VOL. 13, NO. 17, SEPTEMBER 2018 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

ATTACK DATA ANALYSIS TO FIND CROSS-SITE SCRIPTING ATTACK PATTERNS

Pmd Nagarjun[1, 2] and Shaik Shakeel Ahamad[1, 3]
[1]Department of Computer Science and Engineering, K L University, Vijayawada, India
[2]Nagabot Software Development Pvt. Ltd., Nellore, India
[3]CCIS, Majmaah University, Majmaah, Kingdom of Saudi Arabia
E-Mail: pmdnr@nagabot.com

ABSTRACT

Cross-Site Scripting (XSS) attacks are the most popular web application attacks. In XSS attacks, the attacker injects malicious code into a web application and execution of that malicious code at the browser side may steal session tokens, web cookies, or other sensitive information of the user. In this paper, we analysed a large collection of XSS attacks to find XSS attack patterns. Based on this analysis, we are able to find XSS attacks effects on different programming languages, domain extensions, and common web pages. Furthermore able to find script tags frequency, keywords frequency, and special characters frequency in XSS attacks. We also reviewed different prevention techniques of XSS attacks.

Keywords: web application attacks, cross-site scripting, XSS, malicious code, javascript.

1. INTRODUCTION

Web applications have become part of day-to-day activities. Most of the web applications are having vulnerabilities which cause web application attacks. Based on survey of Open Web Applications Security Project (OWASP) [1] among all web application attacks Cross-Site Scripting (XSS) attacks [2] are popular and dangerous attacks. XSS attacks occur because of improper validation of user input data. In Cross-Site Scripting attacks attacker inject malicious code into the web application by exploiting the vulnerability in the web application. Execution of that malicious code at web browser by the user causes stealing of session tokens, web cookies, or other user's sensitive information. XSSed Project created by Kevin and Dimitris [3] contains the largest online archive of XSS attacks happened on websites.

2. TYPES OF CROSS-SITE SCRIPTING ATTACKS

2.1 Reflected (Non-Persistent)

In this type of attacks [2], website URL request contains malicious attack code crafted by the attacker. Web server reflects this malicious code as a response to users who ever try to access that website URL. Attacker sends this malicious website URL to victims by email or messaging and trick victim to click, if the victim clicks on this malicious URL then the server sends attack code as response and that attack code will be executed at the victim's browser.

Example. Malicious URL.
http://example.com/search.php?keyword=<script>alert ("you are victim");</script>

2.2 Persistent

In this type of attacks [2], injected malicious code stored at web server's database permanently. If victim requests stored information at web server then the server may send malicious code as a response.

Example

The attacker injects malicious code as a comment in a vulnerable web page.

Comment - Great information, for more related information, check my site.
<script> alert("you are victim"); </script>

When users visit this attacked web page malicious code in that comment will be executed in their browsers.

2.3 DOM-based attacks

In DOM-based XSS attacks [4] malicious code will stay at the client's side, and there will be no involvement of server. JavaScript reflect malicious code as a response to requests handles at the client-side. This malicious code will modify the DOM environment in the victim's browser.

Example
http://example.com/user.html web page contains below code.
<script>document.write("URL:"+document.location.href);
</script>
Attacker can craft attack URL likes below.
http://example.com/user.html\#<script> alert ("you are victim"); </script>

We analysed only reflected and DOM-based Cross-Site Scripting attacks in our study.

3. LITERATURE WORK

Wurzinger *et al.* [5] proposed a server-side reverse proxy technique to mitigate XSS attacks called SWAP. In this method web application's JavaScript code is replaced with Script Ids (ex: <script> to <scrip1>) so executable JavaScript will be converted into nonexecutable JavaScript. While responding to client, checks for JavaScript, if there is any JavaScript then it

VOL. 13, NO. 17, SEPTEMBER 2018 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2018 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

considered an attack, if no JavaScript means safe then decode all Script Ids and send a response to the client.

