CSEC 793 Capstone in Computing Security
Project Report

EVALUATE THE EFFECTIVENESS OF MODERN PROTECTION TECHNIQUES AGAINST CROSS-SITE SCRIPTING

April 26, 2019

Sumayyah Alahmadi
Department of Computing Security
College of Computing and Information Sciences
Rochester Institute of Technology
sfa8135@rit.edu

1 Abstract

Cross-site scripting (XSS) attack is one on the most and well-studied problems in web security field. This vulnerability allow attackers to inject malicious client- side script into website's page that is being viewed by other users. The attacker can run JavaScript in the web application which lead to session hijacking or malicious redirection to steal sensitive information. There are many tools and techniques that have been suggested to defense against this attack; such as, Content Security Policy (CSP), Web Application Firewall (WAF), and sanitize HTML. However, many of these existing tools may or may not work on different browsers. This project will shed the light on several modern protection solution against XSS and their effectiveness on most used browsers (Chrome, Safari, Firefox, Opera, Internet Explorer, Microsoft Edge). It is important to test the capabilities and effectiveness of these tools and see whether they are sufficient to protect against XSS or not. Beginning by choosing and identifying several recent vulnerable web application and verify which browsers the attack can work on. Then, different modern protection solutions will be implemented on top used browsers to test which browser the defense tool will work with. Comparison of effectiveness will be conducted between different tools on different browsers.

2 Introduction

Web application vulnerabilities were existed since the beginning of the world wide web. However, it got more attention considering the fast expansion of online presence of governmental and critical businesses. The cause of web application vulnerabilities can be due to misconfiguration the web server or failing to sanitize and validate users input. These system flaws or weaknesses allow the attacker to compromise the application's security. One of the most common vulnerabilities is the cross-site scripting (XSS) where the attacker injects malicious input that can be viewed by other users. This is considered a client-side script injection and it is a major threat factor for web application as it is responsible for most of web application exploitation.

There have been tremendous open source software by developer that contribute to mitigate web vulnerabilities and increase security. One of these open source project that produce articles, techniques and tools in the web application security field is the open web application security project OWASP. OWASP has detailed documentation for testing and preventing against XSS attack using proper encoding and escaping techniques. The major three documentations against XSS are secure headers project, content security policy header and mod-security core rule set project. These techniques have been proved to be effective in defense against XSS. Moreover, OWASP recommend seven rules to prevent cross-site scripting; such as, HTML escape, attribute escape and JavaScript escape to prevent against inserting potential malicious data.

There are three types of cross-site scripting: persistent, reflected and Document Object Model (DOM)-based XSS. In persistent or stored XSS attack, the server will save the data that have been provided by the attacker and permanently return it to every other user. Thus, it has more damage than the other two XSS types. It happens in two requests: one when the attacker inject the server by the malicious input, and the second when the victim loads the page that have the malicious part. The other type of cross-site script attack is the reflected or non-persistent where the attacker can create a custom URL that when visited by the victim, it executes malicious script in the victim's browser. The goal of reflected XSS is to steal sensitive and confidential information such as, cookies and account information which leads to impersonate the victim and perform various task on behalf of the legitimate user. This type considered more common than the other two because in order for the attack to be successful, all it needs is clicking on the link by the victim; for example, sending the link inside a phishing email. Lastly, document object module based XSS means that the attack appears on the DOM instead of being part of the HTML script. In other words, investigating the DOM of the page on the run-time is the only way to observe the payload or the malicious script.

In this project, the main focus will be on persistent or stored cross-site script vulnerability. Also, regarding protection solutions, the focus will be on the recommended solutions by OWASP: using security headers and web application firewall. The environment of the testing will be on three operating systems: Linux, Mac OS and windows, and on six top used browsers: Chrome, Firefox, Opera, Safari, Microsoft Edge and Internet explorer.

When considering securing a website, usually hardening the website by using HTTP response security headers are one of the most recommended options to make sure that best practices are being followed. Although HTTP security headers are easy to implement and configure, it provides a strong defense mechanism against large number of attacks due to the fact that they add a security layer against various vulnerabilities. Two of security headers that prevent XSS attacks are Content-Security-Policy header and X-XSS-Protection header. CSP header, which standardized by W3C, white-list allowed source and disable some JavaScript feature to mitigate various types of injection attacks against HTTP and HTTPS.

Web application firewall is one of the most promising countermeasures against cross-site scripting attack because it adds an extra layer of security by monitoring and filtering HTTP traffic between the end user and the web server. While a regular firewall provides security between servers, WAF is capable to monitor, filter and block the content from a specific web application. Web application firewall is designed to provide protection for the application layer in the OSI model against attacks that happen on the application level like XSS, cross-site-forgery, and SQL injection. One of the notorious web application firewalls is the mod-security which is an open source WAF to monitor, log and control access for HTTP traffic. Mod security was tested in the project on Apache server only, but it can work with other servers such as: Nginx and IIS web server. Mod-security is a firewall engine that needs a specific policy to inspect the traffic and see if there is any regular expression match with known attack. In this project, we used OWASP mod-security Core Rule Set (CRS), which is a generic rule set that protects web servers from common attacks including the OWASP top ten attacks with little to no false positive alert. CRS not only block the attack, but also it can keep track of attack attempts which will help the developers focus on web applications that are under attack. The OWASP mod-security core rule set is made up of generic signatures for the most popular attacks. These signatures are stored in .conf files and use various parser and regular expression checks to detect different kinds of attacks.

The ultimate goal for the project is to evaluate the capability of the modern protecting techniques against XSS on the most popular browsers to validate and ensure their effectiveness to protect the users from cross-site script attack. As a result, the project will give the developers and security researchers more insight into the capability of these solutions across different browsers. Finally, analyzing the results by compare and contrast between these methods will be provided along with recommendations for a future work on similar projects.

Regarding the structure of this paper, we present a brief literature review in the first section that is studying the same or similar problems. Next, we proceed to describe the methodology that we followed to implement the project. We then walk through the implementation in detail by showing the outcome to most cases that we implemented, and we conclude with the results to summarize our finding of the project and some ideas for a planned future work on similar projects.

3 Literature Review

Mahmoud et al [1], compares between 11 different XSS detection and prevention techniques from the last seven years in term of strength, weakness, exploitation location, attack type and XSS taxonomy. The paper also discusses several XSS incidents and threats. Some of the mentioned tools in the paper can be used to prevent cross-site scripting attacks; such as, SaferXSS and CSSXC.

In [2], Dolnak illustrated how simple and effective it is to implement and propagate Content-Security-Policy CSP header in improving secure communication over HTTP and HTTPS protocol. The CSP header basically gives permission to the web browser to load scripts only from approved domains and prevent loading malicious code from any third party domain that is not explicitly allowed. As CSP header offers an elegant solution to many web injection issues including XSS, it is one of the major tools that will be tested in this project on several browsers. According to Dolnak [2], statistics show that only 24 percent of websites use a good security mechanism which highlights the necessity of hardening communication over HTTPS protocol. Content-Security-Policy header is one of many HTTP security headers that intend to secure communication between the web server and web browser by allowing communication only between trusted domains.

