ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

A Pattern Based Approach to Secure Web Applications from XSS Attacks

Dr R.P Mahapatra, Ruchika Saini, Neha Saini

Abstract— **As web applications must be available 24/7 and offer data access to customers, employees, suppliers, and others, they are frequently the weak link in enterprise security. So web applications are popular attack targets due to the lack of coordination and lack of security awareness on part of the developers. When hackers gain access to web applications, they often have direct access to confidential back-end data on customers and the company. Cross Site Scripting Attack belongs to top ten web application vulnerabilities. This paper proposes a mechanism to secure java web applications from XSS (Cross Site Scripting Attack) by applying a framework based on pattern matching approach. The proposed framework consist Request/Response Analyser and Modifier modules. The first interaction of request is to Request Analyser/Modifier Module which decides about request is malicious or not and takes decision accordingly. Response analyser and Modifier module deals with the data to be returned the client, it modifies the malicious response to harmless data. Attack Recorder and Response Rejecter Module records the malicious Request/Response for future use. Java Regex has been used for pattern generation and matching the malicious attack signatures. Some comparisons have been made between existing solutions and the proposed framework. We tested the pattern matching based framework on some java web applications. The strength of the framework is that it can be applied on any existing java web application without source code modification.**

Index Terms-**Pattern, Regex, Request, Response, XSS**

I. INTRODUCTION

Pages of dynamic Web Applications often contain user-supplied parts, when insufficiently filtered, malicious scripts can be injected along with these parts, assuming, that these are executed in a user's browser, it is possible to exploit the trust relationship between user and web server for instance, compromising authentication credentials. This type of attack is called Cross Site Scripting (XSS) Attack.

A. Types of XSS Attacks: There are three types of known XSS flaws [1],[2] and 3] Stored XSS, Reflected XSS, and DOM based XSS, among which Stored XSS vulnerability always allows the most powerful attacks. During this type of attack, attack vectors are submitted to web server and stored on the server (in a database, file system or other locations). When other users request this data and attack vectors are displayed on their browser without sufficient checks, an attack may happen. Hackers use XSS vulnerability to gain access to victims' browser for identification e-mail or password. Because the malicious script running under the context of current user, firewall, encryption method or IDS (intrusion detection system) are ineffective in preventing this type of attack. As described in OWASP TOP10 security risks 2010 [4], XSS is one of the most critical risks. Attackers often perform XSS exploitation by crafting malicious URLs and tricking users to clicking them. These links cause client side scripting languages (VBScript, JavaScript, etc.) of the attacker's choice to execute on the victim's browser. There are numerous ways to inject JavaScript (any script) code into URLs for the purpose of a XSS attack [5], [6], [7], [8] and [9].

B. Injection Points: Where can our web applications fall victim these are called injection points [10]. Since XSS works as an interaction with active server content, any form of input should be filtered if it is ever to show up in an html page. The default example, and the easiest to exploit, is parameters passed in through query string arguments that get written directly to page. These are enticingly easy because all of the information can be provided directly in a clickable link and does not require any other html to perform. Sites that are particularly vulnerable to this form of attack would include guest books, html chat rooms, message boards, discussion forms etc. Some servers include special "404 Page Not Found" or servlet error messages that detail the page that was requested, or parameters passed in. If these elements are not filtered they provide a perfectly overlooked breeding ground.

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

II. RELATED WORK

The easiest and the most effective client-side solution to the XSS problem for users are to deactivate Java Script in their browsers. Unfortunately, this solution is often not feasible because a large number of web sites use JavaScript for navigation and enhanced presentation of information. *Noxes*, a tool, [5] is a client-side web-proxy that relays all Web traffic and serves as an application-level firewall. The approach works without attack-specific signatures. Noxes works as a personal firewall which allows or block connections to websites based on filter rules. Filter rules are basically the white list and blacklist of URL's specified by the user. Whenever a browser sends a HTTP request to an unknown website not listed in filter rules Noxes immediately shows a connection alert to client who can then choose to permit or deny the connection and it remembers the client's action for future use. Noxes requires user-specific configuration (firewall rules), as well as user interaction when a suspicious event occurs. This is a drawback of this tool. Another client-side approach is presented in [11], which aims to identify information leakage using tainting of input data in the browser.

All client-side solutions share one drawback: The necessity to install updates or additional components on each user's workstation. While this might be a realistic precondition for skilled, security-aware computer users, it is perceived as an obstacle or is not even considered by the vast majority of users. Thus, the level of protection such a system can offer is severely limited in practice.