Scholte *et al.* [6] conducted a study on vulnerabilities related to input validations and corresponding attacks like XSS and SQL Injection. They collected more than 7000 vulnerabilities reports from the National Vulnerability Database (NVD). They studied 78 popular web frameworks of different programming languages. Their study states that strongly typed languages provide more security and 50% of frameworks not implemented any sanitizing functions for input validations.

Moen *et al.* [7] conducted a study on vulnerabilities in Government (.gov) sites and shows that 80% of E-Government websites were vulnerable to either XSS or SQL Injection attacks.

Chandra and Selvakumar [8] proposed an XSS sanitization tool named BIXSAN (Browser Independent XSS Sanitizer) which works on the server-side. This tool filters static tags and allows the static tags and removes all dynamic tags. It can prevent reflected and stored Cross-Site Scripting attacks by creating document DOM on server-side and use this DOM at the client-side.

Bates, Barth, and Jackson [9] proposed an XSS filter which works on the client-side. In this method XSS filter act as mediate between the HTML parser and JavaScript engine. Compare request and response post data semantics, if there is any malicious JavaScript then it will block that script before reaching JavaScript engine.

Kirda *et al.* [10] developed a client-side web application firewall named Noxes. This works as a personal firewall and proxy. If the user requests any URL then it checks the filter list to validate URL before sending to the server. There will be an alert box for every new URL which needed to be validated by the user. To avoid so many alerts there will be a threshold (k), for each page can have k external links and these are considered valid for one click or that session.

Kals *et al.* [11] proposed and implemented a web application vulnerability scanner named Secubat. SecuBat works based on generic attack filtering technique so no need of large database of known vulnerabilities. Main modules of SecuBat (Crawl, Attack, and Analysis) can run independently. According to their testing, almost 5% of websites are vulnerable to XSS attacks. SecuBat able to find Reflected XSS attacks, Encoded Reflected XSS attacks, Form-Redirecting XSS attacks, and SQL Injection vulnerabilities.

Garn *et al.* [12] conducted tests on five popular broken web application projects to find Reflected XSS and Stored XSS vulnerabilities. The process involves creating attack vectors (tests) and use those attack vectors as inputs to penetration testing tools like Burp suite, OWASP ZAP tool. They stated that the accuracy of penetration tools depends on encoding techniques used by tools.

Hydara *et al.* [13] conducted a literature review on 115 research papers related to XSS attacks from 2004 to 2012. Their study shows that most of the work done on detecting and preventing XSS attack vulnerabilities. Only two studies show removing vulnerabilities from source code. Their study also shows that Reflected XSS attacks are popular XSS attacks compared to other types of XSS attacks.

Wassermann and Su [14] proposed a static analysis process to find XSS vulnerabilities. This process involves string taint analysis and preventing untrusted scripts. They stated that improper input validation is one of the main reason for XSS attacks. They tested their approach on popular PHP open source web applications and they are able to detect XSS vulnerabilities in those applications. Their static analysis method able to detect only Reflected and Stored XSS attacks.

Gupta *et al.* [15] proposed a tool named XSSDM which uses pattern matching methods and static analysis to detect XSS vulnerabilities in web applications. They stated that their approach shows promising results to detect XSS vulnerabilities in various HTML context compared to popular XSS vulnerability detecting tools like Pixy and RIPS.

Guo, Jin, and Zhang [16] proposed a method which uses an optimized repository of XSS attack vectors. This proposed method involves three stages in the first stage they generated a number of XSS attack patterns based on XSS attack grammar. In the second stage, they optimized those XSS attack repository to improve the efficiency of detection. In the third stage, they used this optimized attack repository to detect XSS vulnerabilities dynamically. They claim their method shows a promising result in detecting XSS vulnerabilities with fewer performance issues.

Mao [17] provided a security testing framework which contains different types of tests to conduct proper web application security testing. They conducted web application testing in mainly two aspects of static analysis and dynamic simulation. By using their testing framework they found some security problems in wecoo.com website.

Shahriar and Zulkernine [18] implemented a tool named MUTEC in PHP language which performs testing based on the mutation to find XSS vulnerabilities. 11 mutation operators were proposed by them which will modify PHP code and JavaScript code. By using MUTEC tool they are able to find XSS vulnerabilities on 5 open source applications.

