ISSN:2320-0790

An International Journal of Advanced Computer Technology

XSS Defense: An Approach for Detecting and Preventing Cross Site Scripting Attacks

Neha Gupta
Assistant Professor, CSE Dept
Surya World, Tehsil Rajpura

Abstract: Web Applications provide wide range of services to its users in an easy and efficient manner. From the past few years web based attacks are increasing. Cross Site Scripting (XSS) is one of the major attacks found in web applications. In 2013, OWASP (Open Web Application Security Project) has ranked XSS third in the list of top 10 attacks found in web applications [11]. XSS attacks occur when an application takes insecure data and sends it to the browser without proper validation or escaping. This can result in hijacking of user sessions, defacing websites and redirecting the users to malicious sites. This paper presents a new XSS defense approach which is based on the OWASP guidelines available for prevention of XSS attacks. In this approach for XSS defense there is an XSS checker that will check for the unauthorized characters in each parameter in the input and block them on both client side and server side of a web application. Client side solutions reduces the run time overhead and server side solutions are more reliable as any attack occurring when request is going from client to server will be detected by server side solution only but it incurs runtime overhead. So a combination of both will be more robust as it can prevent most of the attacks and manage runtime overhead effectively. This approach is tested on a prototype. It is found that this approach covers major categories of XSS attacks i.e. reflected and stored and will require no additional frameworks.

Keywords: Cross Site Scripting, Web Application Security, Web Application Attacks.

I. INTRODUCTION

Web Applications have become one of the most important ways to provide a broad range of services to users. In the recent years, web-based attacks have caused harm to the users of web applications. Most of these attacks occur through the exploitation of security vulnerabilities in the web-based programs. So, the mitigation of these attacks is very crucial to reduce its harmful consequence. The main issue is that if malicious content can be introduced into a dynamic web page, neither the web site nor the client is capable of recognizing that anything like this happened and prevent it.

II. CROSS SITE SCRIPTING

XSS (cross site scripting) flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. Cross Site Scripting allows an attacker to embed malicious scripts into a dynamic web page which can be vulnerable and can result in hijacking of user sessions, defacing web sites, or redirecting the user to malicious sites. A high level view of typical XSS attack is as shown in fig. 1[13]. Depending on the ways HTML pages reference user inputs, XSS attacks can be classified as reflected, stored, or DOM-based [12].

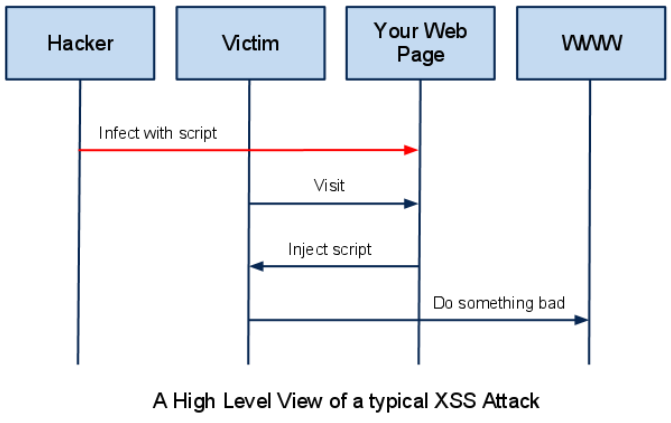


Figure 1: XSS Attack

A. *Reflected or Non-persistent XSS*

These holes are present in a Web application server program where it references accessed user input in the outgoing webpage. This type of XSS exploit is common in error messages and search results. The malicious content does not get stored in server. Sever bounces the original input to the victim.

B. Stored or Persistent XSS

These holes exist when a server program stores user input containing injected code in a persistent data store such as a database. Attacks on social networking sites commonly exploit this type of XSS flaw. Server stores the malicious content and serves the content in original form.

C*. DOM Based XSS*

In contrast, This is an XSS attack wherein the attack payload is executed as a result of modifying the DOM environment in the victim's browser used by the original client side script, so that the client side code runs in an unexpected manner. That is, the page itself does not change, but the client side code in the page executes differently due modifications that have occurred in the DOM environment.