Furthermore, one of preferred countermeasures to XSS these days is deploying web application firewall (WAF). Singh et al [3] discussed the effectiveness of different configuration levels in one of the top open-source WAFs, ModSecurity WAF. The paper intended to conduct experiment evaluation on balancing between security needs and security paranoia levels to analyze the detection capability for different configuration levels. The experiment analyzes several bypass vectors that focus on OWASP top 10 risk including XSS against three different levels in ModSecurity WAF. Three of these attack vectors are: reflected, stored and DOM-based XSS, and only reflected XSS were successful to bypass WAF on the first (default) paranoia level. Thus, our project will use ModSecurity WAF as one of the tested techniques against XSS.

As a significant number of web application were written in PHP, several tools have focused on detecting XSS in PHP source code. In [4], Marashdiah et al focused on detecting XSS by analyzing the web application source code. The analysis approach has been categorized to three sections. First, the static approach examines the source code to detect the infected path and it ties to a specific framework. The second tool is dynamic analysis, which examines the web application in a similar manner to a userâs behavior in the web browser. Third, hybrid analysis tool is used to combine advantages of both static and dynamic analysis which helps to lower the false positive rate in the results.

Encoding HTML special characters and validating user inputs are highly recommended to prevent XSS attack. In [5], Twana et al discusses different methodologies to prevent and mitigate cross-site scripting attacks including replacement, removal, escaping and restriction user input. Also, some guidance is provided; for example, accepting only valid data, rejecting recognized harmful data, and cleaning harmful data. Moreover, developers can prevent XSS attacks by using a regular expression to validate data, or by utilizing PHP built-in function such as, htmlEntities() and htmlSpecialChar().

A nested context-aware sanitization technique has been suggested by Gurpreet et al in [6] to defend against HTML5 cross-site scripting attack vectors. The proposed design was implemented and tested on five social network platform that are vulnerable to XSS attack. To alleviate and prevent the execution of malicious JavaScript code and links on HTML5 web application, the proposed design acts as a robust cross-site scripting sanitization that extracts untrusted JavaScript links and finds diverse nested contexts which leads to accurate results and acceptable false positive and false negative rate.

One of the client-side defense tools against XSS attack is XBuster [7], which is a Mozilla Firefox browser extension that splits HTTP request parameters into HTML and JavaScript context. Then, if the context exceed a specific threshold length, it will be tested to search for a match. Unlike most of XSS filters that may unintentionally facilitate clickjacking attack, XBuster was designed to defend against both attacks. This extension can be utilized in similar projects by attempting to implement it to defend against the five chosen vulnerabilities and observe and analyze the results.

Additionally, one of the novel methods to prevent cross-site-script attack which does not involve browser code modification is Moving Target Defense (MTD). MTD is based on the idea of constantly changing the mode of the system configuration as a way to hardening the system so that it would be difficult for the attacker to catch the current status of the system [8]. This randomness will add extra layer of defense, and it will differentiate between real and injected malicious JavaScript. This method is tested and proved to prevent XSS attack without negative impact on the system performance.

In [9] and [10], authors discuss the role of a content-security-policy header in mitigating cross-site scripting attack. Even though CSP header is a robust client side security layer, it required a significant change in the web application. Mhana et al in [10], suggested an approach that works as a plugin in the browser to save the effort of modifying the code and potentially misconfiguring the content-security-policy header, which may lead to security issues. One of the benefits of this plugin is that it can be installed without interfering with the web source code. On the other hand, Imran [9] performed an experiment to evaluate the effectiveness of report-only with the report-url directive CSP header. The results of the experiment in Firefox, Chrome, Opera and Safari were that the content-security-policy header successfully blocked a number of XSS attacks.

Lavrenovs et al. analyzed the HTTP response security headers for the most popular website according to Alexa's top one million list [11]. The experiment included four different HTTP requests: two HTTP/1.1 requests and two HTTPS requests, one to the domain itself and the other to its www subdomain. The paper focuses on investigating the following HTTP response security headers: Strict-Transport-Security, Content-Security-Policy, X-XSS-Protection, X-Frame-Options, Set-Cookie and X-Content-Type. Moreover, the authors discuss three other headers that may reveal sensitive information about the web server that might be used by the attacker to perform malicious action like: server, date, and

Referrer-Policy. Based on the collected data, the finding shows that there are no big differences in HTTP/1.1 and HTTP/2 response headers. Also, websites that use HTTPS have more security policies than HTTP based on response header, but they usually have some TLS misconfiguration in the HTTPS request.

In [12], Weichselbaum et al studied on a large-scale the practical benefits and flows of implementing content-security-policy headers across one million hosts. The authors discuss three classes for bypassing CSP and how that can damage the security policy. By analyzing the threat model to bypass the CSP header and perform cross-site scripting attack, the authors found that security policies that supposed to control the script execution are ineffective or offer no value to prevent against XSS attack. The security model analysis of content-security-policy headers shows that attackers can bypass 94.72 percent of all security policies, which means that a whitelist mechanism does not provide robust protection against cross-site scripting attack. Moreover, the analysis shows that there is at least one insecure host in the whitelist hosts specified by CSP header security policy. Thus, instead of depending on the domains whitelist policy, cryptographic nonces-based policy and âstrict-dynamicâ keyword were proposed in the paper. This new way of writing policy will enhance the security offered by content-security-policy header because nonces-based CSP header enables legitimate individual script based on the dynamic nonce.

There are many differences between the present paper and some previous work in [13]. While our paper focuses on evaluating the effectiveness by testing CSP header and two other mechanism from the browsers perspective, Calzavara et al. evaluate the effectiveness of content-security policy header by analyzing four key factors that affect CSP header which are: browser support, website adoption, correct configuration and constant maintenance [13]. Except of the browsers support factor, the other three aspects show some limitation as the paper discussed. Even though the authors point out some limitation in CSP headers like: misconfiguration errors, weakness and lack of updates, they argue that many of these issues are fixable with additional research and better monitoring facilities.

In 'A Measurement Study of the Content Security Policy on a Real-World Application', Patil et al. implemented a FireFox extension tool, UserCSP, which provides the user with a custom policy to enhance the security of the content-security-policy header and to ease the CSP header adaption [14]. UserCSP tool will infer the policy automatically, which will give users the chance to be in charge of creating and modifying these policies.

Compared with other related work, this paper focuses more on testing the capability of modern solution techniques on different browsers and evaluates the effectiveness for these mechanisms and which browsers the techniques can work best with. This project will give developers and security researchers more insight into which techniques to use to hardening the web application and increase security by mitigating client-side injection attacks such as cross-site scripting attack.

4 Methodology

In order for the project to be completed in organized steps, five phases have been followed with each vulnerable web application, browser, and solution technique to achieve the required results. The five phases are: Identify vulnerability, deploy vulnerable web application, test the vulnerable parameter, implement modern defensing solution against cross-site script attack, and analyze the result.