Javed [19] did security analysis on top 25 popular rich text editors or WYSIWYG editors to find XSS vulnerabilities. Some of the popular editors involved in their security analysis were EditLive, TinyMCE, PHPHTMLEditor etc these editors used by thousands of websites. They also analysed rich text editors used in popular websites like Amazon, Yahoo mail, Twitter, Github etc. They used stepwise systematic attack methodology to analyse these WYSIWYG editors. They performed attacks on popular features of these rich text editors like link insertion, image insertion, video insertion etc. Their analysis shows that all 25 WYSIWYG editors were having XSS vulnerabilities.

Kazanavicius *et al.* [20] proposed the Embedded Web Application Firewall (EWAF) based on the blacklist and whitelist filters. Their tests show that whitelist filters are better compared to blacklist filters at high loading situations. Based on user request EWAF analyse which attacks were possible like XSS, SQL Injection, etc. After analysing possibilities of attacks, the request sent through corresponding XSS module or SQL Injection or other modules based on attack type. Then take a decision whether the request is good or it contains attack string based on results of corresponding attack module.

Yusof and Pathan [21] tested the effectiveness of Content Security Policy (CSP) on a prototype website that runs on Apache web server. They collected XSS attacks from different XSS cheat sheets and used 50 unique vectors to simulate all 3 types of XSS attacks. They conducted these attack tests on four popular browsers Firefox, Chrome, Opera and OS X Yosemite. Their test results show that the CSP technique able to prevent all tested XSS attacks on all four browsers.

Shanmugasundaram, Ravivarman, and Thangavellu [22] studied different prevention techniques for XSS attacks. They stated that websites contain XSS vulnerabilities because developers or users don't have sufficient knowledge on XSS problems, developers are unable to implement existing solutions in their applications, improper encoding of untrusted data and most of the existing tools only prevent reflected and stored XSS attacks they are unable to prevent DOM-based XSS attacks.

4. OUR ANALYSIS METHOD

We collected Cross-Site Scripting attack URLs from the XSSed website [3] from 2010 to 2015. All attack URLs are converted into a readable format. The total number of Cross-Site Scripting attacks collected were 9421. We collected information related to top 1000 websites from Alexa website [23]. Simple examples of encoded XSS attack URLs along with their decoded format are given below.

Example-1.
Attack URL:
https://www.example.edu/psearch.php?page_not_found=%3Ch1%3EYou%20are%20%3Cbr%3E%20XSS%20Attacked%3C%2Fh1%3E%20%3Cscript%3Ealert%20(%22Your%20Cookie%20is%3A%22%20%2Bdocument.%20cookie%20)%20%3C%2Fscript%3E

Decoded Attack URL:
https://www.example.edu/psearch.php?page_not_found=<h1>You are <br> XSS Attacked</h1> <script>alert ("Your Cookie is:" +document. cookie) </script>

Example-2.
Attack URL:
https://www.example.edu/puser/pdata/showuserprofile.php?WEBID=r5nrr3wBe7CiV7PxHiU1xcAhmCUvl&PID=P141923&SSHTOKEN=9089881&ID=1919&NAME=USER&WELMSG=%3Cscript%20%3Edocument%20.getElementById%20(%22clicklink%22)%20.innerHTML%3D%20%22http%3A%2F%2Fexample.net%2Fattackwebsite.php%22%20%3B%3C%20%2Fscript%3E

Decoded Attack URL:
https://www.example.edu/puser/pdata/showuserprofile.php?WEBID=r5nrr3wBe7CiV7PxHiU1xcAhmCUvl&PID=P141923&SSHTOKEN=9089881&ID=1919&NAME=USER&WELMSG=<script>document .getElementById ("clicklink") .innerHTML= "http://example.net/attackwebsite.php" ;< /script>

Example-3.
Attack URL:
https://www.example.edu/usermsg.asp?user-msg=%3Cscript%20%3Edocument%20.write%20(%22%3Ca%20href%3D%27http%3A%2F%2Fexample.edu%2Fsessionsteal.asp%27%20%3E%20Songs%3C%2F%20a%3E%22)%3B%3C%2Fscript%3E

Decoded Attack URL:
https://www.example.edu/usermsg.asp?user-msg=<script>document.write("<ahref='http://example.edu/sessionsteal.asp' > Songs</ a>");</script>

5. RESULTS

We analysed a large collection of XSS attacks to find XSS attack patterns. Figure-1 shows extracted information after analysing XSS attacks.