Server Side Solutions makes valuable contribution in the field as XSS-Guard [8] transform the server programs such that they produce a shadow page for real response page. The key idea in the approach is to learn the intention of the web application while creating the HTTP response page. This is done through shadow pages, which are generated every time a HTTP response page is generated. These pages are similar to the real HTTP responses returned by the web application with mainly one crucial difference: they only retain the (authorized) scripts that were intended by the web application to be included, and do not contain any injected scripts. Given the real and shadow pages, one can compare the script contents present in the real page with web-application intended contents, present in the shadow page. Any "difference" detected here indicates a deviation fromthe web application's intentions, and therefore signals an attack.

A Multi-Agent System [12] has been explored for the automated scanning of websites to detect the presence of XSS vulnerabilities exploitable by a stored XSS attack. It works by finding the input points of the application susceptible of being vulnerable to a stored–XSS attack then Injecting selected attack vectors at the previously detected points. Finally it checks the web application for the injected scripts in order to verify the success of the attack. It is not capable runtime detection and prevention of attack also it can be used for attack detection only, i.e. no mechanism for prevention.

Other Server Side solution also has some draw back as in E-Guard algorithm [13] approach there is no mechanism to handle scripts which are stored in grey list; these are left for future analysis. So this algorithm does not provide a satisfactory or can say a complete prevention from XSS attack. This is a passive method which does not provide dynamic detection and prevention of XSS attack. Also these solutions do not provide a proper framework; some of them have incomplete implementation.

III. PROPOSED APPROACH

We proposed a framework (Figure- 1) which used pattern based approach for XSS Attack prevention. We summarized the pattern matching stepwise algorithm. Framework includes Request/Response Analyser, Request/Response Modifier and Attack Recorder Module. In Request/Response Analyser module request and response are analysed for the attack. If attack is found then the request/response are directed to Request/Response modifier module which changes the malicious attack signatures to harmless code. Also if request comes with some unknown attack signature, crafted rules R1, R2, R3, and R4 applied on the input data. These rules help to identify new generated attacks.

Also these patterns can be recorded by Attack Recorder Module in ADB (Attack Database) for future use so that these can be added to Attack Vector Repository. Initially attack vectors (Code) for testing have been collected from ha.cker.org [14]

A. Examples for XSS Attack Vector codes-

1. Using body tag
<BODY onload!#$%&()*~+-_.,:;?@[/|\]^`=alert("XSS")>

2. Double open angle brackets
<iframe src=http://ha.ckers.org/scriptlet.html <

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

<IFRAME SRC="javascript:alert('XSS');"></IFRAME>

3. DIV background-image with unicoded XSS exploit
<DIV STYLE="background-image:\0075\0072\006C\0028'\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">