III. RELATED WORK

Over the past few years, there has been lot of research going on in both institutes as well as industries to prevent XSS attacks. Researchers have proposed some detection and prevention mechanisms discussed below:

[1]T.Jim, N.Swamy and M.Hicks developed a mechanism that modifies the browser so that it can execute only legitimate scripts. In this the website embeds a security policy in its pages that specifies allowed scripts to run and browser enforce these policies. This mechanism requires minimal effort and low performance overhead. However, it requires installation of additional frameworks.

[2]Siddharth Tiwari, Richa Bansal and Divya Bansal developed a client side solution for cross site scripting which is a three step process i.e. script detector, analyzer, and data monitoring system. Every HTTP request will be passed to the script detector which checks for the maximum number of characters. Analyzer checks for the special characters in the request .If special characters exist it will be passed to the parser else request is processed. Data monitoring system monitors the flow of data. Though it is platform independent, but it degrades the performance of client system.

[3]M.T. Louw and V.N. Venkatakrishnan developed a tool that works on existing browsers. To accomplish this a parse tree is generated at server of the application with precautions that ensure that there is no dynamic content and the generated parse tree is then conveyed to document generator of the browser on the client browser without taking vulnerable paths. It requires code instrumentation and installation of additional framework.

[4]E.Kirda et al developed Noxes which acts like a personal firewall that either allows or blocks connections to websites based on the filter rules, which are user-specified URL white lists and blacklists. It provides an additional layer of protection. Noxes alerts the client and asks the client to permit or deny the connection, and remembers the client's action for future reference. This approach covers all type of XSS attacks and clients don't have to rely on the web application for security. However, it requires client actions it does not detect exploits that involve Web content manipulation.

[5]Hossain Shahriar and Mohammad Zulkernine developed MUTEC in which we apply the idea of mutation based testing technique to generate adequate data sets for testing the XSS vulnerabilities. A test case kills mutant if it causes different output between original program and the mutant.. This technique helps in discovering the vulnerabilities before the actual deployment. However, it requires intensive labor and the effectiveness of testing based techniques depends entirely on the correctness of specification.

[6]P.wurzinger, C.Platzer, C.ludl, E.kirda and C.Kruegel developed a server side solution that detects and prevents cross site scripting attacks. SWAP includes a reverse proxy that intercepts all HTML responses and a modified browser which detects the script content. This approach requires only simple automated changes of original web application.. However, there is performance overhead and it is capable of detecting only JavaScript based attacks.

[7]Sid Stamm, Brandon Sterne and Gervase Markhan developed an approach that has content restrictions and content security policy. Content restrictions allow designers to specify content interaction on their websites. Content security policy specifies from where resources may be requested and the type of resources that may b loaded. However there is no single policy for all the documents.

[8]Rattipong Putthacharoen and Pratheep Bunyatnoparat developed a technique that is implemented in web proxy where cookies that are passed between user and web application are rewritten automatically. Basically, the name attribute in cookie will be rewritten automatically by a randomized value before it is sent to the browser database. Cookie that will be returned by the browser will also be rewritten back to original value at web proxy before being forwarded to web server thus preventing cookie stealing. Drawbacks are compatibility issues and performance overhead

[9]Hossain Shahriar and Mohammad Zulkernine developed a server side approach which is based on boundary injection and policy generation notation. In this approach we pre and posted each dynamic content generation with a boundary which is a HTML or JavaScript content. Token is also inserted in each pair of boundary which is used to uniquely identify content generation. Pair of boundary contains information on expected content features. It protects programs that suffer incorrect input filtering. However, this approach incurs runtime overhead and also requires user-defined security policies.

[10]Takeshi Matsuda, Daiki Koizumi and Michio Sonoda developed a detection algorithm against cross site scripting attacks by extracting an attack feature of cross site scripting attacks and then considering the appearance position and frequency of symbols. It focuses attention on characters which are included in XSS attacks. However, it requires the learning of detection threshold.

IV. XSS DEFENSE DETECTION AND PREVENTION FRAMEWORK

To handle XSS attacks, XSS defense is proposed as shown in fig. 2 which adds XSS checker to check for unauthorized characters in input on both client side and server side. XSS checker will check for allowed characters and block the unauthorized characters. XSS checker detection and prevention framework is as shown in fig. 2. This approach is based on the OWASP guidelines for preventing XSS[14]. The steps used in the process of detection and prevention of XSS attacks by XSS checker is as follows:

Step 1: Get parameters from the request.