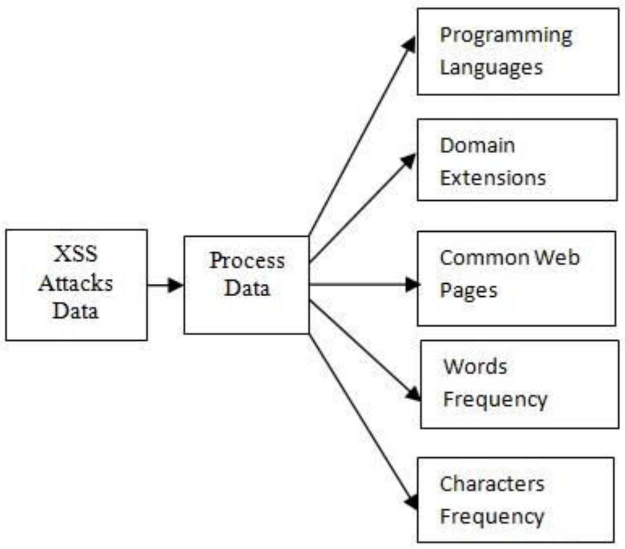

Figure-1. Extracted information from XSS attacks.

5.1 Attacks on websites

Total XSS attacks were 9421. Figure-2 shows Government and popular websites were attacked 2941 times. Top 1000 websites attacked 882 times.

VOL. 13, NO. 17, SEPTEMBER 2018 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2018 Asian Research Publishing Network (ARPN). All rights reserved.

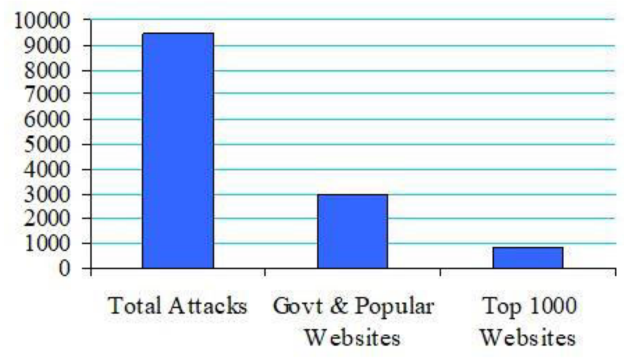

Figure-2. Total attacks on websites.

Figure-3 shows multiple attacks on the same websites, 861 websites were affected multiple times, 611 government and popular websites attacked multiple times. In top 1000 websites, 8 websites were attacked multiple times.

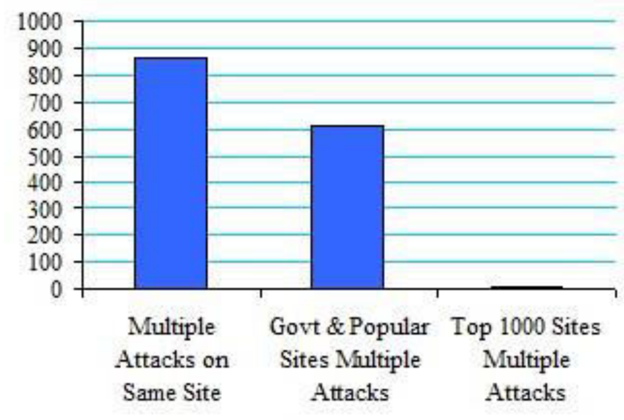

Figure-3. Multiple attacks on the same website.

5.2 Programming languages analysis

We analysed XSS attacks on web pages developed with different programming languages. From Figure-4 shows that almost half of XSS attacks were on websites developed with PHP language. 82.4% [24] of websites were developed with PHP. PHP is easy to learn and easy to develop language so new developers write insecure PHP applications which cause a lot of vulnerabilities in those websites [25].

