efficiency and the capability of our tool against advanced CSRF attacks including both vulnerabilities (XSS and exploiting sub-domains). At first, we developed a webpage that contains malicious XSS. The box below shows how the malicious code was injected.

```
<form method="post" action="http://localhost/csrf/money/index.php<script>alert('xss');</script>" target="dummyframe">
   <input type="hidden" name="receiver" placeholder="receiver" type="text" value="bob">
        <input type="hidden" name="amount" placeholder="amount" type="number" value="100">
        <input name="submit" value="download" type="submit">
</form>
```

After the request is triggered from this form, CSRF Detector shows his ability to block such malicious request and alert user about it. We also test CSRF Detector with request that originating from sub-domains to parent domain. The figure below shows that CSRF detector was successfully stripped cookies from such request.

4.3. Testing with white list Websites

We conducted an experiment using a white list to check if our proposed tool will generate any warning on possible CSRF attack and identify its false positive rate. The experimental considering 300 websites belonging to different rank ranges (computer, science, News... etc) of the Alexa global top 1500.

The results of this experimental shown that we got no false alert which means **no false positive** over 300 websites visited.

The table below shows the characteristics of some websites we visited from white list:

Websites	Number of request	Cross-domain Request/ alerts	Numberof images / alerts	Number of iframes / alerts
Facebook.com	475	82/0	167/0	6/0
Twitter.com	142	24/0	83/0	0/0
Amazon.com	300	33/0	222/0	5/0
Yahoo.com	149	101/0	24/0	5/0
Microsoft.com	47	20/0	19/0	4/0
paypal.com	50	49/0	18/0	0/0
Imdb.com	90	45/0	62/0	4/0
Cnn.com	150	63/0	47/0	8/0
Sciencedirect.com	34	17/0	7/0	2/0
Researchgate.net	57	12/0	17/0	1/0

Table 4.1: Alexa top websites in different rank ranges.

4.4. Testing with real-life vulnerable websites

To test our solution with real-life websites, we used at first our tool CSRF detector to discover unprotected forms on random 100 Algerian websites from Alexa top websites in order to find vulnerable websites. We found some famous Algerian websites are vulnerable to CSRF attacks, including a serious vulnerability that can change credentials information such as email. But we can't give their URL's in this report for security reason. Then, we used Burp 'CSRF PoC generator' [46] which is tool used to generate a proof-of-concept (PoC) cross-site request forgery (CSRF) attack for a given request. **The result of this experiment shows that CSRF Detector is successfully detects and prevents every attack generated by Burp CSRF PoC.**

5. Chrome extensions for combating CSRF

According to various statistics shown previously, Google Chrome is the most popular Web browser in the world, being used by well more than 50% of Internet users. For that reason, our comparison study has focused on identifying and analyzing anti-CSRF extensions for this particular browser. Out of thousands of currently available Chrome extensions, we were able to identify (only) five that were claiming and/or appeared to provide protection against CSRF attacks, their names are enlisted in the left-most column in **Table 4.2**. All five extensions were tested against several different variants of basic and advanced CSRF attacks and the first variant of the CSRF attack included obfuscated/malicious requests that were hidden using the standard <iframe> and <img> approach.

	CSRF <iframe>	CSRF <img>	Reflected XSS	Stored CSRF	Reflected CSRF	Check vulnerability	No False positive	Google Safe browsing
CSRF discover [47]	X	X	X	X	X	√	X	X
Csfire [27]	X	X	X	X	√	X	√	X
Scriptsafe [49]	√	X	√	√	X	X	√	X
Scriptblock [50]	X	X	√	X	X	X	X	X
CSRF Blocker[29]	√	√	√	X	√	X	X	X
Our tool	√	√	√	√	√	√	√	√

Table 4.2:comparison summary of Google Chrome extensions for combating CSRF.

Key:

√: plug-in successfully blocked the attack &/or “affirmative”.

X: plug-in did not successfully block the attack &/or “negative”.

6. Performance

We visited several websites from white list sites with and without using CSRF Detector extension, and then we have used Google chrome Analyzing Performance to measure webpage loading time. Figure 4.13 shows line chart of the average delay time of loading webpage (in seconds) measured by webpage size (bytes).