- Identify vulnerability

  In this phase, five vulnerable web applications have been chosen from exploit database website which is a website that contain public exploits and corresponding vulnerable software that is developed by penetration tester and vulnerability researcher. Here are the five vulnerable web applications that have been tested on the project:

  1) WordPress Plugin Quizlord 2.0

  2) Joomla Core 3.9.1

  3) Gila CMS 1.9.1

  4) Rukovoditel ERP CRM 2.4.1

  5) InoERP 0.6.1

  Three of these vulnerabilities is a content management system (CMS) which is a web-based system to create and edit digital content, and the other two vulnerabilities are for Enterprise resource planner which is a real-time management for core business process. WordPress and Joomla considered to be the most popular CMS systems and both vulnerabilities have been discovered in 2018 and 2019 respectively. Gila CMS vulnerability is an interesting one because it does not work on Opera and Chrome even before implementing security mechanism as it turns out that both browsers have a build-in XSS detection that make some XSS attack unsuccessful. The other two vulnerabilities were found in ERP web application, and they have been tested on all browsers.

- Deploy vulnerable web application

  Each vulnerable web application have been deployed on three different environment: Linux, Mac OS and Windows to cover all six browsers. Safari was tested on Mac OS. Firefox, Chrome and Opera were tested on Linux ubuntu 18.04. Internet Explorer and Microsoft Edge were tested on Windows 10. Testing begin by deploying vulnerable web application on virtual machine to test and make sure that the attacks work then testing security tools for each browser and each vulnerable web application.

- Test the vulnerable parameter

  Among other 100 vulnerable web applications, only those that actually not patched and still vulnerable to XSS has been chosen. Testing the vulnerable parameter by performing persistent XSS attack where the following script is being injected into the web server:

poc"><script>alert(1)</script>then if the web server is vulnerable and the attack is successful, an alert will pop-up with number 1 indicating that this user input was not sanitize or validate and the JavaScript input treated as trusted script regardless. In this example, number one choose for simplicity and proof of concept, but in real scenario, attacker can steal sensitive information, redirect the victim to malicious website or execute browser exploit. Script tag choose to be used in proof of concept but XSS can be performed using other tags as well. This script will be stored on the server and reappeared in other user's browsers.

- Implement modern solution against XSS

  There are many tools and techniques to mitigate and protect the application from XSS, but among all of them, there are main three protection techniques that have been used and focused on in this project based on OWASP recommendations: Web application firewall mod-security, Content-Security policy header and X-XSS-Protection header. Through the report each one of these techniques will be discussed in details.

  First, unlike regular firewall that is used as safety gateway server, WAF monitor and filter the content of HTTP traffic and block the malicious packets. To protect the client host, a proxy is recommended to be used, but to protect a server, we use web application firewall which is why it called reverse proxy. There are many different forms for WAF: an application, filter or server plugin. In this project, a WAF used as Plugin for Apache server. A choice of open-source web application firewall has been used, mod-security WAF which originally designed as a module for Apache HTTP server. It is responsible for monitoring, logging and access control. Moreover, along with implementing mod-security, OWASP mod security Core Rule Set CRS is enabled for generic attack detection rules that are definitely help with XSS protection. Second, if the browser detect reflect XSS, the X-XSS-Protection header will stop loading the page. This header is useful for older browsers version that do not support content security policy header, and it is not supported by Firefox. There are 4 modes for this header, and the mode: mode=block which filter XSS and sanitize the page without reporting the incident. Third, a response header that help guard XSS is content security policy CSP header. It's allowed multiple security policy header, and it is not supported by Internet Explorer older version. It has the following syntax: Content-Security-Policy : policy- directive() , and it has many type of directives such as: fetching, document, navigating, reporting and other.

- Analyze the results

  The final step is to gather and summarize information to compare and contrast between all the browsers, XSS vulnerabilities and protection solution against XSS attack.

5 Project Implementation

Here are the project implementation steps in detailed for each vulnerable web application: WordPress Plugin Quizlord 2.0, Joomla Core 3.9.1, Gila CMS 1.9.1, Rukovoditel ERP CRM 2.4.1, and InoERP 0.6.1. These vulnerable web application were found in exploit-db.com which is a database for public exploits and vulnerable software. After attempting about 100 vulnerable web applications, these are the one that still successfully work with no patch or problem in configuration. We make our choices of web browsers based on prevalence of the browsers usage between users, and these are the six browsers that will be tested on the project: Chrome, Firefox, Opera, Safari, Microsoft Edge and Internet explorer. Furthermore, the project focus is on the three main countermeasures to cross-site script vulnerability which are: Web application firewall, Content-Security-policy header and X-XSS-Protection header. Even though there are other mechanism to defense against XSS vulnerability, we choose these three techniques as they are recommended as the best techniques and also they were suggested by OWASP prevention documentation. For each vulnerable web application, we discuss in details the vulnerable parameters that are not sanitized or validated which cause the cross-site-script vulnerability.

Starting with WordPress Quiz lord plugin which has the title parameter not sanitize. This vulnerability tested and successfully worked on all six browsers. Mod-security WAF and CSP headers successfully protected the user from XSS while X-XSS-Protection header was not able to defense against XSS in all browsers. Here are two figures [1] and [2] that shows testing Quizlord plugin in Firefox in ubuntu 18.04 and Safari in Windows 10.