Step 2: If parameter exists check if there is any unauthorized character in parameter value.

Step 3: If there is any unauthorized character show an error message and then move to the next parameter. If there are no unauthorized characters simply move to the next parameter.

Step 4: If there are no other parameters in the request it will check other content for XSS violation.

Step 5: If there is any XSS violation in other content an error message will be shown otherwise it will move to the servlet layer.

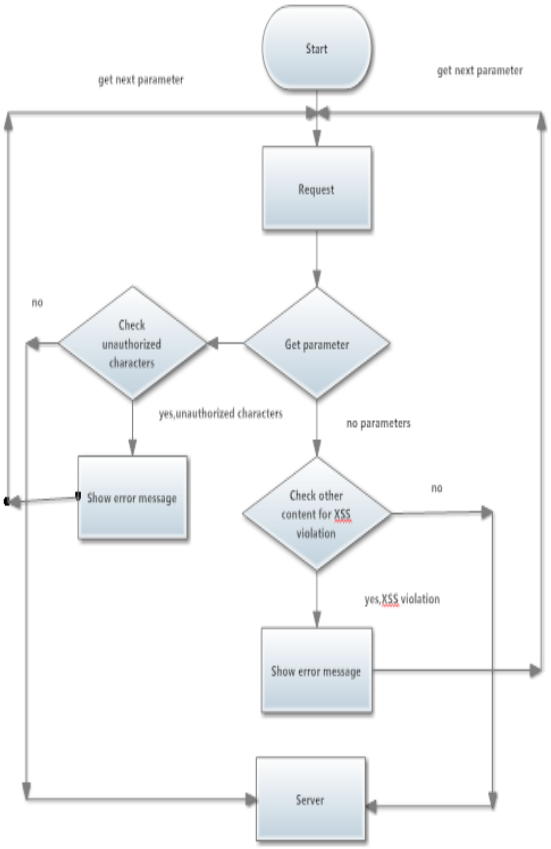


Figure 2: XSS Detection and Prevention Framework

V. IMPLEMENTATION AND EVALUATION

The proposed work is implemented on a prototype client server java application. First the attacks are generated on the vulnerable client server web application before adding the XSS checker to look for the vulnerabilities in the web application. Major type of XSS attacks i.e. reflected and stored are performed and the application was found vulnerable to all the types of XSS attacks.DOM based attack cannot be covered as there is no generalized solution for it. Also, the parameters could be modified when request goes from client to server. For performing such attacks BURP tool[15] is used that acts

as proxy between the client and server which can intercept the request and modify it which can result in an attack. Attacks performed are as shown in fig. 3, fig. 4 and fig. 5.

In fig. 3 stored XSS attack is performed on the prototype client server application in JAVA. Firstly the first name is saved as JavaScript code and later when the record is searched using phone number the record is found and the script in the first name got executed. In fig. 4 reflected XSS attack is performed on the prototype. A record is searched with last name which is basically a script. As a result the script got executed and no such record was found. In fig. 5 an attack is performed in which the parameter of request gets modified when request goes from client to server. First name was modified from shilpi to a script and the user got saved as a script which results in execution of the script.This is done using a BURP tool which acts as proxy and intercepts the request that goes from client to server.

After that the proposed approach of XSS Defense is applied to the prototype web application by adding the XSS checker on client side as well as server side. All the input fields are divided into categories based on the type of characters it blocks i.e

- Category A: This type blocks <,>,",',& and !.In the prototype application it is used for email and phone number.
- Category B: This blocks <,> and !.In the prototype application it is used for first name, middle name and last name.
- Category C: This doesn't block any character. In the prototype application we have no such field.

After that a checker function is included in the file both at client side and server side. This function checks for unauthorized characters based on the category specified.

In Fig. 6 when stored XSS attack is performed the defense mechanism prevents it by Client Side Defense. A script is entered in in first name field but rather than saving it the defense mechanism prevents it and shows an error message. In Fig. 7 when reflected XSS attack is performed the defense mechanism prevents it by the client side Defense. A script is searched which is expected to be executed. But the Defense mechanism does not allow it to be executed and shows an error message.