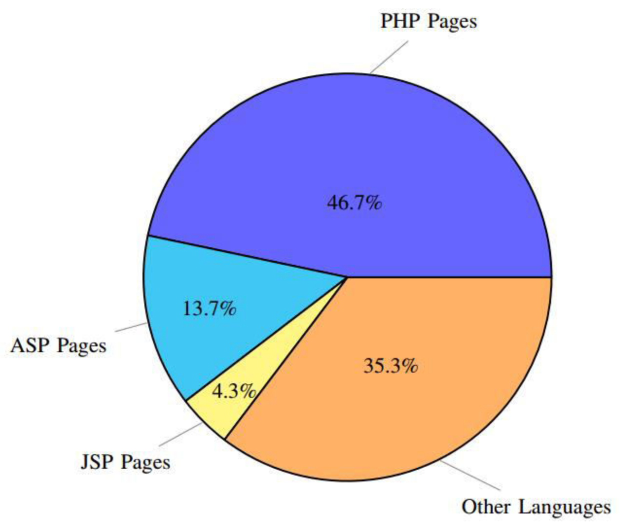

Figure-4. Attacks on web pages developed with different programming languages.

5.3 Domain extensions analysis

We analysed XSS attacks on different domain extensions. Figure-5 shows that attackers always try to attack popular domain extensions like .com, .org, .gov, .edu etc.

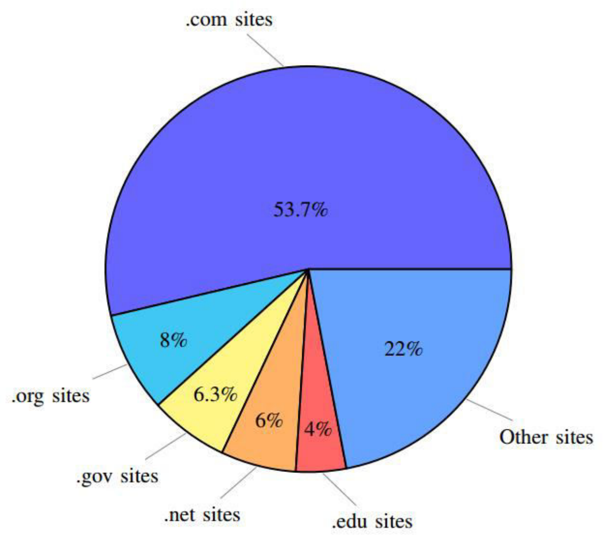

Figure-5. Attacks on domain extensions.

From Table-1 total .gov websites [26] were <0.1% but total attacks on .gov sites were 6.3%. Similarly .edu websites were 0.1% but attacks on .edu sites were 4%. It shows that attackers are more focused on exploiting Government and Educational websites because of weak security measures were taken by these websites.

Table-1. Top level domain extensions and attacks.

Domain extension	% of Attacks	% of Websites
.com	53.7%	48.5%
.org	8%	4.7%
.gov	6.3%	<0.1%
.net	6%	4.6%
.edu	4%	0.1%

5.4 Web pages analysis

Attackers focus on the functionality of different web pages to find common vulnerabilities to exploit them. We analysed different web pages based on their functionality like search, error reporting, login pages etc., and a number of attacks occurred on them, see Figure-6.

Coding secure search functionality is one of the difficult tasks in web application development. Figure-6 shows that almost 30% of attacks were performed on search pages. Instead of writing own search feature, developers can use the popular Google custom search [27] to avoid risk being attacked through vulnerabilities in own search pages.