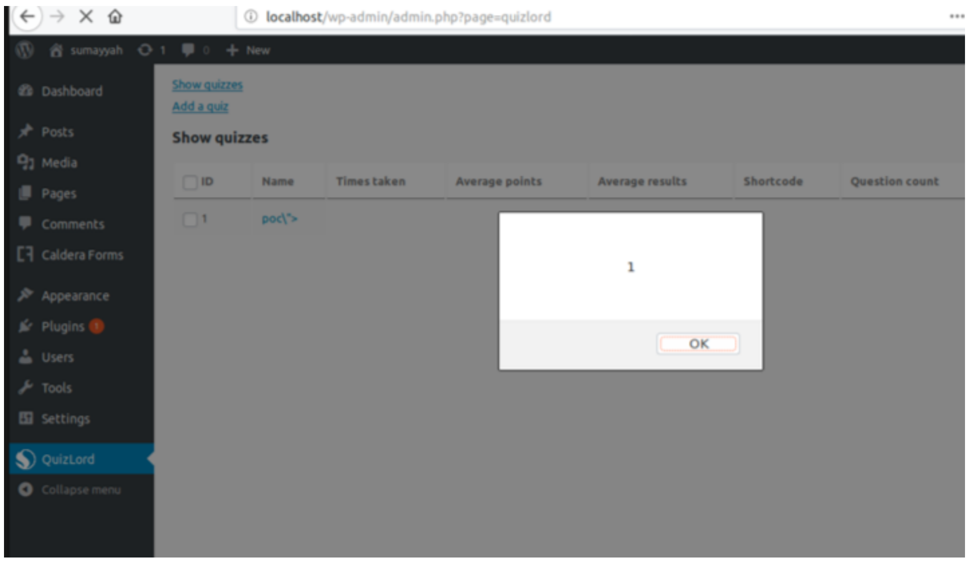
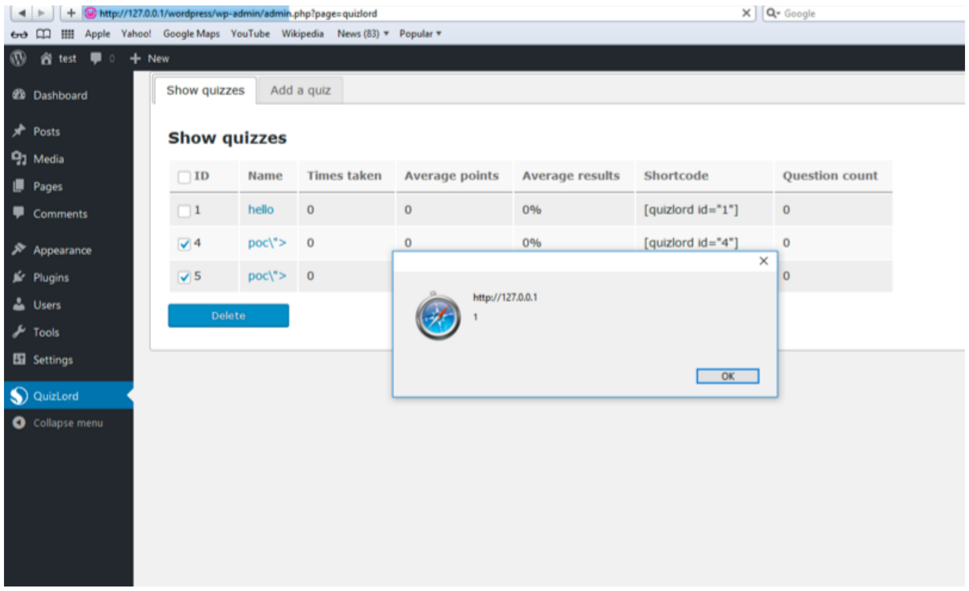


Figure 1: Testing Quizlord plugin on Firefox.



Show quizzes

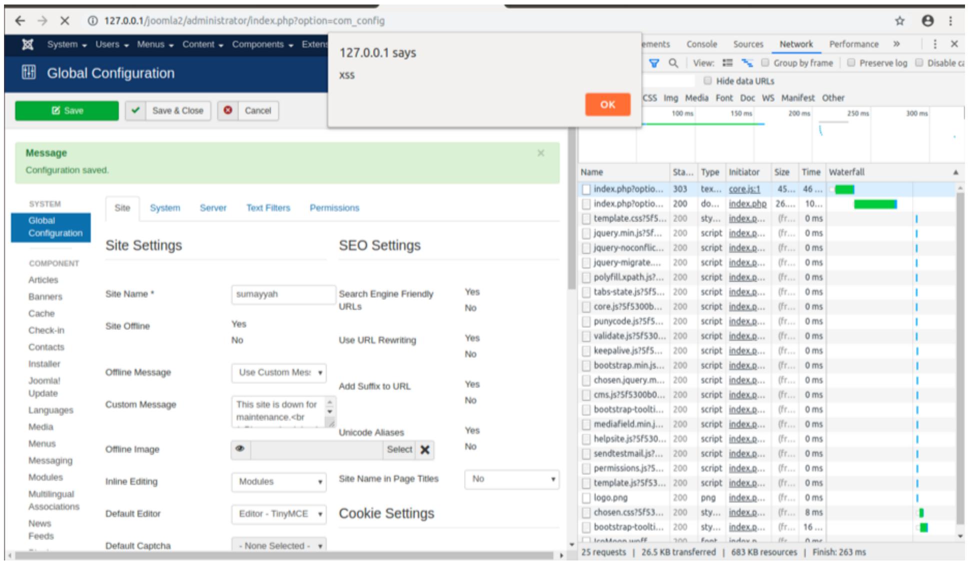
ID	Name	Times taken	Average points	Average results	Shortcode	Question count
1	hello	0	0	0%	[quizlord id="1"]	0
4	poc\">	0	0	0%	[quizlord id="4"]	0
5	poc\">	0				0

http://127.0.0.1
1

Figure 2: Testing Quizlord plugin on Safari.

Here are the project implementation steps in detailed for each vulnerable web application: WordPress Plugin Quizlord 2.0, Joomla Core 3.9.1, Gila CMS 1.9.1, Rukovoditel ERP CRM 2.4.1, and InoERP 0.6.1. These vulnerable web application were found in exploit-db.com which is a database for public exploits and vulnerable software. After attempting about 100 vulnerable web applications, these are the one that still successfully work with no patch or problem in configuration. We make our choices of web browsers based on prevalence of the browsers usage between users, and these are the six browsers that will be tested on the project: Chrome, Firefox, Opera, Safari, Microsoft Edge and Internet explorer. Furthermore, the project focus is on the three main countermeasures to cross-site script vulnerability which are: Web application firewall, Content-Security-policy header and X-XSS-Protection header. Even though there are other mechanism to defense against XSS vulnerability, we choose these three techniques as they are recommended as the best techniques and also they were suggested by OWASP prevention documentation. For each vulnerable web application, we discuss in details the vulnerable parameters that are not sanitized or validated which cause the cross-site-script vulnerability.

Second vulnerability is Joomla CMS global configuration in the vulnerable parameter Text Filters, and it worked on all browsers. All security defense that is mentioned in this report was tested against this vulnerability and show success except X-XSS-protection header.



127.0.0.1/joomla2/administrator/index.php?option=com_config

127.0.0.1 says

xss

Message

Configuration saved.

Site Settings

Site Name *	sumayyah
Site Offline	Yes / No
Offline Message	Use Custom Mes:
Custom Message	This site is down for maintenance.<br
Offline Image	Select
Inline Editing	Modules
Default Editor	Editor - TinyMCE
Default Captcha	- None Selected -