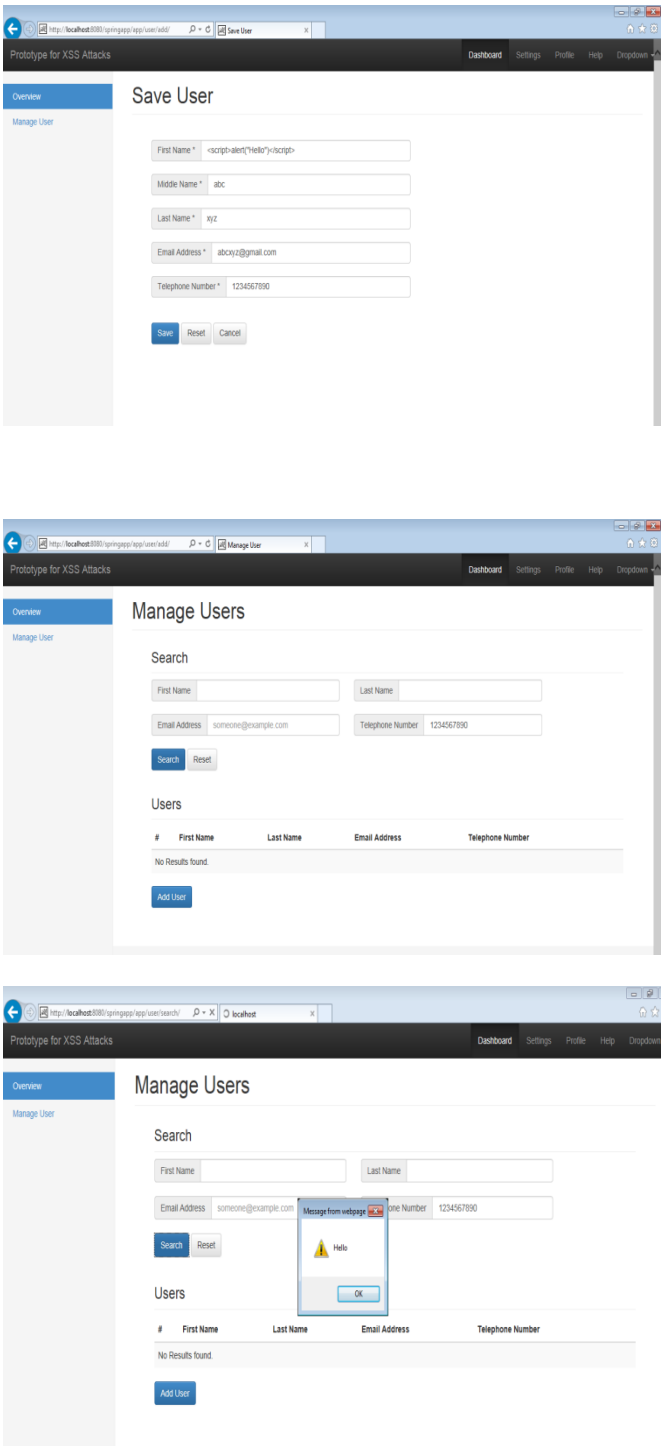


Figure 3: Stored XSS Attack on Prototype

In Fig. 7 an attack was performed in between when the request is going from client to server. Here client sends safe data but someone modifies it in middle and changes the middle name with a script. This attack is prevented using server side defense which shows an error message and does not allow malicious code to be saved as shown in Fig. 8.