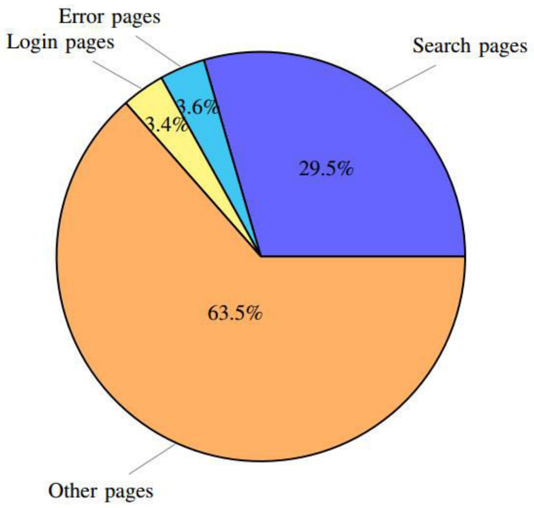

Figure-6. Attacks on particular web pages.

5.5 Script tags analysis (Script word in number of attacks)

Among all 9421 XSS attacks, 7632 attacks contains "script" word, 7226 attacks contains "<script" word, 6213 attacks contains "</script>" word, 1221 attacks contain only "<script" word, 208 attacks contain only "</script>" word and 6005 attacks contain both "<script", "</script>" words. Figure-7 and Figure-8 shows that script open (<script>) and end (</script>) tags were involved in almost all attacks.

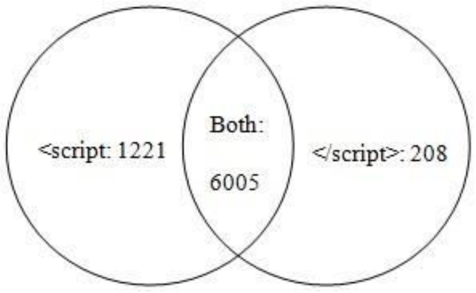

Figure-7. <script and </script> tags analysis in total attacks.

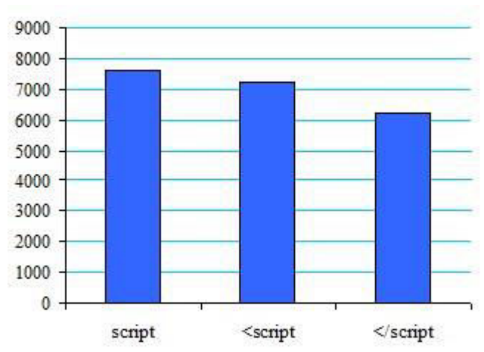

Figure-8. Script, <script and </script> tags analysis in total attacks.

By proper validation of these script tags from input data, most of the attacks can be avoided.

5.6 Keywords analysis

We analysed different keywords involved in XSS attacks like document.cookie, iframe, onmouseover etc. shown in Table-2. Among all keywords after "alert" keyword, "document.cookie" is a common keyword in most attacks.

The keywords shown in Table-2 may not be the original keywords involved in actual attacks. Because attacks showed by XSSed were validated by them. Instead of malicious code, they replace with simple alert examples.

Table-2. Keywords used in XSS attacks.

Keyword	Number of occurrences
alert	7707
document.cookie	2503
Iframe	858
onmouseover	110
eval	91

5.7 Characters analysis - Frequency of special characters in all attacks

We analysed how frequently some special characters involve in XSS attacks. The frequency of special characters is shown in Table-3.

Table-3. Frequency of special characters in XSS attacks.

Special characters	Number of occurrences
>	37248
.	37020
<	30470
“	11853
)	10066
(	9983
‘	4280

From Figure-9 shows that <, > characters were involved in almost all attacks. So to perform a successful Cross-Site Scripting attacks <, > are necessary. This can be avoided by proper validation of input data.

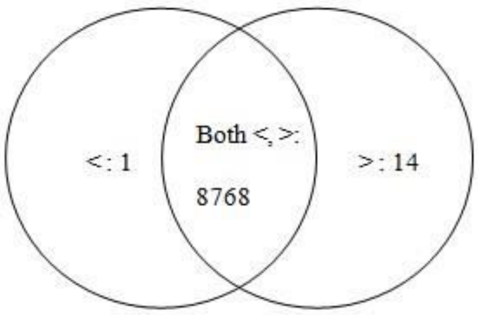

Figure-9. Frequency of < and > characters in total attacks.

6. XSS PREVENTION TECHNIQUES

6.1 Validating user data

One of the root causes of XSS attacks is an improper validation of user data [14]. Developers need to validate user data properly at server-side.