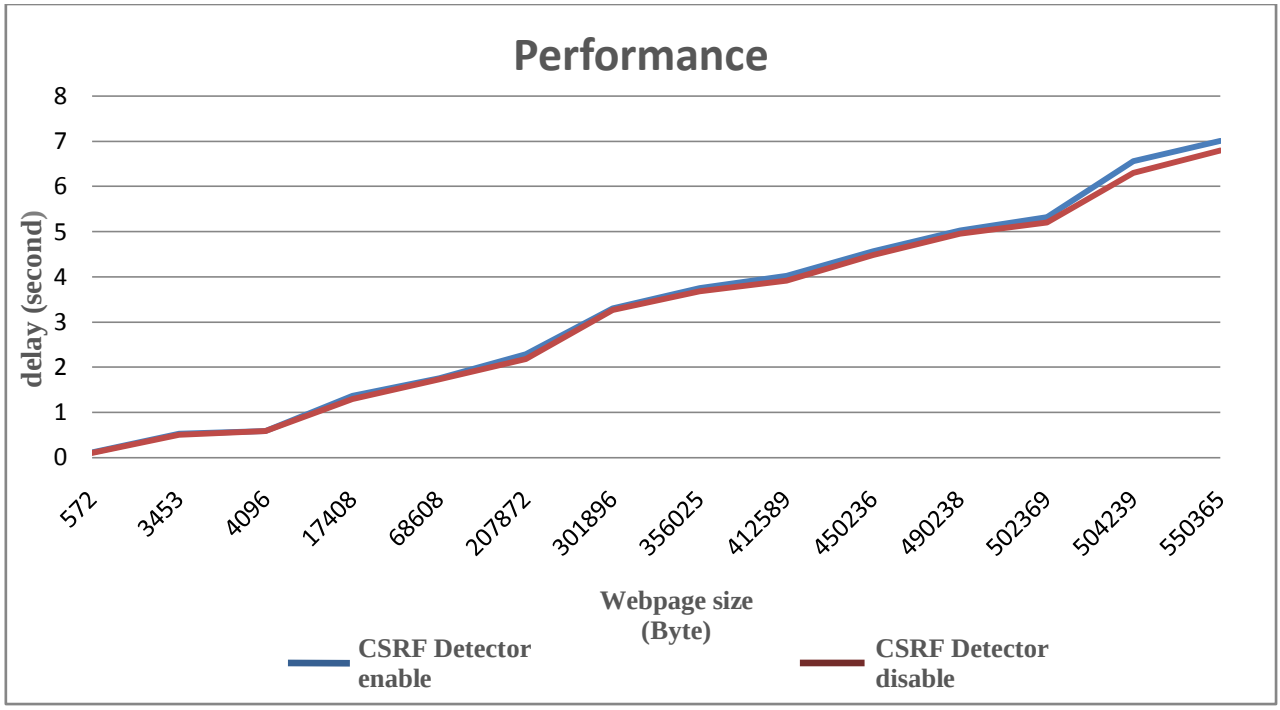

Figure 4.13: Performance line chart.

The line chart shown in the figure 4.15 represents the performance of our tool. The blue line shows loading delay time when CSRF Detector is disabled and the red one when CSRF detector is enabled. We can say that the rise of the average loading time is simultaneously, and the highest difference time between them is 0.21 seconds. We have also notice that in some cases when Webpage triggered a big number of cross-domains requests, the performance will be higher than normal cases, because of our tool will strip the no importance of additional information send with those requests in form of cookies.

Finally, the result of our performance testing shows that CSRF Detector effect on browser performance is negligible.

7. Conclusion

In this chapter, we presented the implementation and experimentation of our CSRF Detector extension. The extension was tested by simulating the CSRF attacks. The testing shows the efficiency of our extension and proves the high detection of advanced and basic CSRF attack.

GENERAL CONCLUSION

GENERAL CONCLUSION

Cross-site request forgery is a widely exploited vulnerability in web sites. CSRF attacks occurs when the attacker takes the advantages of implicit authentication mechanisms of HTTP protocol and cached credentials in the browser to execute a sensitive action on a target website behalf of an authenticated victim user without his knowledge. We have shown in this project the need for an autonomous client-side CSRF protection mechanism, especially since the already existing server-side protection mechanisms fail to get adopted against advanced CSRF attacks and the available client-side solutions do not suffice. We have provided a solution to this requirement with our client-side browser extension. We developed a Google Chrome extension "CSRF Detector" to defeat the attacker attempt to perform CSRF attacks by **analyzing web requests** and **web pages content** to detect all the basic and advanced CSRF attacks.