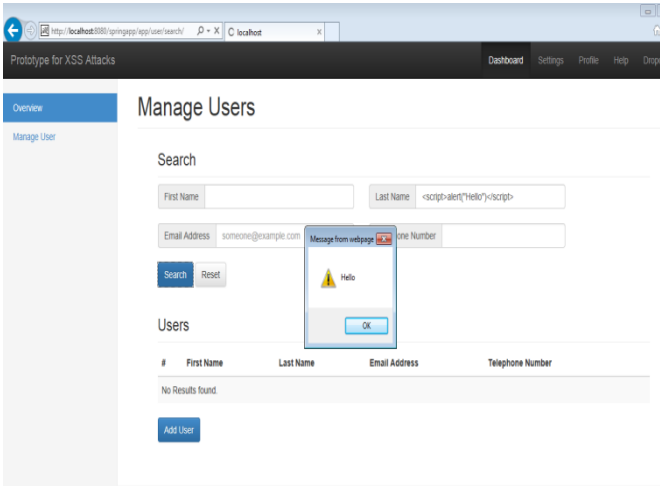
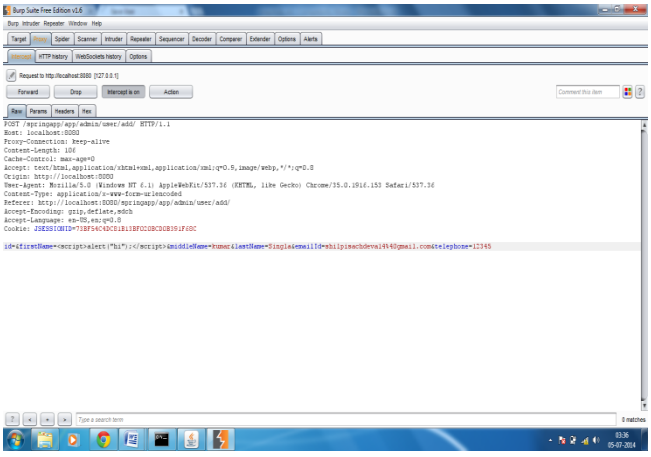