SEO Settings

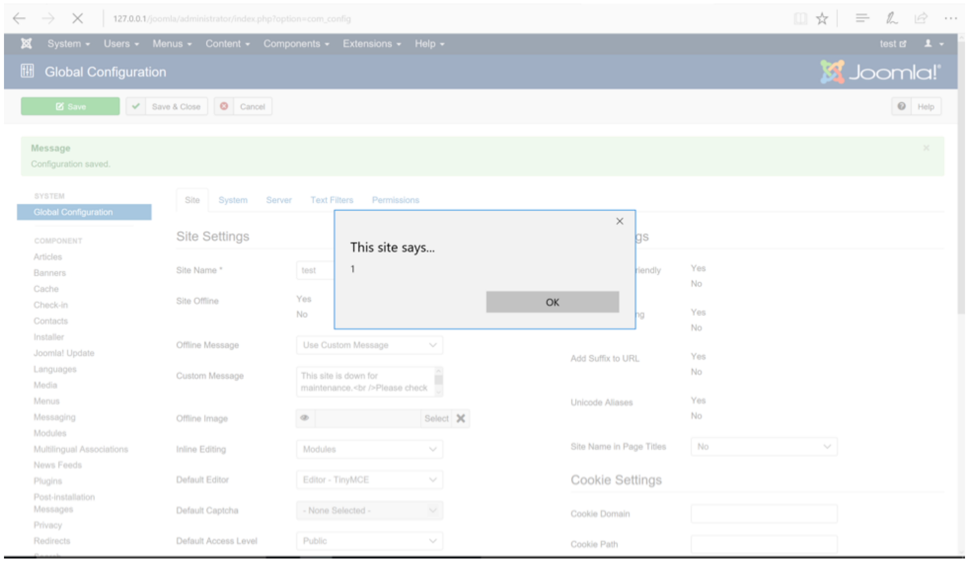
Search Engine Friendly URLs	Yes / No
Use URL Rewriting	Yes / No
Add Suffix to URL	Yes / No
Unicode Aliases	Yes / No
Site Name in Page Titles	No

Cookie Settings

Name	Sta...	Type	Initiator	Size	Time
index.php?optio...	303	tex...	core.js:1	45...	46 ...
index.php?optio...	200	do...	index.php	26....	10...
template.css?5f5...	200	sty...	index.p...	(fr...	0 ms
jquery.min.js?5f...	200	script	index.p...	(fr...	0 ms
jquery-noconflic...	200	script	index.p...	(fr...	0 ms
jquery-migrate....	200	script	index.p...	(fr...	0 ms
polyfill.xpath.js?...	200	script	index.p...	(fr...	0 ms
tabs-state.js?5f5...	200	script	index.p...	(fr...	0 ms
core.js?5f5300b...	200	script	index.p...	(fr...	0 ms
punycode.js?5f5...	200	script	index.p...	(fr...	0 ms
validate.js?5f530...	200	script	index.p...	(fr...	0 ms
keepalive.js?5f5...	200	script	index.p...	(fr...	0 ms
bootstrap.min.js...	200	script	index.p...	(fr...	0 ms
chosen.jquery.m...	200	script	index.p...	(fr...	0 ms
cms.js?5f5300b0...	200	script	index.p...	(fr...	0 ms
bootstrap-toolti...	200	script	index.p...	(fr...	0 ms
mediafield.min.j...	200	script	index.p...	(fr...	0 ms
helpsite.js?5f530...	200	script	index.p...	(fr...	0 ms
sendtestmail.js?...	200	script	index.p...	(fr...	0 ms
permissions.js?5...	200	script	index.p...	(fr...	0 ms
template.js?5f53...	200	script	index.p...	(fr...	0 ms
logo.png	200	png	index.p...	(fr...	0 ms
chosen.css?5f53...	200	sty...	index.p...	(fr...	8 ms
bootstrap-toolti...	200	sty...	index.p...	(fr...	16 ...

25 requests | 26.5 KB transferred | 683 KB resources | Finish: 263 ms

Figure 3: Testing Joomla CMS configuration on Firefox.



127.0.0.1/joomla/administrator/index.php?option=com_config

This site says...

1

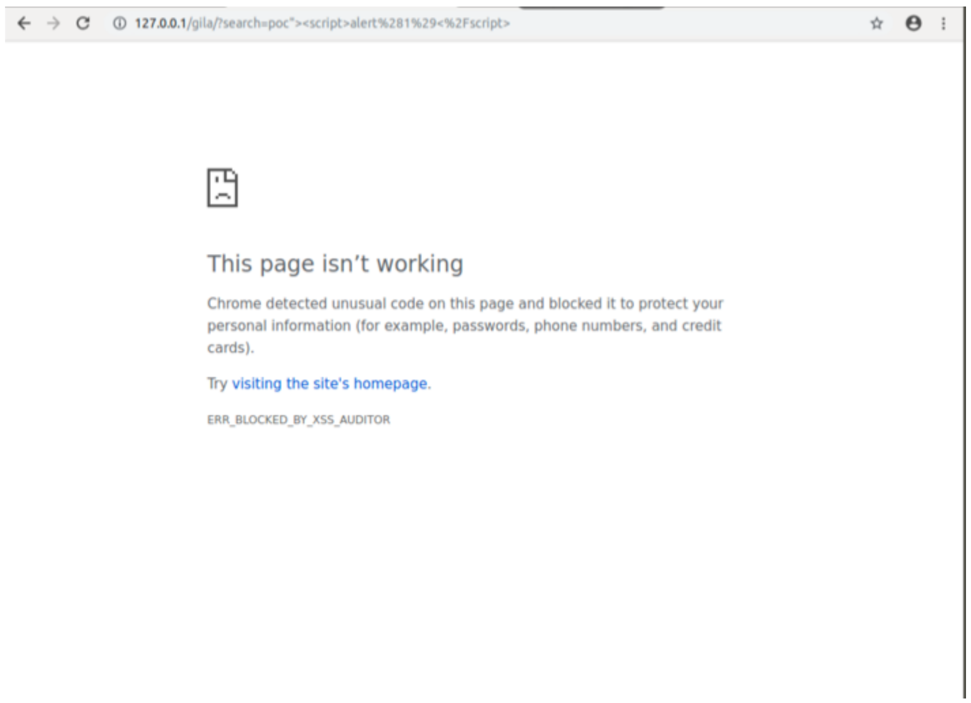
Message

Configuration saved.

Site Name *	test
Site Offline	Yes / No
Offline Message	Use Custom Message
Custom Message	This site is down for maintenance.<br />Please check
Offline Image	Select
Inline Editing	Modules
Default Editor	Editor - TinyMCE
Default Captcha	- None Selected -
Default Access Level	Public
Site Name in Page Titles	No

Figure 4: Testing Joomla CMS configuration on Safari.

Third, with Gila CMS,when tested with the follwing scripts: poc"><script>alert(1)</script>, we found different results than a previous two. Gila is a vulnerable web application with search parameter not sensitized and vulnerable to injection attacks. This vulnerable user input can be exploited successfully on the following browsers:Firefox, Safari, Microsoft Edge and Internet explorer, but not with Chrome and Opera. This is a positive result because it indicats that the browsers has a built-in functionality to prevent XSS. As it shown in the figures [5] and [6] below, it says ERR Blocked By XSS Auditor.



