Journal of Advances in Science and Engineering, ISSN: 2636-607X
Volume 1 Number 2 (2018): 25-35

Journal of Advances in Science and Engineering (JASE)

Cross Site Scripting Attacks in Web-Based Applications: A Critical Review on Detection and Prevention Techniques

[1]Aliga, P. A., [2*]John-Otumu, A. M., [3]Imhanlahimi, R. E., & [4]Akpe, A. C. E.

[1,3] Department of Computer Science, Ambrose Alli University, Ekpoma, Nigeria; [2,4] Directorate of ICT, Ambrose Alli University, Ekpoma, Nigeria

*Corresponding author's e-mail: macgregor.otumu@gmail.com

Abstract

Web-based applications has turn out to be very prevalent due to the ubiquity of web browsers to deliver service oriented application on-demand to diverse client over the Internet and cross site scripting (XSS) attack is a foremost security risk that has continuously ravage the web applications over the years. This paper critically examines the concept of XSS and some recent approaches for detecting and preventing XSS attacks in terms of architectural framework, algorithm used, solution location, and so on. The techniques were analysed and results showed that most of the available recognition and avoidance solutions to XSS attacks are more on the client end than the server end because of the peculiar nature of web application vulnerability and they also lack support for self-learning ability in order to detect new XSS attacks. Few researchers as cited in this paper inculcated the self-learning ability to detect and prevent XSS attacks in their design architecture using artificial neural networks and soft computing approach; a lot of improvement is still needed to effectively and efficiently handle the web application security menace as recommended.

Received 20 July 2018; Revised 3 Sept. 2018; Accepted 7 Sept. 2018; Available online 15 Sept. 2018.

Keywords

Internet;
Prevention techniques;
XSS Attack;
WebApp;
Web vulnerability.

1. Introduction

A web-based application is any software application, which can be used on the Internet or intranet (Adekunle, 2014). According to Jacob *et al.* (2016a) web applications are seen as programs that can make request to a web server using protocol such as HTTP. Web applications are everywhere today because of web browser popularity. It is the ability of the web application to perform updates without transferring and installing the application on any client oriented devices that gave birth to this popularity and also for their cross platform compatibility support. Web based applications may include banking solutions, e-commerce solutions, online stores and so on. A web-based application can sometimes be referred to as a client-based application, where some part of the application is resident on a client based device while the processing is done on a web server over the Internet or intranet. However, before now, every single web page was sent to their client as a non-dynamic document, while the series of these web pages could create collaborative web proficiencies.

The introduction of a scripting language for client side application development called JavaScript in 1995, gave programmers the extra ability to add dynamic elements to web interfaces that could be executed on client end. In other words, programmers can develop applications with embedded scripts that can perform task such as validation of input forms, walloping or illuminating some part of web pages, and other simple computations, rather than transporting the data to a web server for processing on the server end. Macromedia presented vector animation player software called Flash in 1996. The flash software can be a supplementary to web browsers in a form of plug-ins to insert computer graphics such as cartoons on web pages. The flash player also permitted the use of a scripting language to code connections on the client end without the need to link with the server.

The notion of web application was presented to Java language in the server specification version 2.2 in 1999, though both JavaScript and XML had previously been developed. Nevertheless, Ajax had still not hitherto

Article citation: Aliga, P. A., John-Otumu, A. M., Imhanlahimi, R. E., & Akpe, A. C. E. (2018). Cross Site Scripting Attacks in Web-Based Applications: A Critical Review on Detection and Prevention Techniques. *Journal of Advances in Science and Engineering, 1*(2): 25-35. https://doi.org/10.37121/jase.v1i2.19

been invented and the XML and HTTP request object had merely been recently announced on Internet Explorer 5 as an energetic entity. The word Ajax was devised in 2005 and software applications like Gmail started to style their client ends more collaborative, in which a web page script is intelligent enough to exchange with the server for keeping/repossessing records devoid of transferring a complete web page. Lastly, HTML5 was completed in 2011; it delivered lifelike and audio-visual aid abilities deprived of the essentials of client side plug-ins. HTML5 similarly enhanced the semantic content of the web documents.

1.1 Organisation of Web-Based Applications

Web applications are commonly fragmented into coherent layers known as "Tiers" where each tier is allocated a role. Traditional applications are stereotypically made up of single tiers, which exist on the client machine; web applications offer themselves to n-tiered methodology naturally.

The most shared organisation of web applications is the 3-tiered application (beginning from topmost to bottommost): (a) presentation tier; (b) application logic tier; and (c) storage tier as shown in Figure 1. The web browser is the 1st or topmost tier (presentation), followed by an engine using some active web content technology such as ASP, PHP, Python, Ruby-on-Rails, or Struts is the middle tier (application logic tier), while the database is the 3rd or bottommost tier (storage). The browser directs requests to the middle tier, which services them by making interrogations (queries) and updates for the database and finally generates a user interface.