Figure 4: Reflected XSS attack on Prototype



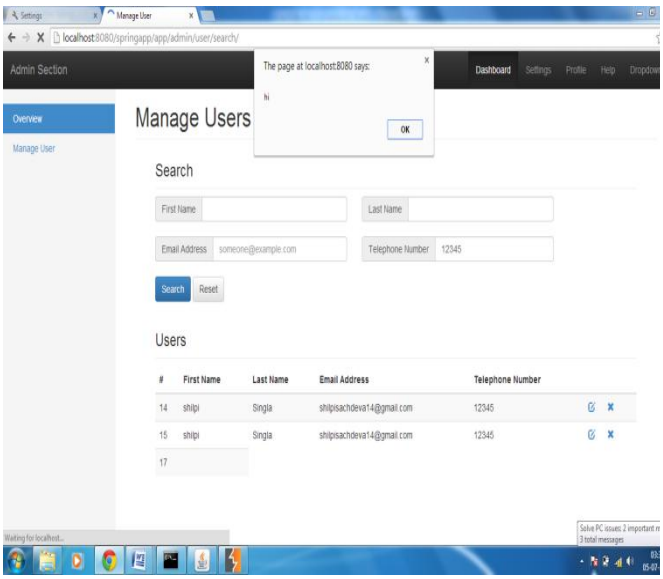


Fig 5: Attack using BURP

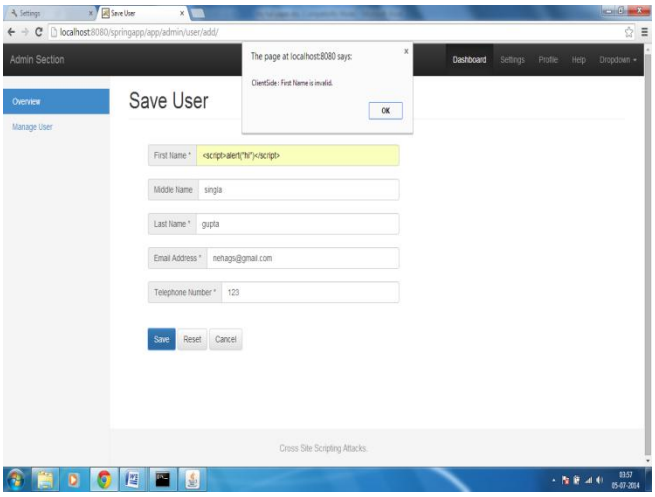


Figure6: Client Side Defense for Stored XSS Attack

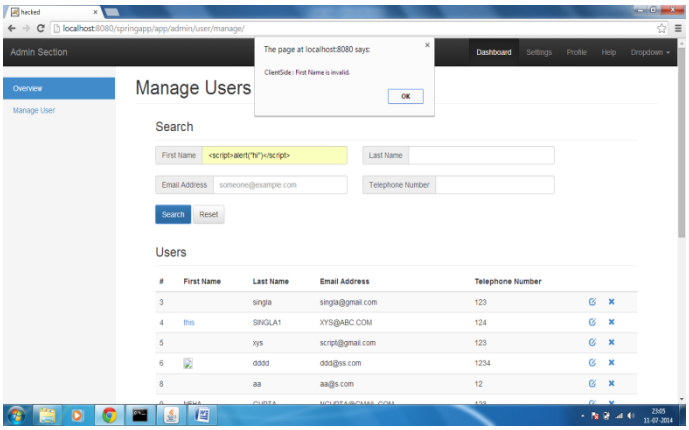


Figure 7: Client Side Defense for Reflected XSS Attack

VI. RESULTS

We have performed Chi square Hypothesis testing to prove the importance of defense approach used shown in Fig 9. Hypothesis testing is test for accepting and rejecting the assumption about the population. Population here refers to the kind of data over which test is applied. Chi Square ($\chi 2$) is a non parametrical test to find the association or dependency between the classified variables. Chi square test is divided into three categories for testing [16] which are Chi Square test for Goodness of Fit, Chi Square test for Homogeneity and Chi Square test for Independence. We have used Chi Square test for independence which is applied when we have two categorical data form single population and we want to test dependency between variables. Here we

have used this for finding

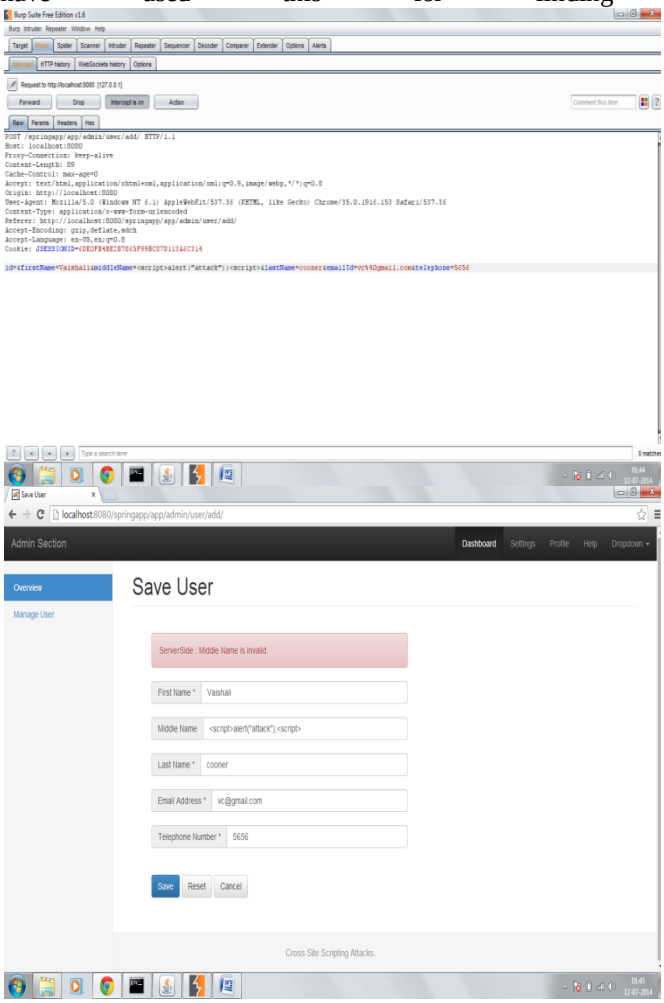


Figure 8: Server Side Defense for XSS attacks

dependency between number of users and number of attacks and also for need of defense approach.

Test1: Testing Dependency for number of users and number of attacks.

Step1: State of Hypothesis

H0=the number of users have no effect over number of attacks over a web application.

H1=the number of users affect number of attacks over a web application.

H0 is assumed as null hypothesis. Number of users and number of attacks are variables.