127.0.0.1/gila/?search=poc"><script>alert%281%29<%2Fscript>

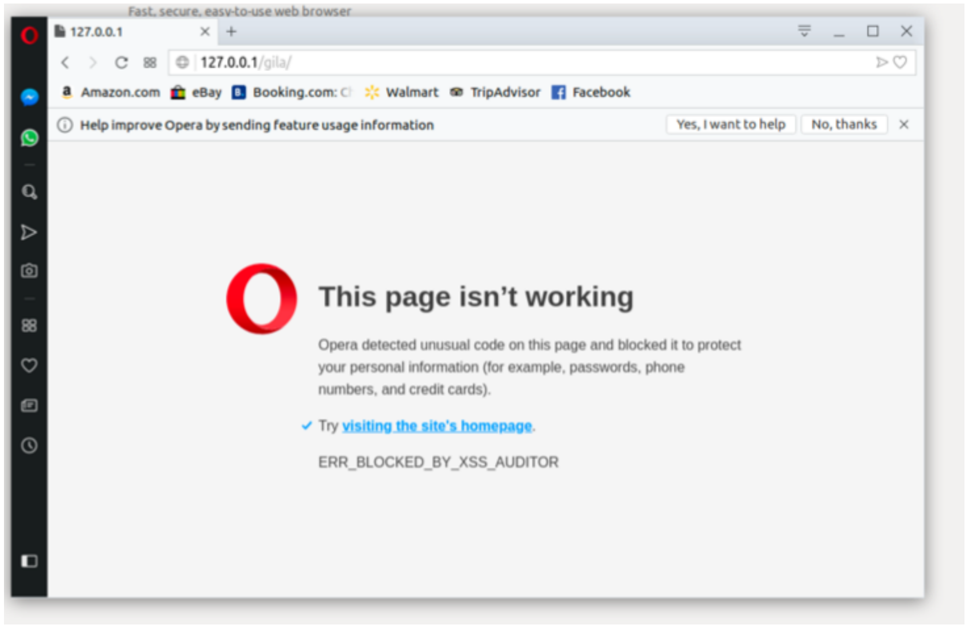
This page isn't working

Chrome detected unusual code on this page and blocked it to protect your personal information (for example, passwords, phone numbers, and credit cards).

Try visiting the site's homepage.

ERR_BLOCKED_BY_XSS_AUDITOR

Figure 5: Testing Gila CMS configuration on Chrome.


Fast, secure, easy-to-use web browser

127.0.0.1

127.0.0.1/gila/

This page isn't working

Opera detected unusual code on this page and blocked it to protect your personal information (for example, passwords, phone numbers, and credit cards).

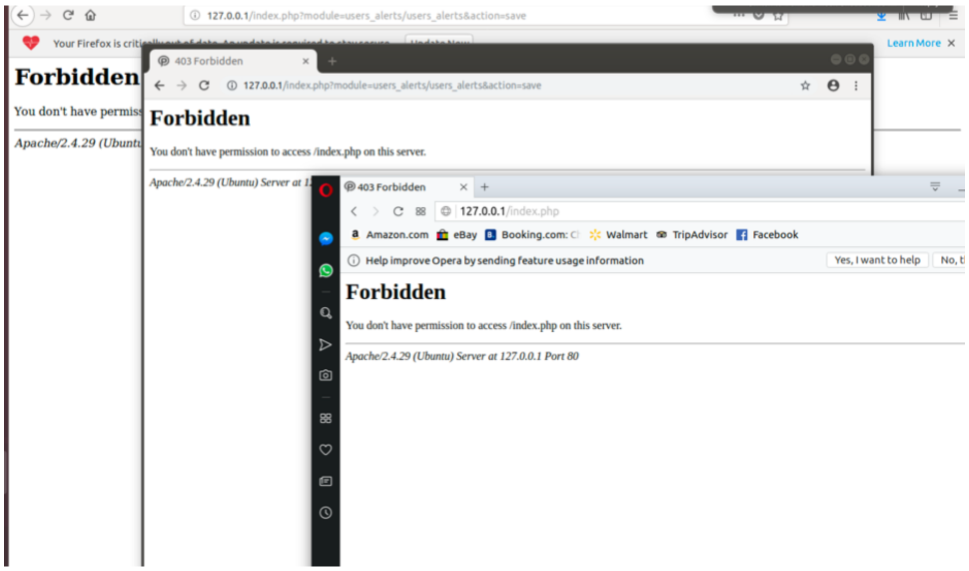
✓ Try visiting the site's homepage.

ERR_BLOCKED_BY_XSS_AUDITOR

Figure 6: Testing Gila CMS configuration on Opera.

Forth, the Rukovoditel web application is vulnerable because the user alert input is not validated. It was tested on the six browsers and 2 out of three solutions successfully defeated XSS attack. The figures [7] and [8] below show how mod-security WAF and CSP header response when detecting persistent XSS.

The last vulnerable web application that we tested is InoERP which has a title parameter in the asking question section not sanitizes and thus vulnerable to client injection attack. All browsers are victims to this attack, and the two figures [9] and [10] below show the testing environment in Opera browser and Internet Explorer browsers. Solutions have been implemented and only mod-security web application firewall and content-security-policy header worked.


127.0.0.1/index.php?module=users_alerts/users_alerts&action=save

Your Firefox is critically out of date. An update is required to stay secure.

Forbidden

You don't have permission to access /index.php on this server.

Apache/2.4.29 (Ubuntu) Server at 127.0.0.1 Port 80

403 Forbidden

127.0.0.1/index.php?module=users_alerts/users_alerts&action=save

Forbidden

You don't have permission to access /index.php on this server.

Apache/2.4.29 (Ubuntu) Server at 127.0.0.1 Port 80

403 Forbidden

127.0.0.1/index.php

Amazon.com eBay Booking.com Walmart TripAdvisor Facebook

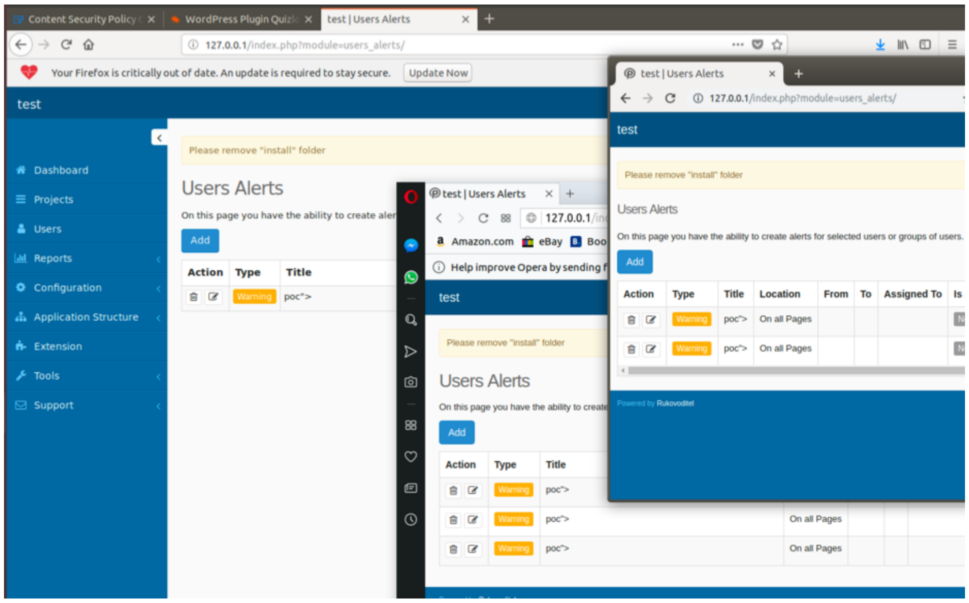
Help improve Opera by sending feature usage information — Yes, I want to help

Forbidden

You don't have permission to access /index.php on this server.