Some of the validation checks on server-side will be username need to be only alphanumeric, user age needs to be an integer, etc.

In PHP language developers uses preg_match(), filter_var() and other functions to validate user data. preg_match() function uses regular expressions to validate data.

Example-1. Validate name.

```
$username = $_POST["username"];
if (!preg_match("/^[a-zA-Z ]*$/",$username)) {
  $printError = "Only white space and letters allowed";
}
```

Example-2. Validate URL.

```
$useremail = $_POST["useremail"];
if(!filter_var($useremail, FILTER_VALIDATE_EMAIL)) {
  $printError = "Enter Valid Email";
}
```

Scholte *et al*. [6] study show that 50% of popular web frameworks not implemented proper techniques to validate input data.

6.2 Encoding/Escaping data

Developers need to implement web applications which will escape untrusted data before displaying in the browser to the user.

Encoding/Escaping is a primary technique to defend against XSS attacks [28]. Different escaping schemes [29] are used based on locations, where untrusted user data inserted in HTML document. Table-4 shows different escaping schemes with example locations.

6.3 Content Security Policy (CSP)

CSP restricts browsers to only allow trusted recourses like scripts, images, videos, etc. Developers or Server administrators can create a whitelist of trusted recourses [30]. Those trusted details are sent to the browser as security policies in HTTP response from web server. Most of the XSS defense techniques prevent one or two types of XSS attacks but the CSP method able to prevent all three types of XSS attacks.

By using CSP rules developer can block untrusted external recourses, can disable in-line JavaScript and CSS codes and can disable eval function of JavaScript.

Table-4. Escaping schemas and corresponding HTML document locations.

Escaping schemas	HTML document locations
HTML Escape	<div> Escaped HTML data here </div>
Attribute Escape	<div style="Escaped Attribute data here" > text </div>
JavaScript Escape	<script> alert("Escaped JavaScript Data") </script>
CSS Escape	<div style="color: Escaped CSS data here"> text </span>
URL Escape	<a href="http://www.example.com?data= Escaped URL data">click here</a >

To enable CSP developer need to include Content Securit Policy HTTP header.

Example. Content Security Policy HTTP header.

```
Content-Security-Policy:
script-src 'self' testscripts.example.net;
media-src 'self';
style-src 'self';
img-src *;
default-src 'none';
```

Above CSP policy restricts scripts loading from testscripts.example.net or same origin (host), restricts loading of audio, video files from the same origin, restricts loading of style sheets to the same origin, images can be downloaded from any host and all other resources are restricted to download from any host.

7. CONCLUSIONS

Our results show that websites developed with PHP language were attacked more frequently with XSS attacks. Furthermore shows that implementing a search feature in web applications involve a lot of risk being attacked, to avoid this developer can use Google Custom Search module. < and > characters are not common characters used in user input data. So by properly filtering special characters mainly < and > can avoids 90% of Cross-Site Scripting attacks. Compared to normal websites more XSS attacks happening on government and educational websites so these sites maintainers need to follow proper security measures. Developers need to properly validate user input data, escape untrusted data and implement Content Security Policy to avoid Cross-Site Scripting attacks.

REFERENCES

[1] OWASP. 2016. Owasp top ten project, https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project

[2] Grossman J. 2007. XSS Attacks: Cross-site scripting exploits and defense. Syngress.

[3] Fernandez K. & Pagkalos D. 2007. XSSed Project.

[4] OWASP. 2015. DOM Based XSS, https://www.owasp.org/index.php/DOM_Based_XSS

[5] Wurzinger P., Platzer C., Ludl C., Kirda E. & Kruegel C. 2009, May. SWAP: Mitigating XSS attacks using a reverse proxy. In Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems (pp. 33-39). IEEE Computer Society.

[6] Scholte T., Robertson W., Balzarotti D. & Kirda E. 2012, March. An empirical analysis of input validation mechanisms in web applications and languages. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing (pp. 1419-1426). ACM.