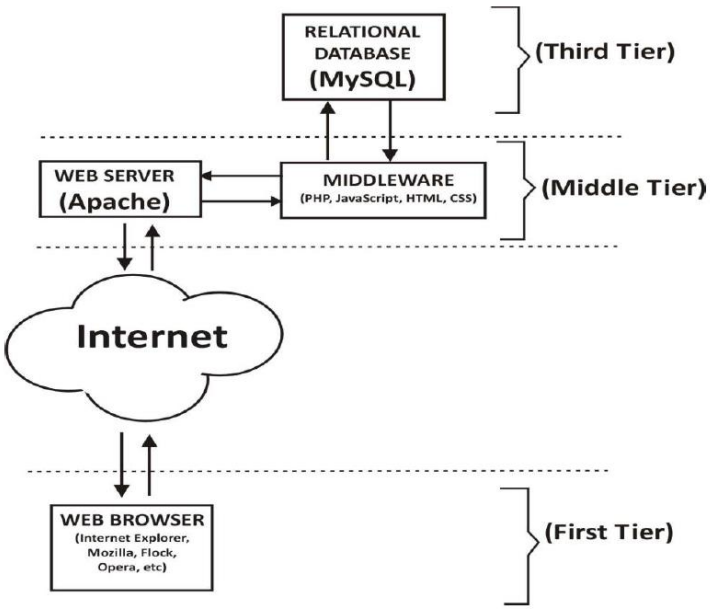

Figure 1. Web application organisation (3-Tiered) (John-Otumu *et al.*, 2015)

There are several known strength of web based applications (Adekunle, 2014): the web applications do not require any difficult technique to install in big establishments. A well-suited web browser is all that is required. In addition, browser applications normally involve pint-sized or no disk space on the client machines. Web applications require no upgrade technique since all novel features are carried out on the server machine and it is spontaneously transported to the handlers. They easily fit in to other server side web processes. Moreover, web applications operate within a web browser and therefore deliver cross-platform compatibility. The initiation of HTML5 has enabled programmers to build richly collaborative backgrounds natively inside browsers.

Despite the numerous benefits associated with web applications, they also possess serious level of drawbacks as enumerated further down. For example, in training, web interfaces when matched to thick clients, naturally power substantial disadvantage to user know-how and rudimentary usability. Also, web applications categorically necessitate well-suited web browsers. Standards passivity is a problem with any non-typical office document initiator, which originates glitches when file sharing and collaboration turn out to be precarious. In addition, browser applications depend on application files retrieved from faraway servers over the Internet. Hence, whenever link is intermittent, the application will terminate. Furthermore, web applications rest totally on the availability of the server transporting the application. Therefore, it will be to the users' detriment if the server crashes or the hosting company shuts down. The firms that make available web-based applications can

 I S S N : 2 6 3 6 - 6 0 7 X

hypothetically trail everything the users do. Finally, the current security trend of web application attack (SQL injection and XSS) shows a high level of vulnerability within web based applications.

According to Adhyaru (2016) there are numerous confidentiality threats in web applications and countless websites are presently being hacked on a day-to-day basis by unidentified personalities. The website is under attack for diverse motives as revealed in Table 1.

Table 1. Reasons for web application attacks

Attack Objective	Proportion (%)
Theft of delicate facts	42
Mutilation	23
Planning Malware	15
Unidentified	08
Treachery	03
Coercion	03
Link Spam	03
Worm	01
Phishing	01
Data Conflict	01

Source: Open Web Application Security Project (OWASP), 2013

1.2 Cross Site Scripting

Cross Site Scripting (XSS) is a common weakness that exist in web applications, which allows an attacker to insert malicious code (client-side script) into web pages (Imhanlahimi *et al.*, 2017; Kamal *et al.*, 2015; Umasankari *et al.*, 2013). The concept of XSS allows an attacker to execute scripts in the target browser, which can result to users' session being take over, despoilment of websites, port scan of internal networks (Raghuvanshi and Tiwari, 2014).

XSS attack vectors are normally categorised into three types (Imhanlahimi *et al.*, 2017; Jacob *et al.*, 2016b; Raghuvanshi and Tiwari, 2014): DOM-based XSS (Type0); Reflected XSS (Type1), and Stored XSS (Type2). XSS attack is measured one of the top ten web application susceptibilities of 2013 by the Open Web Application Security Project (OWASP, 2013). According to Cenzic Application Vulnerability Trends Report as illustrated in Figure 2 (CAVTR, 2013), XSS attacks characterises 26% of the total population of web application security vulnerability and this is considered as the top most attack. XSS attacks are usually being executed with JavaScript, HTML, ActiveX, VBScript, Flash, etc.