Our work makes the following contributions:

- Our proposed tool 'CSRF Detector' detect CSRF attacks with a hybrid mechanism based on the notion of content checking and request analysis.
- CSRF detector can detect reflected XSS attack in web site for the purpose of preventing attackers from using such vulnerability to carried out the CSRF attacks.
- CSRF Detector also has the ability to detect cross and sub-domain CSRF attacks.
- CSRF Detector adopts **Google Safe Browsing** service to identify unsafe websites and notify users to protect themselves from harm websites (malware, phishing …).
- CSRF Detector also serves as a tool for web developers to detect CSRF vulnerability. It allows developers to quickly discover forms that are likely unprotected against CSRF attacks.

We completed this project by conducting some tests to evaluate the efficiency and the capability of the CSRF Detector against CSRF attacks. To make sure that its results match with what is predetermined, we developed a small web application and some malicious web pages in order to simulate the basic and advanced CSRF attacks. We also tested our solution in Real-life with vulnerable and white list websites. The obtained experimental results demonstrate that our CSRF detector extension successfully detects all the generated attacks with no false positive, this proves the effectiveness of our extension to detect all basic and advanced CSRF attacks and protect users from harm without effect on browser performance.

Perspectives:

The proposed approach in this project introduces a novel client-side protection mechanism for CSRF attacks but it still not yet perfect, and our discussed techniques need more improvement so that can provide full and complete protection against CSRF attacks. Other authentication mechanisms such as SSL authentication needs to be further investigated to prevent CSRF attack abusing such credentials. Besides to this limits, CSRF Detector extension is designed only to work with Google Chrome browser; we will take this in consideration and adopt it in other browsers for future work. Finally, as more capabilities are added to browser clients, and as more sites involve sophisticated programming and client-server interactive services, CSRF attack will become more prevalent, and because of the complexity of Web technologies continue to increase, we can expect further new categories of CSRF attacks to emerge. Before the server-side defenses are updated to prevent such attacks, CSRF Detector can be a transitional solution defense with the advantage of a fast adoption to protect users and web applications from these attacks.

REFERENCES

References

Books :

[2] Leon, S and Rechard, R, Web Application Architecture: Principles, protocols and Practices,England, 2003.

[3] David Gourley, Brian Tottyb, Marjorie Sayer, Sailu Reddy, and Anshu Aggarwal, HTTP: The definitive guide , O'Reilly Media, United States of America, 2002.

[4] Stephen Thomas, HTTP essentials, Wiley Computer Publishing John Wiley & Sons, Inc. New York, 2001.

[5] Michael Cross, Devlopers guide to Web application security, United States of America, 2007.

[6] Leon Lepofsky, The manager guide to web application security, apress, United State of America, 2014.

[9] R. Fielding, U. Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners- Lee, Hypertext Transfer Protocol -- HTTP/1.1, The Internet Society, June 1999.

Theses:

[1] Theodoor Scholte, Securing Web Applications by Design, Doctorat, Telecom ParisTech French, 2012.

[7] Antoine Delignat-Lavaud, On the security of authentication protocols on the web, Cryptography and Security , Doctorat, PSL Research University, Frensh, 2016.

[8] Anil Saini, Attacks and Mitigation in Web Browsers Submitted , Doctorat, Department of Computer Science and Engineering, JAIPUR, 2016.

[10] Kristoffer Wanderydz, Web application security in the JAVA environment, Doctorat, Department of Information Technology - Security Karlskrona, Sweden, 2012.

[12] ALEXANDRE VERNOTTE, Pattern-Driven and Model-Based Vulnerability Testing for Web Applications, Doctorat, university of franche-comte, Frensh, 2015.

[17] Philippe De Ryck, Client-Side Web Security Mitigating Threats against Web Sessions, Doctorat, Faculty of Engineering Science Leuven, December 2014.

Websites:

[11] OWASP, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet Accessed: 15/03/2018.

[13] Mozilla. https://developer.mozilla.org/en/docs/DOM, Accessed: 16/03/2018.

[14] Microsoft Corporation, http://msdn.microsoft.com/en-us/library/ff648636.aspx Accessed : 18/03/2018.

[20] CWE/SANS, http://cwe.mitre.org/top25/index.html Accessed: 29/03/2018

[26] Mozilla, https://wiki.mozilla.org/Security/Origin, Accessed: 01/04/2018.