Apache/2.4.29 (Ubuntu) Server at 127.0.0.1 Port 80

Figure 7: implementing WAF against vulnerable RukovoditelERP on FireFox, Opera and Chrome.


Content Security Policy | WordPress Plugin Quiz | test | Users Alerts

127.0.0.1/index.php?module=users_alerts/

Your Firefox is critically out of date. An update is required to stay secure. Update Now

test

Dashboard · Projects · Users · Reports · Configuration · Application Structure · Extension · Tools · Support

Please remove "install" folder

Users Alerts

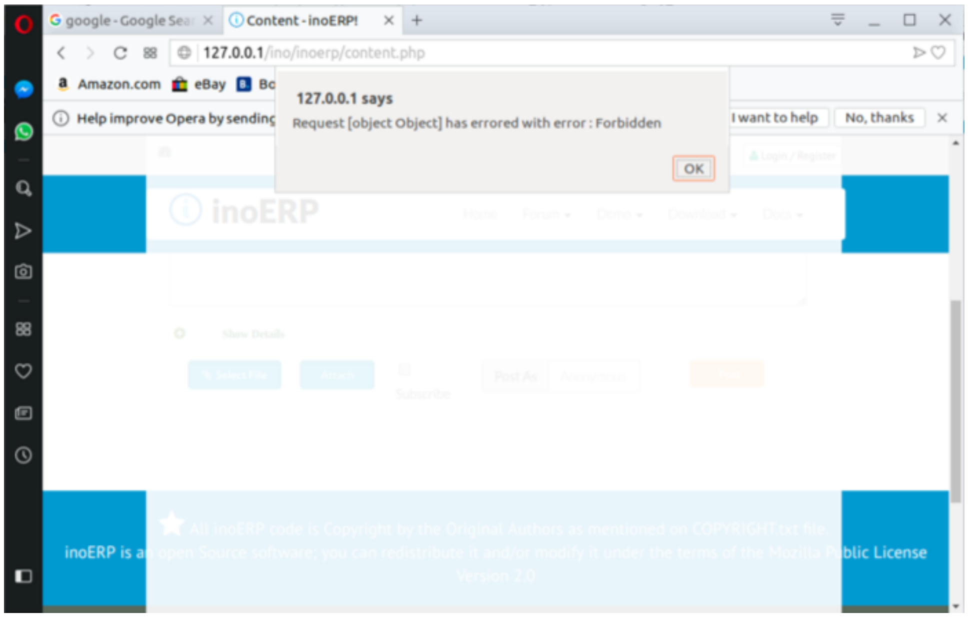
On this page you have the ability to create alerts for selected users or groups of users.

Add

Action	Type	Title	Location	From	To	Assigned To
	Warning	poc">	On all Pages			
	Warning	poc">	On all Pages			
	Warning	poc">	On all Pages			

Powered by Rukovoditel

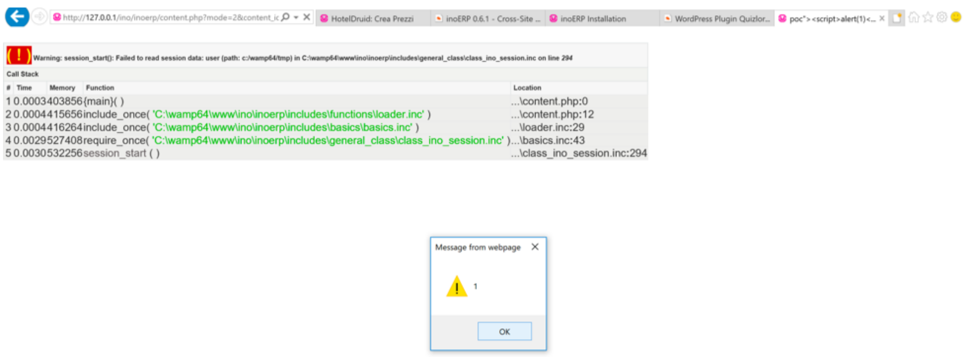
Figure 8: implementing CSP header vulnerable RukovoditelERP on FireFox, Opera and Chrome.



```
127.0.0.1/ino/inoerp/content.php

127.0.0.1 says
Request [object Object] has errored with error : Forbidden
```

Figure 9: Testing InoERP on Opera.



```
http://127.0.0.1/ino/inoerp/content.php?mode=2&content_ic

( ! ) Warning: session_start(): Failed to read session data: user (path: c:/wamp64/tmp) in C:\wamp64\www\ino\inoerp\includes\general_class\class_ino_session.inc on line 294
Call Stack
# Time      Memory Function                                                                     Location
1 0.0003 403856 {main}( )                                                                        ...\content.php:0
2 0.0004 415656 include_once( 'C:\wamp64\www\ino\inoerp\includes\functions\loader.inc' )         ...\content.php:12
3 0.0004 416264 include_once( 'C:\wamp64\www\ino\inoerp\includes\basics\basics.inc' )            ...\loader.inc:29
4 0.0029 527408 require_once( 'C:\wamp64\www\ino\inoerp\includes\general_class\class_ino_session.inc' ) ...\basics.inc:43
5 0.0030 532256 session_start ( )                                                                ...\class_ino_session.inc:294

Message from webpage
1
```

HotelDruid: Crea Prezzi
inoERP 0.6.1 - Cross-Site ...
inoERP Installation
WordPress Plugin Quizlor...
poc"><script>alert(1)<...
OK

Figure 10: Testing InoERP on Internet Explorer.

6 Results

As the project implementation included five vulnerabilities, six browsers, and three protection techniques, we have 90 experiments that summarized by the following chart and table. we compare the effectiveness of the three modern protection techniques by measuring the success rate for each solutions comparing to the other ones.

First technique is using Mod-Security web application firewall which works all the time on every browser and every vulnerable web application to prevent cross-site scripting attack. when tested against vulnerable parameter using a malicious script , mod-security WAF always redirect the user to forbidden page with a note indicating that user do not have permission to access this page on web server. The second one is tesing the content security policy header which works as 86 percent out of the 90 attempts. When tested against vulnerable parameter using the follwing script poc"><script>alert(1)</script>for simplicity, the cross-site script attack fails to work most as CSP header add a strong security layer to the browser, except Microsoft Internet Explorer that does not support CSP header. The last mechanism that designed to mitigate cross-site script vulnerability is X-XSS-Protection header which is found as not highly effective and can be replaced by content-security-policy header.

Chart 1: shows that mod-Security Web application firewall always defend against XSS, while CSP headers protect most of the time, except when applying to Internet Explorer. On the other hand, X-XSS-Protection only worked a few time.