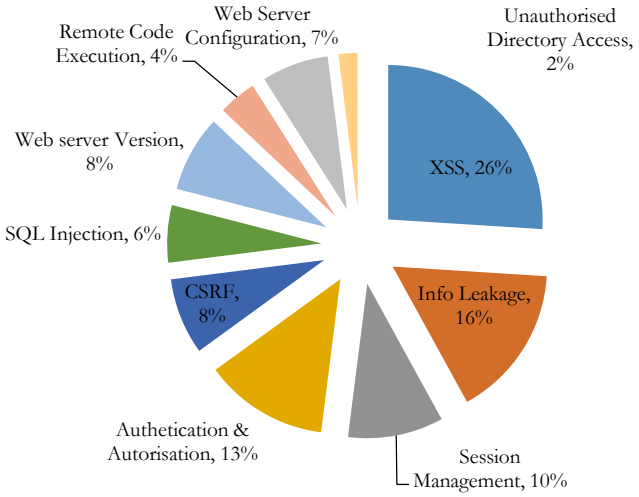

Figure 2. Web application security vulnerability chart (CAVTR, 2013)

A feeble input confirmation on the web application normally leads to XSS attacks in order to collect facts from account by stealing, altering user's settings and cookie theft (Nithya *et al.*, 2015). The detection and prevention of XSS is now a topic of dynamic investigation in the industry and academia since the automated tools and security system or mechanisms executed are not whole or precise enough to promise an unconditional level of confidence on the web application (Nithya *et al.*, 2015). Consequently, a mechanism which will be

effortlessly deployed, and that which will also offer a worthy performance in detecting and preventing XSS attack is of utmost interest in this paper.

According to Rohilla *et al.* (2016) solution measures to XSS attacks can be located on the client end, server end or both ends; however, a major disadvantage of the server side solution is that it does not interrupt HTTP and SSL connections. Certain conditions or behaviour like mischievous scripts implementation, takeover user periods, users linking to a mischievous server, users entrance to counterfeit URL, unprotected secure socket layer encoded networks, infected cookies attack, form alteration, browsers execution limitation, implementation of programs on the client machine that makes weak server data noticeable, theft of login particulars like username and passwords, failure of web sites to identify character set, browsers understanding of information they receive from servers, captivating complex information from clients and their website activities, theft of business data, keystroke recording of clients that officially visit a web site, taking down a web site, initiating Denial of Service, takeover of a site for a ransom, redirecting visitors to a wrong websites, and launching data for unlawful acquirement may necessitate the solution deployment location to prevent XSS attack. However, the solution to the XSS attack may be located either on the client side or on the server side or even a hybrid approach based on the conditions perceived.

2. Related Works

The constant security threat of XSS attack in web applications has become very worrisome in the research community because web application is used for different kinds of business models over the Internet to meet different organisations and clients' needs. This section critically reviews some recent XSS detection and prevention mechanisms or models in terms of architecture, framework, algorithm used, and solution side.

Rao *et al.* (2017) proposed a strategy for restricting cross site scripting attacks using the code filtering algorithm. The algorithm works by sanitising users input that may contain any scripting tags/codes, thereby preventing the scripting tags / codes from being stored or executed from the database of the web application. Results showed that an attacker normally checks whether a web application is vulnerable or not before the attacker tries to inject the malicious code into the server. Also found from the experimental result was the case of restricting the redirection of a specific web application page to some other pages through which an attacker can be stopped.

Imhanlahimi *et al.* (2017) proposed an intelligent defensive model for identifying and preventing XSS attack in web applications using soft computing (SC) approach, which is made up of the following components described in (1), where, *GA* is genetic algorithm, *NN* is neural network and *FL* is fuzzy logic:

$$SC = GA + NN + FL. \quad (1)$$

The researchers observed that while other forms of attack affects the server side; the XSS affect the client side (user) i.e. the real manipulation is inside the target's web browser while the defencelessness lies in the server side. They also noted that web applications accept multifaceted Hypertext Markup language (HTML) input from operators and the inconsistent web browser performances serving as vectors for fruitful XSS attacks made the encounter very problematic to unravel. However, an enhanced security browser was developed for identifying and preventing the execution of XSS codes on web applications using the three fundamental components of the soft computing model proposed as described in the mathematical notation above. The techniques used provided not only a self-learning ability but also had the ability to take care of the greatest problem of the XSS scripting code which is the inability of the browser to differentiate between benign and malicious codes. Now, because computing devices are being operated by both technical as well as non-technical users, the model offers every user equal security measure, as non-technical users do not have the fear of not being able to successfully select the right policies to apply in order to protect their sensitive data as found in measures that are based on policies. Results from the developed system suggested that despite the anomalous behaviour of browser, the proposed approach is very effective amongst widely deployed existing web browsers.