[31] Sitelock, https://blog.sitelock.com/2017/04/what-is-a-website-vulnerability, Accessed: 02/04/2018.

[32] Nvisium, https://nvisium.com/resources/blog/2017/11/30/owasp-top-10-2007-2017-the-fall-of-csrf.html , Accessed: 02/04/2018.

[36] OWASP, https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), Accessed: 05/04/2018.

[37] Requestpolicy http://www.requestpolicy.com, Accessed: 05/04/2018.

[38] Ruby on Rails, http://api.rubyonrails.org/classes/ActionController/RequestForgeryProtection.html, Accessed: 10/04/2018.

[39] Statista, https://www.statista.com/statistics/268299/most-popular-internet-browsers/, Accessed: 20/04/2018.

[40] Chrome, https://developer.chrome.com/extensions , Accessed: 20/04/2018.

[41] Chrome, https://developer.chrome.com/extensions/webRequest Accessed: 21/04/2018.

[43] OWASP, https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001) , Accessed: 21/04/2018.

[44] Chrome, https://developers.google.com/safe-browsing/v4/, Accessed: 22/04/2018.

[45] Alexa, www.alexa.com/topsites, Accessed: 01/05/2018.

[46] Portswigger, https://portswigger.net/burp/help/suite_functions_csrfpoc, Accessed: 01/05/2018.

[47] CSRF-Discover, https://chrome.google.com/webstore/detail/csrf-discoverer/acpmnjbbeadfhlikfibcjbkmfbopadek , Accessed: 13/05/2018.

[49] Scriptsafe, https://chrome.google.com/webstore/detail/scriptsafe/oiigbmnaadbkfbmpbfijlflahbdbdgdf, Accessed: 17/05/2018.

[50] ScriptBlock, https://chrome.google.com/webstore/detail/script-blocker-for-chrome/aakchmnmjneadkakfihibdbepehaflop, Accessed: 20/05/2018.

Articles:

[15] Shahriar, Hossain and Mohammad Zulkernine, Client-side detection of cross-site request forgery attacks, In Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, IEEE, 2010, pp. 358-367.

[16] Martin Johns and Justus Winter, RequestRodeo: Client Side Protection against Session Riding, The OWASP Europ conference, Leuven, Belgium, May 2006.

[18] Sentamilselvan K, Survey on Cross Site Request Forgery (An Overview of CSRF), Conference Paper, Kongu Engineering College, March 2013.

[19] A.Barth, C.Jackson, and J.C.Mitchell. “Robust defenses for cross site request forgery”. In Proc. ACM Conference on Computer and Communications Security (CCS), Oct, 2008.

[21] Mohd. Shadab Siddiqui and Deepanker Verma. \Cross Site Request Forgery: A common web application weakness". In: Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on. IEEE, 2011.

[22] Sapna Satish, CSRF- The hidden menace, Conference Paper, The OWASP Europ conference, June,2008.

[23] Ramarao R, Radhesh M, Alwyn R Pais,Preventing Image based Cross Site Request Forgery Attacks, Department of Computer Engineering, National Institute of Technology, Karnataka, INDIA

[24] Xing, Xinyu, Elhadi Shakshuki, Darcy Benoit, and Tarek Sheltami. "Security analysis and authentication improvement for ieee 802.11 i specification." InGlobal Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, 2008, IEEE, pp. 15.

[25] Zeller, William, and Edward W. Felten. "Cross-site request forgeries: Exploitation and prevention, The New York Times, 2008, p: 1-13.

[26] Telikicherla, Krishna Chaitanya, Venkatesh Choppella, and Bruhadeshwar Bezawada, CORP: A Browser Policy to Mitigate Web Infiltration Attacks, InInformation Systems Security, Springer International Publishing, 2014, pp. 277-297.

[27] De Ryck, Philippe, Lieven Desmet, Wouter Joosen, and Frank Piessens. "Automatic and precise client-side protection against CSRF attacks." InComputer Security–ESORICS 2011, Springer Berlin Heidelberg, 2011, pp. 100-116.

[28] Sentamilselvan. K , Lakshmana Pandian. S, Ramkumar. N, Cross Site Request Forgery: Preventive Measures, International Journal of Computer Applications (0975 – 8887) Volume 106 – No.11, November 2014.

[29] Wim Maes, Thomas Heyman, Lieven Desmet, Wouter Joosen, Browser Protection against Cross-Site Request Forgery, Katholieke University Leuven, Belgium, 2009.