[7] Moen V., Klingsheim A. N., Simonsen K. I. F. & Hole K. J. 2007. Vulnerabilities in e-governments. International Journal of Electronic Security and Digital Forensics. 1(1): 89-100.

[8] Chandra V. S. & Selvakumar S. 2011. Bixsan: Browser independent XSS sanitizer for prevention of XSS attacks. ACM SIGSOFT Software Engineering Notes. 36(5): 1-7.

[9] Bates D., Barth A. & Jackson C. 2010, April. Regular expressions considered harmful in client-side XSS filters. In: Proceedings of the 19th international conference on World wide web (pp. 91-100). ACM.

[10] Kirda E., Kruegel C., Vigna G. & Jovanovic N. 2006, April. Noxes: a client-side solution for mitigating cross-site scripting attacks. In Proceedings of the 2006 ACM symposium on Applied computing (pp. 330-337). ACM.

[11] Kals S., Kirda E., Kruegel C. & Jovanovic N. 2006, May. Secubat: a web vulnerability scanner. In: Proceedings of the 15th international conference on World Wide Web (pp. 247-256). ACM.

[12] Garn B., Kapsalis I., Simos D. E. & Winkler S. 2014 July. On the applicability of combinatorial testing to web application security testing: a case study. In Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing (pp. 16-21). ACM.

[13] Hydara I., Sultan A. B. M., Zulzalil H. & Admodisastro N. 2015. Current state of research on cross-site scripting (XSS)–A systematic literature review. Information and Software Technology. 58, 170-186.

[14] Wassermann G. & Su Z. 2008 May. Static detection of cross-site scripting vulnerabilities. In Proceedings of the 30th international conference on Software engineering (pp. 171-180). ACM.

[15] Gupta M. K., Govil M. C., Singh G. & Sharma P. 2015, August. XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications. In Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on (pp. 2010-2015). IEEE.

[16] Guo X., Jin S. & Zhang Y. 2015, September. XSS vulnerability detection using optimized attack vector repertory. In Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2015 International Conference on (pp. 29-36). IEEE.

[17] Mao C. 2009, November. Experiences in security testing for web-based applications. In Proceedings of the 2nd International Conference on Interaction

Sciences: Information Technology, Culture and Human (pp. 326-330). ACM.

[18] Shahriar H. & Zulkernine M. 2009 May. Mutec: Mutation-based testing of cross site scripting. In Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems (pp. 47-53). IEEE Computer Society.

[19] Javed A. 2014. Revisiting XSS Sanitization, http://www.blackhat.com/docs/eu-14/materials/eu-14-Javed-Revisiting-XSS-Sanitization-wp.pdf

[20] Kazanavicius E., Kazanavicius V., Venckauskas A. & Paskevicius R. 2012. Securing web application by embedded firewall. Elektronika ir Elektrotechnika. 119(3): 65-68.

[21] Yusof I. & Pathan A. S. K. 2016. Mitigating Cross-Site Scripting Attacks with a Content Security Policy. Computer. 49(3): 56-63.

[22] Shanmugasundaram G., Ravivarman S. & Thangavellu P. 2015, April. A study on removal techniques of Cross-Site Scripting from web applications. In: Computation of Power, Energy Information and Communication (ICCPEIC), 2015 International Conference on (pp. 0436-0442). IEEE.

[23] Alexa. 2017. Alexa top 1 million sites, http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[24] W3techs.com. 2016. Usage statistics and market share of php for websites, https://w3techs.com/technologies/details/pl-php/all/all

[25] OWASP. 2016. Php security cheat sheet, https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet

[26] W3techs.com. 2016. Usage of top level domains for websites, https://w3techs.com/technologies/overview/top_level_domain/all

[27] Google. 2017. Google custom search engine, https://cse.google.co.in/cse/

[28] Cross-site scripting. 2017. Wikipedia, the Free Encyclopedia. https://en.wikipedia.org /w/index. php?title=Cross-site_scripting&oldid= 783156453

[29] OWASP. 2017. XSS (Cross Site Scripting) Prevention Cheat Sheet, https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

[30] Mozilla. 2017. Content Security Policy (CSP), https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.