Goswami *et al.* (2017) proposed an effective proxy-based client/server method for detecting XSS attack. The method concentrated on matching the load between client and the server. It executes a preliminary scrutiny in the client side for susceptibility using different measure. If the doubt level surpasses outside an edge value, then the request is discarded, else, it is advanced to the proxy for more processing. The proposed method used unsupervised grouping technique reinforced by vigorous combination technique to identify and group confounded JavaScripts. The proposed proxy based method endeavoured to poise the load in both the client side and the server side. A mainstream of the discovery mission is carried out by the proxy, whereas the preliminary check for weakness is prepared in the client side. Geetha *et al.* (2017) suggested a client side way out for stopping XSS attacks using automata-based symbolic string susceptibility analysis and susceptibility mark generation referred to as XHunter for spontaneous confirmation of string changing codes. The technique is

 I S S N : 2 6 3 6 - 6 0 7 X

cross platform and has been executed on a cross platform browser, so it can be used with other operating systems with little manipulation. The investigation takes an attack pattern stated as a consistent expression and a JSP code as input which can be executed as a common solution to be used in all available web browsers. The results revealed that the proposed methodology is very effective.

Jacob *et al.* (2016b) studied several existing detection techniques for XSS attacks on both client and server end. They proposed an expert system for detecting both persistent and non-persistent XSS attack that is also capable of working at both client and server side based on the shortcomings of existing systems. The proposed detection model is capable of validating both the users input delivered at the client side and the reaction web pages from the server side. The model has dissimilar structural design for both the client and server side. The untiring XSS attack comprises five sections: (input sanitizer, filtering module, filtered output, attack rule library and attack repository). The non-persistent detection of XSS attack also comprises of five blocks: (input checker, prevention using content security policy (CSP), notify client, attack rule library and attack repository), and the detection of XSS attack at the server side which also comprises five parts: (feature injection, policy storage, web server, output response deviator, sanitization and feature removal). Figure 3 shows the proposed recognition system.

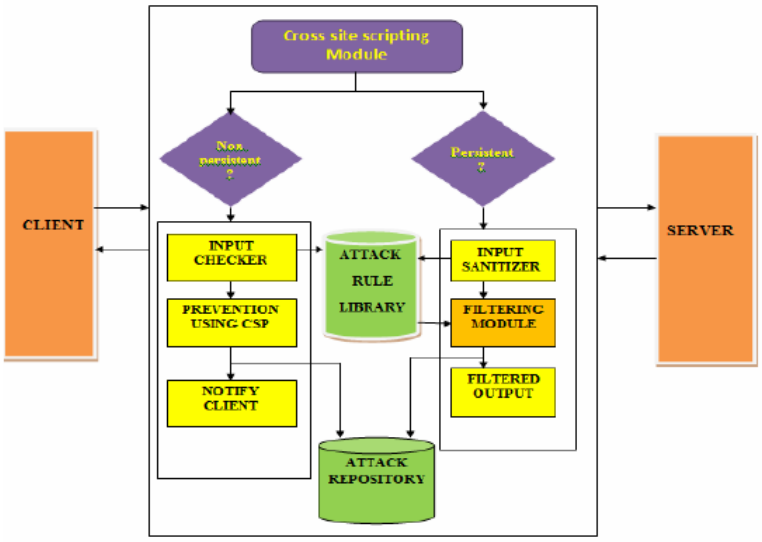

Figure 3. Detection mechanism for XSS attack (Jacob *et al.*, 2016)

Duraisam and Subramaniam (2016) proposed a Concolic testing-based technique for detecting possible vulnerable outputs. The proposed system is a hybrid software confirmation method that accomplishes symbolic implementation. The testing application is designed for only JSP web pages. The approach is able to sense and avert the XSS attack in two levels: (Detection and Prevention Phase). Two well-organised procedures were exploited in the mechanism: Concolic algorithm for detecting the XSS attack and pattern filtering algorithm for preventing the detected XSS attack. Manaa and Hussein (2016) proposed a mixed solution for averting XSS attacks in websites using encoding filtering mechanism to effectively stop both the stored XSS and the reflected XSS attacks respectively via input fields in web pages. ASP.net using C#, SQL server to store information in the database, JavaScript and HTML and PHP were used to actualise the development. The solution was tested in many web sites in both the client and server side.

Singh *et al.* (2015) proposed an all-purpose security mechanism (D-WAV), illustrated in Figure 4, for detecting XSS attack in web application and guarding the server side by straining imaginable JavaScript in order to successfully guard against facts drip from the user's location. Test result suggested that the proposed approach could filter vulnerable JavaScript in Web applications from the server side.

Maheswari and Anita (2015) proposed an expert system dynamic mechanism for detecting client-side XSS attack in a real time environment. The proposed system architecture of the expert system comprises key modules like temporal rule manager, user interface, attack analyser module, decision manager, and data collection manager. The mechanism works by searching for likely loop holes in the web application. It initially discovers the HTML forms in the web page being verified. The matching values are changed with strings that characterise an XSS attack. If the resulting HTML page sets a specific JavaScript value (`document.vulnerable = true`) then the given XSS string is discerned as vulnerable by the mechanism.