[30] OWASP foundation, The ten most critical web application security risks 2017 , The Creative commons attribution share-alike license, The United State of America, 2017.

[33] Avinash Sudhodanan, Roberto Carbone, Luca Compagna, Nicolas Dolgin, Alessandro Armando and Umberto Morelli, Large-scale Analysis & Detection of Authentication Cross-Site Request Forgeries, Conference paper, IEEE European Symposium on Security and Privacy, 2017.

[34] Abdalla AlAmeen ,Building a Robust Client-Side Protection Against Cross Site Request Forgery, International Journal of Advanced Computer Science and Applications, Vol. 6, No. 6, 2015.

[35] A. Sudhodanan, A. Armando, R. Carbone, and L. Compagna. Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications. In 23nd Annual Network and Distributed System Security Symposium, NDSS, San Diego, CA, USA, February 21-24, 2016.

[42] Sebastian Lekies, Walter Tighzert, and Martin Johns,Towards stateless, client-side driven Cross-Site RequestForgery protection for Web applications, Sicherheit, 2012.

الملخص

يعد CSRF واحد من أخطر الهجمات السيبيرانية و من بين أسوأ عشر التغرات الأمنية في تطبيقات الويب. يحدث هجوم CSRF عندما يستغل المهاجم مزايا اثبات الهوية الضمنية لبروتوكول HTTP وبيانات الدخول المخفية مؤقتا في المتصفح لتنفيذ إجراء حساس على الموقع المستهدف نيابة عن المستخدم الضحية بدون علمه. في هذا المشروع ، نقدم آلية حماية خفيفة يمكن إضافتها إلى متصفح Chrome Google كتطبيق امتداد للمتصفح. الامتداد CSRF Detector منجز كليا بجانب العميل لمنع المهاجم من محاولة القيام بهجمات CSRF وذلك من خلال تحليل طلبات صفحات الويب وفحص محتوى صفحات الويب المستقبلة قبل عرضها على المستعمل للكشف عن جميع هجمات CSRF الأساسية و المتقدمة.
تظهر نتيجة التقييم الخاصة بأن امتداد 'CSRF Detector ' يكتشف بنجاح جميع الهجمات الناتجة وتظهر قدرته على حماية المستخدمين وتطبيقات الويب ضد هجمات CSRF.

الكلمات المفتاحية : هجمات CSRF ,كاشف CSRF, iframe, XSS .

Abstract

CSRF is one of the most serious cyber-attacks and has been recognized among the major threats and among the top ten worst vulnerabilities to web applications. CSRF attack occurs when the attacker takes the advantages of implicit authentication mechanisms of HTTP protocol and cached credentials in the browser to execute a sensitive action on a target website behalf of an authenticated victim user without his knowledge. In this project, we present a lightweight protection mechanism that can be added to Google Chrome browser as an extension. Our tool “CSRF Detector” is purely implemented on the client-side to defeat the attacker attempt to perform CSRF attacks by **analyzing web requests** and **web pages content** to detect all the basic and advanced CSRF attacks. Our evaluation result shows that CSRF Detector extension successfully detects all the generated attacks and it has the ability to protect users and web applications against CSRF attacks with no false positive.

Key words: iframe, XSS, CSRF Detector, CSRF attacks.

Résumé

CSRF est l'une des cyber-attaques les plus sérieuses, elle est reconnue parmi les principales menaces et parmi les dix principales vulnérabilités des applications Web. Une attaque CSRF se produit lorsque l'attaquant prend les avantages des mécanismes d'authentification implicites du protocole HTTP et des informations d'identification mises en cache dans le navigateur pour exécuter une action sensible sur le site Web cible avec le compte d'un utilisateur victime authentifié à son insu. Dans ce projet, nous présentons un mécanisme de protection léger qui peut être ajouté au navigateur Google Chrome en tant qu'extension. Notre outil "CSRF Detector" est implémenté côté client pour empêcher l'attaquant d'effectuer des attaques CSRF en analysant les requêtes web et le contenu des pages web reçues pour détecter toutes les attaques CSRF basiques et avancées. Notre résultat d'évaluation montre que l'extension CSRF Detector détecte avec succès toutes les attaques générées et montre sa capacité à protéger les utilisateurs et les applications Web contre les attaques CSRF sans faux positif.

Mots clés : iframe, XSS, CSRF détecteur, CSRF attaques.