Step2: Significance Level

The significance level we have chosen is 0.001 which states that if H0 is accepted than it has 0.001% probability likely to be dependent. If H0 is rejected than H1 has 99.99% likely to be dependent on variables. We have used contingency table of 2x2. Where degree of freedom df is (r-1)(c-1) where r stands for number of rows and c for number of columns. Here df is resulted

Type of approach	Runtime overhead	Attacks
Client Side	Negligible	All client side attacks covered, attacks occurring when request goes from client to server not covered.
Server Side	Considerable	All client side attacks covered, attacks occuring request goes from client to server also covered.
Defense Approach(Combined client and server side)	Intermediate	All client side attacks covered, attacks occurring when request goes from client to server also covered.

TABLE I: Evaluation of type of techniques

The critical value for significance level 0.001 with df 1 is 10.83 and chi square value is 115 as shown in Table II. Chi square significance value can be checked from given table[17].A value of chi square equal or greater 10.38 would be expected to occur only once in a thousand times if the null hypothesis is true i.e. there are very less chances of this occurring. Hence our chi square test rejects the null

hypothesis and states that number of users affect number of attacks in a when application.

Test2: Testing Need of Defense Approach

Step1: State of Hypothesis

H0=Defense Approach is not needed for web application attacks.

H1=Defense approach is needed for web application attacks.

	Observed Value(O)	Expected Value(E)	(O-E)	(O-E)2	X2
No. of Users	30	60	-30	900	15
No. of Attacks	200	100	100	10000	100
Chi Square Value					115

TABLE II: Chi Square Dataset

Step2: Significance Level

The significance level we have chosen is 0.001 which states that if H0 is accepted than it has 0.001% probability likely to be dependent. If H0 is rejected than H1 has 99.99% likely to be dependent on variables. We have used contingency table of 4x2. Where degree of freedom df is (r-1)(c-1) resulted as 3.

The critical value for significance level 0.001 with df 3 is 16.27 and chi square value is 209 as shown in Table III.Hence our chi square test rejects the null hypothesis and states that defense approach is needed for prevention of attacks and this is proven to be 99.99% true.

	Observed Value(O)	Expected Value(E)	(O-E)	(O-E)2	X2
No. of Users	30	60	-30	900	15
No. of Attacks	200	100	100	10000	100
No. of Users with Defense approach	60	30	30	900	30
No. of Attacks with Defense approach	20	100	-80	6400	64
Chi Square Value					209

Table III: Chi Square Dataset

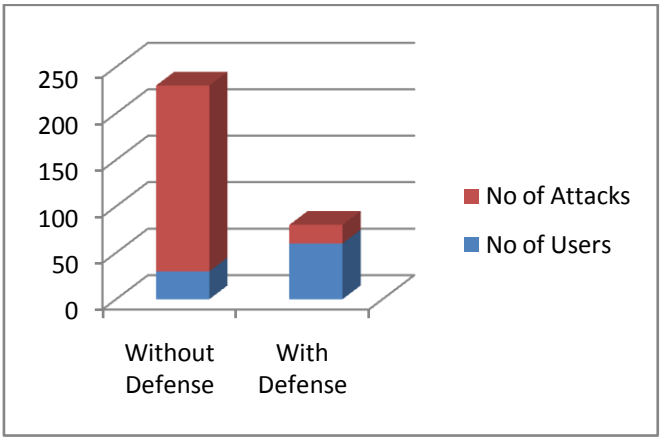


Figure 9: Result Comparison using chi square test

We also compared the results based on page load times with no approach used and when SWAP approach is used. It is found that with no approach page load time is 53.34 ms, with SWAP page load time is 200.50ms and with our approach page load time is 53.94ms for a 10kb page as shown in Table IV and Fig. 10