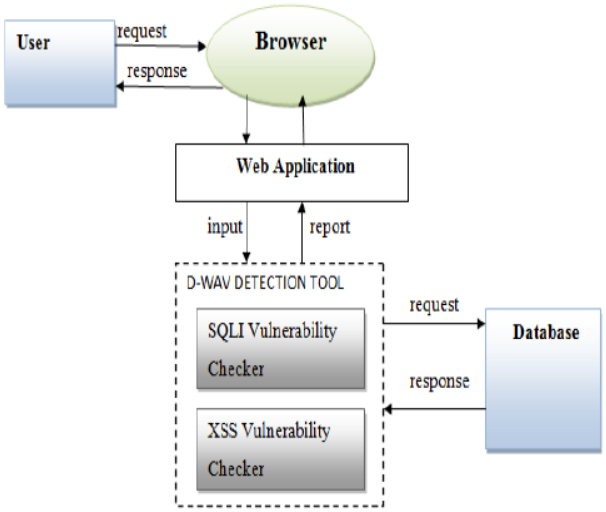

Figure 4. D-WAV system architecture (Singh *et al.*, 2015)

Hydara *et al.* (2015) used genetic algorithm model to detect XSS attack in web applications. The model has two major parts; the first part converts the source codes of the applications to be verified to Control Flow Graphs (CFGs) using the White Box Testing method, in which each node characterise a statement and each edge characterise the movement of data from one node to another. A static analysis tool called PMD was used in this task. The second part fixated on sensing the vulnerabilities in the program codes using an improved Genetic Algorithm (GA). The proposed method was executed and certified on Java based web applications only and it is intelligent enough to sense XSS susceptibilities in the program code before an application is installed.

Anjaneyulu *et al.* (2015) proposed a mechanism for protecting web applications against XSS attack on the client-side. The proposed solution uses a password authentication system and a physical address in encoded form so that the period during which user is active on a specific system can be kept on the server. So each time a cyber-criminal tries to access the active session without authorisation, the server system can sense it out because the physical address of his machine is quite different from the physical address of the user's machine. Figure 5 shows the algorithm used.

```
Start
   -Take password from user and authenticate it.
   -Encrypt MAC address and attach it with the password
   -Store MAC address to the server
   -during active session:
Repeat {
          For every request{
                If(MAC add of device==stored MAC add)
                Entertain request
                      Else
                Terminate connection
                      }
    }till connection terminate or user logged out
```

Figure 5. Algorithm for preventing XSS attack (Anjaneyulu *et al.*, 2015)

Umasankari *et al.* (2013) recommended an active mechanism for eradicating XSS susceptibilities in web based applications. The suggested method consists of two main portions: The XSS weakness discovery and XSS vulnerability elimination. The first portion recognises the prospective XSS exposure in server side applications, while the second portion first finds the code location where the malicious data can be escaped from before it decides on the essential evading mechanisms using the ESAPI's API. The method suggested by Umasankari *et al.* (2013) also offers two choices to the user in order to protect the implicated declaration: (1) Lenient Mode which demands the user to input a proper sanitisation method (2) Strict mode which absolutely takes away data that is not trusted from the code location it is initiated. The first technique embraces the stationary analysis method to trail the movement of user inputs into HTML output statements and recognises the possible susceptible statements, while the second technique uses pattern matching and data dependency analysis to recognise the HTML contexts in which the user inputs are referenced and the vital escaping tools that avert

 I S S N : 2 6 3 6 - 6 0 7 X

program inoculation. Then it executes source code generation and replacement to source hypothetical susceptible statements with proper escaping APIs. Test results suggested that the automated mechanism developed is fruitful in eradicating all the actual XSSVs established in the verified matters.

Mahapatra *et al.* (2012) recommended a server side framework to protect java web based applications from XSS Attack based on pattern matching methodology. The suggested framework consists of modifier and request/response analyser sections. The request analyser selects which request is mischievous or not in order to give its verdict accordingly. The response analyser and modifier section handles the data to be reverted to the client; it alters the malicious response to innocuous data. Attack recorder and response rejecter section registers the malicious request/response for upcoming use. The Java Regex was used for pattern generation and matching of the malicious attack signatures. The pattern matching framework was also tested on some java web based applications. The power of the framework can be utilised on any existing java web based application deprived of program code alteration. The authors recommended application of artificial intelligence (AI) and related areas in detecting new forms of web attack; hence further investigations should be channelled toward that direction.

Shalini & Usha (2011) designed an efficient client side solution to deliver operational security against XSS attack, keeping the notion of usable safety with augmented web browsing. The solution used a three phase procedure and was executed using Mozilla Firefox version 1.5 an open source web browser. The Mozilla Firefox executes JavaScript programs contained within the web pages with the support of the JavaScript engine (SpiderMonkey). The C programming language was used to develop the engine, which is a vital portion of the web browser. The proposed solution was tested with very many malicious inputs, non-defenceless input with white listed tags and defenceless websites. The solution is platform independent as distrusted attacks can be blocked by stopping the inserted script from being delivered to the JavaScript engine instead of executing dangerous changes on the HTML.