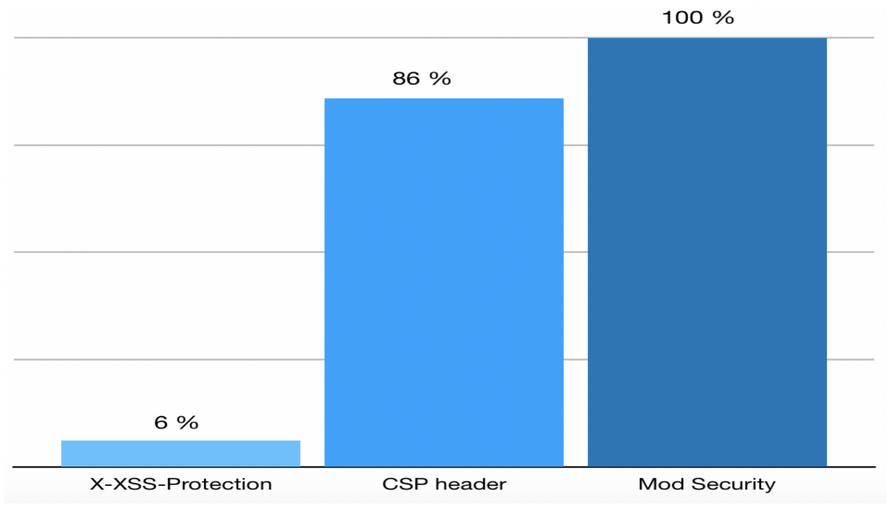


Chart1: Success rate of each protection techniques when tested against vulnerable web application

The other way to summarize the results is by categorize them by browsers and solutions as in Table 1 which shows which mechanism works with which browsers. Web application firewall Mod-Security works on every browsers, while content-security-policy headers works

Browsers	Web application firewall Mod-Security	Content Security Policy headers	X-XSS-protection headers
Chrome	✓	✓	✗
Firefox	✓	✓	✗
Opera	✓	✓	✗
Safari	✓	✗	✗
Internet Explorer	✓	✗	✗
Microsoft Edge	✓	✓	✗

Table 1: Results of testing XSS techniques on different browsers

on the four browsers, every browser except Microsoft Internet Explorer and Safari, and X-XSS-Protection is not effective with any browsers. We should note that Firefox browser does not support X-XSS-Protection as well. On the other hand, web application firewall works effectively on all browsers.

7 Conclusion

This project aims to evaluate the effectiveness of modern protection solution against XSS vulnerability on top used browsers. It mainly focuses on testing the capability of web application firewall and two security headers that are designed to mitigate cross-site-scripting attack, content-security-policy header and X-XSS-Protection header. The result of the project proves that not all XSS solution mechanism worked as expected on different browsers. Also, some of the top-used browsers do not support some of the security headers. For example, Firefox browser do not support using X-XSS-Protection headers, and Microsoft Internet Explorer does not support content-security-policy header. As our finding proves that web application firewall probably is one of the best solutions against cross-site scripting attack, we recommend using web application firewall to prevent cross-site-scripting attack and other injection attacks. Furthermore, Defense in depth by using multiple tools and techniques to protect against XSS attack is always recommended. In future work, we recommend testing more solutions against cross-site script; for example, Input/output validation and HTML entity encoding. Along with other security headers that are designed to protect from other attacks, like X-Frame-Options, Set-Cookie and X-Content-Type. Moreover, we can evaluate the effectiveness for other web application vulnerabilities such as cross-site request forgery, SQL injection and directory traversal on most popular browsers.

8 Acknowledgment

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our advisors Prof. Mishra and Prof. Olson whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report. I am also grateful to the financial support from my sponsor SACM, and I would like to thank my family for their support through all the stages of my life especially this one.

References

[1] Mahmoud, S. K., Alfonse, M., Roushdy, M. I., Salem, A. M. (2017). A comparative analysis of Cross Site Scripting (XSS) detecting and defensive techniques. 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS). doi:10.1109/intelcis.2017.8260024

[2] Dolnak, I. (2017). Content Security Policy (CSP) as countermeasure to Cross Site Scripting (XSS) attacks. 2017 15th International Conference on Emerging ELearning Technologies and Applications (ICETA). doi:10.1109/iceta.2017.8102476

[3] Singh, J. J., Samuel, H., Zavarsky, P. (2018). Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall. 2018 1st International Conference on Data Intelligence and Security (ICDIS). doi:10.1109/icdis.2018.00030

[4]Marashdih, A. W., Zaaba, Z. F., Suwais, K. (2018). Cross Site Scripting: Investigations in PHP Web Application.A 2018 International Conference on Promising Electronic E Ì Technologies (ICPET). doi:10.1109/icpet.2018.00011

[5]Taha, T. A., Karabatak, M. (2018). A proposed approach for preventing cross-site scripting.A 2018 6th International Symposium on Digital Forensic and Security (ISDFS). E Ì doi:10.1109/isdfs.2018.8355356

[6] Kaur, G., Pande, B., Bhardwaj, A., Bhagat, G., Gupta, S. (2018). Defense Against HTML5 XSS Attack Vectors: A Nested Context-Aware Sanitization Technique.A 2018 8th E Ì International Conference on Cloud Computing, Data Science Engineering (Confluence). doi:10.1109/confluence.2018.8442855

[7] Rao, K. S., Jain, N., Limaje, N., Gupta, A., Jain, M., Menezes, B. (2016). Two for the price of one: A combined browser defense against XSS and clickjacking.A 2016 International E Ì Conference on Computing, Networking and Communications (ICNC). doi:10.1109/iccnc.2016.7440629

[8] Chen, P., Yu, H., Zhao, M., Wang, J. (2018). Research and Implementation of Crosssite Scripting Defense Method Based on Moving Target Defense Technology.A 2018 5th In- E Ì ternational Conference on Systems and Informatics (ICSAI). doi:10.1109/icsai.2018.8599463

[9] Yusof, I., Pathan, A. K. (2016). Mitigating Cross-Site Scripting Attacks with a Content Security Policy.A Computer,49(3), 56-63. doi:10.1109/mc.2016.76 E Ì

[10] Mhana, S. A., Din, J. B., Atan, R. B. (2016). Automatic generation of Content Security Policy to mitigate cross site scripting.A 2016 2nd International Conference on E Ì Science in Information Technology (ICSITech). doi:10.1109/icsitech.2016.7852656 13

[11] Lavrenovs, A., Melon, F. J. (2018). HTTP security headers analysis of top one million websites. 2018 10th International Conference on Cyber Conflict (CyCon). doi:10.23919/cycon.2018.8405025

[12] Weichselbaum, L., Spagnuolo, M., Lekies, S., Janc, A. (2016). CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS16. doi:10.1145/2976749.2978363 [13] Calzavara, S., Rabitti, A., Bugliesi,

M. (2016). Content Security Problems? Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS16. doi:10.1145/2976749.2978338

[14] A Measurement Study of the Content Security Policy on Real ... (n.d.). Retrieved from https://pdfs.semanticscholar.org/10c8/b4c2b70372a4b80dc81e367c02531eea157e.pdf