Approach	Page Load time(ms)
No approach	53.34
SWAP	200.50
Defense Apprach	53.94

Table IV: Comparison of Page Load Times

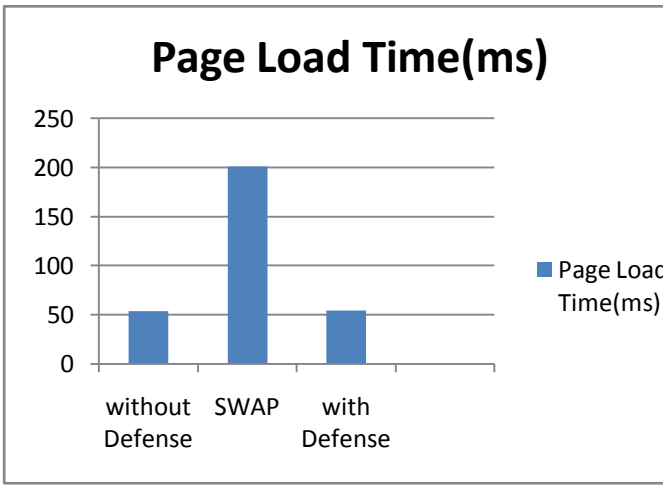


Figure 10: Result comparison based on Page Load Time

VII. Conclusion

As web application attacks are increasing at a very high rate there detection and prevention is a major issue. Cross site scripting is one of the most common attacks found in web applications. We proposed a combination of client side and server side solution which detects and prevents cross site scripting attacks based on the OWASP prevention guidelines. For this XSS checker function is added on both client and server. If an attack is detected at client side only it will not be forwarded to server thus saving runtime overhead which was not possible with server side solution and attacks occurring when request is forwarded from client to server will also be detected and prevented which was not possible with client side solution. We have also performed Chi Square Test to analyze the need for Defense approach which proves to be true. Also, we also checked for page load times which is considerably less when compared with an existing approach.

VIII. References

The heading of the References section must not be numbered. All reference items must be in 8 pt font. Please use Regular and Italic styles to distinguish different fields as shown in the References section. Number the reference items consecutively in square brackets (e.g. [1]).

[1] T.Jim , N.Swamy and M.Hicks, " Defending against Cross-Site Scripting Attacks with Browser-Enforced Embedded Policies,"Proc of the WWW,Banff,Alberta,May 2007,pp. 601-610.

[2] Siddharth Tiwari, Richa Bansal, Divya Bansal, "Optimized Client Side Solution for Cross Site Scripting," IEEE 16th International Conference on Networks, December 2008, pp.1-4.

[3] M.T. Louw and V.N. Venkatakrishnan, "Blueprint: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers," Proc. 30th IEEE Symp. Security and Privacy (SP 09), IEEE CS, 2009, pp. 331-346.

[4] E. Kirda et al., "Client-Side Cross-Site Scripting Protection," Computers & Security,"Proc of 21st ACM Symposium on Applied Computing,Oct. 2009, pp. 592-604.

[5] H. Shahriar and M. Zulkernine, "MUTEC: Mutation-Based Testing of Cross Site Scripting," Proc. 5th Int'l Workshop Software Eng. for Secure Systems (SESS 09), IEEE, 2009, pp. 47-53.

[6] P.wurzinger,C.Platzer,C.ludl,E.kirda and C.Kruegel, "SWAP:Mitigating XSS Attacks using Reverse Proxy, "Proc. Of the SESS,Vancouver,Msy 2009,pp. 33-39.

[7] S.Stamm, B.Sterne and G.Markham, "Reining in the Web with Content Security Policy," Proc. of WWW, Releigh, North Carolina, April 2010, pp. 921-930.

[8] R.Putthacharoen and P.Bunyatnoparat," Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewritng Technique,"Proc. of IEEE 13th International Conference on Advanced Communication Technology, Feb 2011,pp. 1090-1094.

[9] Hossain Shahriar and Mohammad Zulkernine, "S2XS2: A Server Side Approach to Automatically Detect XSS Attacks ,"IEEE Ninth International Conference on Dependable,Automatic and secure computing,2011.

[10] Takeshi Matsuda , Daiki Koizumi and Michio Sonoda, "Cross Site Scripting Attacks Detection Algorithm Based on the Appearance Position of Characters"The 5th International Conference on Communications,Computers and Applications.Istanbul,Turkey,October 2012,pp.-65-70.

[11] Open Web Application Security Project,Top 10 ,https://www.owasp.org/index.php/Top_10_2013-Top_10

[12] Cross site scripting Wikipedia, http://en.wikipedia.org/wiki/Cross-site_scripting

[13] Cross site scripting ,accunetix,http://www.acunetix.com/websitesecurity/cross-site-scripting/

[14] XSS PREVENTION RULES by OWASP, https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

[15] BURP Suite http://portswigger.net/burp/

[16] http://stattrek.com/chi-square-test/independence.aspx

[17] http://www.unc.edu~farkouh/usefull/chi.html