Selvamani *et al* (2010) proposed a browser side way out for alleviating XSS attacks in web based applications. The solution is capable of analysing web pages with implanted links i.e. each time a web page is fetched by the client side solution on behalf of the user; it examines the page and mines all external links implanted in that page. The main contribution of the proposed solution is that it efficiently reduces XSS attacks.

3. Performance Analysis and Evaluation

This section evaluates the performance of the detection and prevention techniques for XSS attacks reviewed and also a comparison evaluation of the conditions necessary for XSS solution location or deployment. Table 2 analyses the various XSS detection/prevention techniques reviewed. The table further showed the technique solution side, measure used and support for self-learning ability to efficiently and effectively detect and prevent XSS attacks.

Table 2. Analysis of the XSS detection/prevention techniques reviewed

S/N	Author	Solution location			Measures used			Self-Learning Ability
		Client	Server	Both	Detect	Prevent	Both	
1.	Rao *et al.* (2017)	√	x	x	x	√	x	x
2.	Imhanlahimi *et al.* (2017)	√	x	x	x	x	√	√
3.	Geetha *et al.* (2017)	√	x	x	x	x	√	x
4.	Goswami *et al.* (2017)	x	x	√	x	x	√	√
5.	Jacob *et al.* (2016b)	x	x	√	x	x	√	x
6.	Duraisam and Subramaniam (2016)	√	x	x	x	x	√	x
7.	Manaa and Hussein (2016)	x	x	√	x	x	√	x
8.	Maheswari and Anita (2015)	√	x	x	x	x	√	x
9.	Singh *et al.* (2015)	x	√	x	x	x	√	x
10.	Hydara *et al.* (2015)	√	x	x	√	x	x	x
11.	Anjaneyulu *et al.* (2015)	√	x	x	x	x	√	x
12.	Umasankari *et al.* (2013)	x	x	√	x	x	√	x
13.	Mahapatra *et al.* (2012)	x	√	x	x	x	√	x
14.	Shalini and Usha (2011)	√	x	x	x	x	√	x
15.	Selvamani *et al.* (2010)	√	x	x	x	x	√	x

A graphical representation of the percentage rating of XSS solution side reviewed is depicted in Figure 6. As observed, the client side solution to XSS attacks on web application is rated highest with 60%, followed by both sides (hybrid approach) rated 27%, and then the server-side rated 13%. Figure 7 and Figure 8 show the percentage rating of different measures utilised and the percentage rating of the self-learning ability support respectively. Table 3 and Figure 9 show conditions for XSS Solution Location Deployment and the corresponding percentage rating for the XSS solution location deployment respectively. As observed (Figure 9), the client side solution location to XSS attacks on web application is rated high with 65%, and then the server side approach rated 35%.

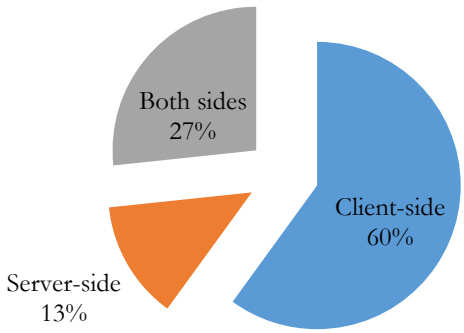

Figure 6. Graphical representation of percentage rating of XSS solution side reviewed

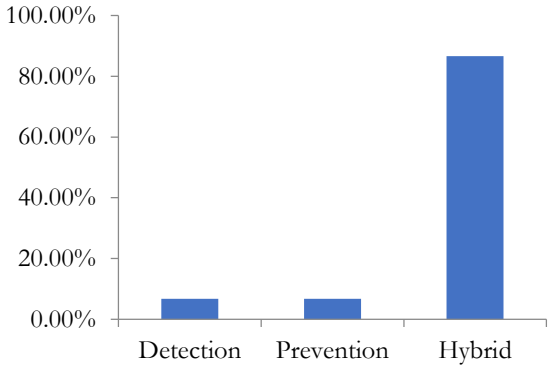

Figure 7. Percentage rating of the different measures used

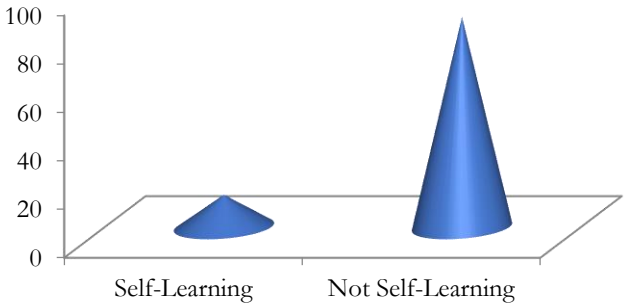

Figure 8. Percentage rating of self-learning ability support

 ISSN: 2636-607X

Table 3. Conditions for XSS solution location deployment

S/N	Criteria	Solution location	
		Client	Server
1.	Mischievous scripts implementation	√	x
2.	Takeover user periods	√	x
3.	Users linking to a mischievous server	√	x
4.	Users entrance to counterfeit URL	√	x
5.	Unprotected secure socket layer encoded networks	x	√
6.	Infected cookies attack	√	x
7.	Form alteration	√	x
8.	Browsers execution limitation	√	x
9.	Implementation of programs on the client machine that makes weak server data noticeable	x	√
10.	Theft of login particulars like username and passwords.	√	x
11.	Failure of web sites to identify character set	√	x
12.	Browsers understanding of information they receive from servers	√	x
13.	Captivating complex information from clients and their website activities	√	x
14.	Theft of business data	x	√
15.	Keystroke recording of clients that officially visit a web site	√	x
16.	Taking down a web site	x	√
17.	Initiating Denial of Service	x	√
18.	Takeover of a site for a ransom	x	√
19.	Redirecting visitors to a wrong web sites	x	√
20.	Launching of data for unlawful acquirement	√	x

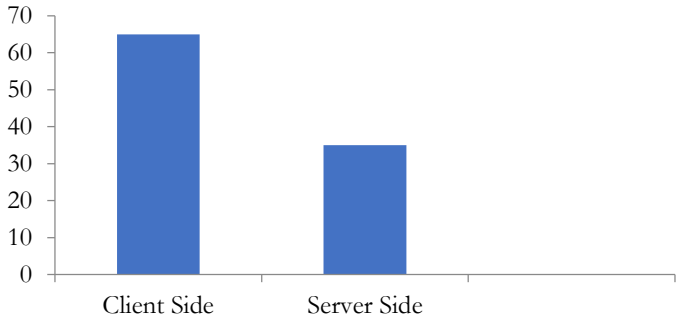

Figure 9. Analysis of conditions for XSS solution location deployment

4. Discussion

This section discusses the findings of this paper. Adhyaru (2016) was of the view that; there are so many confidentiality threats in web applications and this has really led to too many websites being hacked on a daily basis by anonymous personalities. These websites are targeted because of diverse categories of motives as shown in Table 1. Stealing of sensitive information for the dynamic web site is rated the highest (42%), followed by web site defacement (23%) amongst other reasons. This also agrees with the web application security vulnerability chart in Figure 2 with the alarming (26%) rate of XSS attack. CAVTR (2013) shows that the attackers are seriously looking for sensitive information to steal from the users of those sites being hacked with a major aim of defrauding the users or organisations in-charge.

Also shown from analysis carried out on the recent XSS detection and prevention techniques; the result of Figure 6 showed that 60% of the solution to XSS attack is on the client end, which is in agreement with the report of Imhanlahimi *et al.* (2017); while the server-side solution to XSS attack is rated low (13%), which also agrees with the major disadvantage of the server side solution that it does not interrupt HTTP and SSL connections (Rohilla *et al.*, 2016).

The analysis of conditions for XSS solution location deployment (Figure 9) shows more criteria for XSS solution location on the client side with about 65%, as compared to the solution location on the server side that is 35%. Again, this also agrees with the finding in Figure 6 while 60% of XSS detection and prevention techniques on the client side and 13% on the server side as the case may be.

Figure 7 shows the percentage rating of the measure used for XSS attack with the hybrid approach (i.e. having both the detection and prevention mechanism together) is rated highest with 86.66% indicates a defensive match for the alarming increase rate of web application attacks with their terrible motives, while the percentage rating of the self-learning ability support in detecting XSS attacks in Figure 8 indicates that 13.33% of the techniques reviewed supported self-learning ability to easily detect new XSS attacks while 86.67% of the techniques did not support self-learning. This suggests that most of the available techniques may not be able to detect new XSS attack, which is also one of the major reasons why Mahapatra *et al.* (2012) recommended that future investigation on XSS detection and prevention techniques should consider using artificial intelligence (AI), machine learning and neural networks.

5. Conclusion

In this paper, the true concept of web applications and its structure; web applications vulnerability via XSS attacks are presented and discussed. Some latest recognition and avoidance methods of the attacks mentioned are prudently studied and examined. The study shows that most of the available detection and prevention solutions to XSS attacks are on the client-side and they lack self-learning ability to detect new XSS attacks though some few researchers have started pioneering the self-learning ability support in this regards. A lot of improvement can still be done by the research community to effectively handle the web application security for obtaining more efficient results as recommended.

After critically studying the different detection and prevention techniques or models for XSS attacks in web based applications in this research work; the results indicates that only two (2) out of the fifteen (15) XSS techniques reviewed support self-learning mode, which is also analysed and rated 13.33%. This percentage is very poor as new attacks on web applications are being released on a daily basis, and this requires techniques that can easily learn and detect new XSS attacks on their own.

This paper recommends that future investigation on XSS detection and prevention techniques should consider using artificial intelligence (AI), machine learning and neural networks to design and implement self-learning detection and prevention systems.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

Adekunle, O. O. (2014). *Web server technology*, National Open University of Nigeria.

Adhyaru, R. P. (2016). Techniques for attacking web application security. *International Journal of Information Sciences and Techniques*, *6*(1): 45 – 52.

Anjaneyulu, G. S. G. N., Singh, S., & Kumari, B. (2015). Preventing cross-site scripting attacks on the client-side. *Archives of Applied Science Research*, *7*(2): 9 - 14.

Cenzic Application Vulnerability Trends Report (CAVTR), (2013). Application vulnerability trends report. Retrieved from: https://info.cenzic.com/rs/cenzic/images/CenzicApplication-Vulnerability-Trends-Report-2013.pdf

Duraisamy, A., & Subramaniam, M. (2016). Detection and prevention of cross site scripting attacks using concolic testing and patterning approach in web applications. *Australian Journal of Basic and Applied Sciences*, *10*(18): 66 – 71.

Geetha, T., Karthikeyan, R., Jeevitha, B., & Kamala, V. (2017). A client side solution for mitigating cross site scripting attacks. *International Journal of Engineering Science and Computing*, *7*(6): 13063 – 13067

Goswami, S., Hoque, N., Bhattacharyya, D. K., & Kalita, J. (2017). An unsupervised method for detection of XSS attack. *International Journal of Network Security*, *19*(5): 761 – 775.

Hydara, I., Sultan, A. B., Zulzalil, H., & Admodisastro, N. (2015). Cross-site scripting detection based on an enhanced genetic algorithm. *Indian Journal of Science and Technology*, *8*(30): 1 – 7.

Imhanlahimi, R. E., John-Otumu, M. A., & Imhanlahimi, G. O. (2017). Identification and prevention of cross site scripting attacks on web applications using soft computing approach. *1st InfoSec Conference of Information Technology Systems and Security Professional*, *1*(1): 1 – 9.

Jacob, L., Chatterjeer, M., & Nadar V. M. (2016b). Detection model for XSS attack, *International Journal of Computer Science and Information Technology & Security*, *6*(3): 64 – 67.

Jacob, L., Nadar, V. M., & Chatterjee, M. (2016a). Web application security: a survey, *International Journal of Computer Science and Information Technologies*, *7*(1): 441 – 445.

John-Otumu, A. M., Okonigene, R. E., and Imhanlahimi, R. E. (2015). Architecture and development of an automated workflow system for employees' savings & loan scheme in Nigerian universities. *Quest Journals Journal of Software Engineering and Simulation*, *2*(11): 6 – 16.

Kamal, P., Nagpal, B., & Murari, R. (2015). Overview of detection methods of Cross site scripting attack. *International Journal of Advanced Research in Computer Engineering & Technology, 4*(6): 2942 – 2945.

Mahapatra, R.P., Saini, R., & Saini, N. (2012). A pattern based approach to secure web applications from XSS attacks. *International Journal of Computer Technology and Electronics Engineering*, *2*(3): 196 – 203.

Maheswari, K. G., & Anita, R. (2015). A dynamic tool for detection of XSS attacks in a real-time environment. *ARPN Journal of Engineering and Applied Sciences*, *10*(10): 4627 – 4634.

Manaa, M. E., & Hussein, R. (2016). Preventing cross site scripting attacks in websites, *Asian Journal of Information Technology*, *15*(6): 2797 – 2804.

Nithya, V., Pandian, S. L., & Malarvizhi, C. (2015). A survey on detection and prevention of cross-site scripting attack. *International Journal of Security and Its Applications*, *9*(3): 139-152.

Open Web Application Security Project (OWASP), (2013). Retrieved from: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Raghuvanshi, K. K., and Tiwari, N. (2014). Prevention and detection techniques for XSS injection attacks. *International Journal of Computer Trends and Technology*, *12*(2): 104 – 106.

Rao, R. K., Ram, S., Kumar, M. A., Supritha, R., & Reza, S. A. (2017). Cross site scripting attacks and preventive measures. *International Research Journal of Engineering and Technology*, *4*(2): 2016 – 2019.

Rohilla, M., Kumar, R., & Gopal, G. (2016). XSS attacks: analysis, prevention and detection. *International Journal of Advanced Research in Computer Science and Software Engineering*, *6*(6): 264 – 271.

Selvamani, K., Duraisamy, A., & Kannan, A. (2010). Protection of web applications from cross-site scripting attacks in browser side. *International Journal of Computer Science and Information Security*, *7*(3): 229 – 236.

Shalini, S., & Usha, S. (2011). Prevention of cross-site scripting attacks on web applications in the client side. *International Journal of Computer Science Issues*, *8*(4): 650 – 654.

Singh, P., Thevar, K., Shetty, P., & Shaikh, B. (2015). Detection of SQL injection and XSS vulnerability in web applications. *International Journal of Engineering and Applied Sciences*, *2*(3): 16 – 21.

Umasankari, P., Uma, E., & Kannan, A. (2013). Dynamic removal of cross site scripting vulnerabilities in web application. *International Journal of Advanced Computational Engineering and Networking*, *1*(4): 37 – 40.