4. Attack by Hex encoding without semicolons:

<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>

5. List-style-image-
<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS

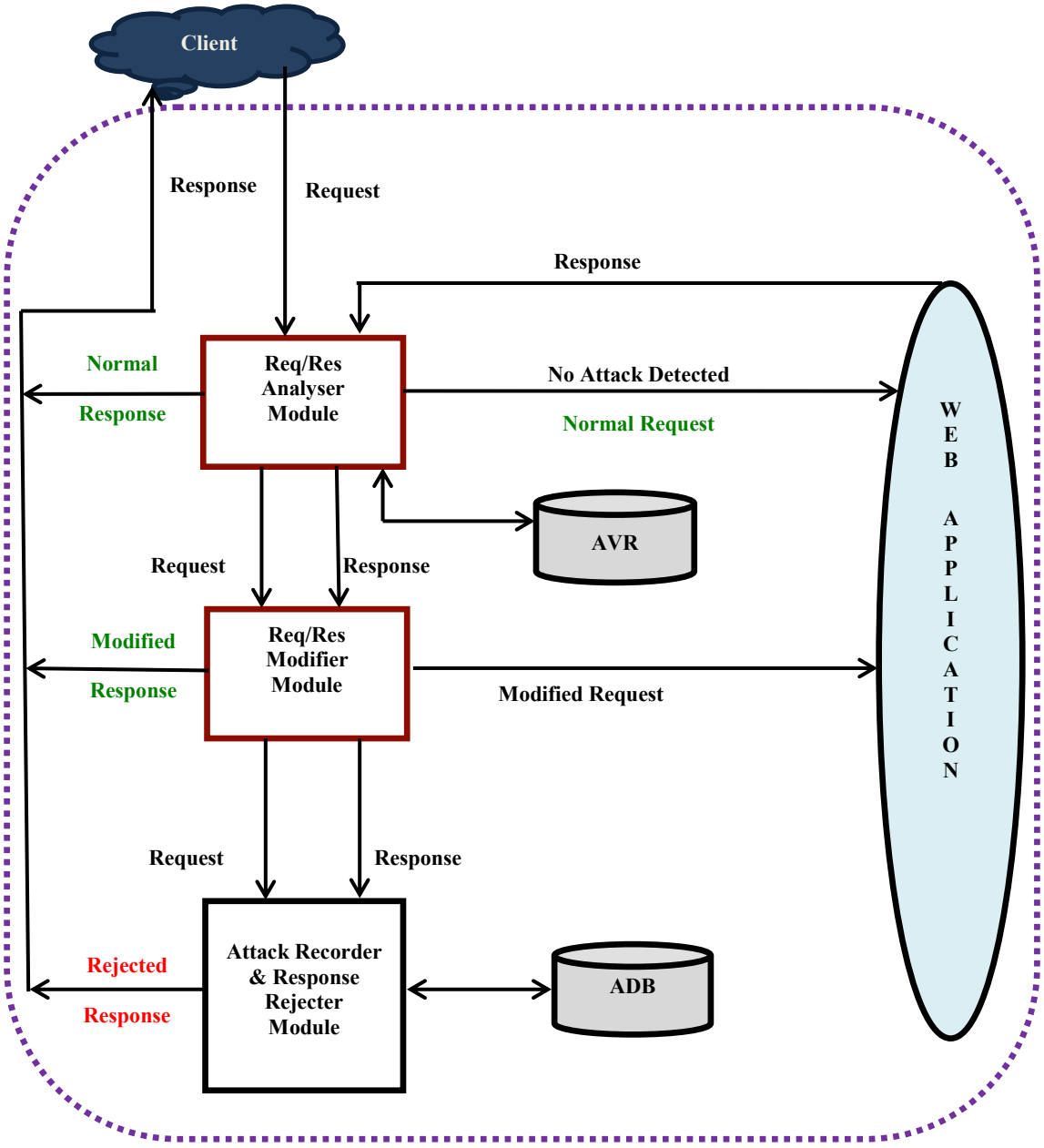

Fig- 1: Proposed Framework for XSS Attack Prevention

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

A Pattern based approach followed by some crafted rules has been used. Compiled pattern can be shown in Fig.-2.

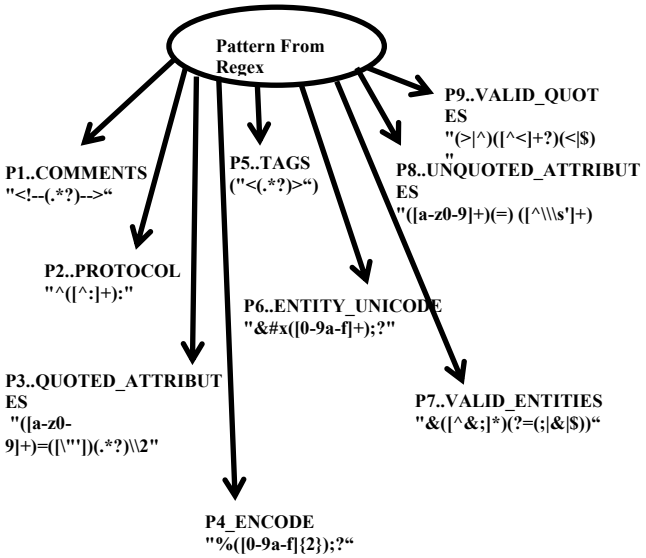

Fig- 2: Patterns for XSS Attack Signature

Regular Expressions and Wrapper classes has been used for Detection and Prevention of XSS Attack in web application. Rule Description can be summarized as-

R1. HEX value with semicolons encoding.
R2. DEC value without semicolons encoding.
R3. Mixed encoding with HEX and DEC values.
R3. Normal annotation like "/*xss*/". .
R4. Complicated annotation like "expre/* xss*/ssi/*xss*/on". Some other notations are also provided for insertion, like "/*/*/", "/*/*xss*/".

B. Stepwise Algorithm for Attack Pattern Matching-

Pre-processing Steps

Step 1 Compile Patterns P1, P2, P3, P4…

Step 2 Allocate and populate an Array, Map and List for storing allowed and disallowed values -

private final Map<String, List<String>> vAllowed = vAllowed = new HashMap<String, List<String>>();

private final Map<String, Integer> vTagCounts = new HashMap<String, Integer> ();

private final String[] vSelfClosingTags = new String[]{"img"};

private final String[] vNeedClosingTags = new String[]{"a", "b", "strong", "i", "em"};

private final String [] vDisallowed = new String[]{};

private final String[] vAllowedProtocols = new String[]{"http", "mailto"}; // no ftp.

private final String [] vProtocolAtts = new String[]{"src", "href"};

private final String [] vRemoveBlanks = new String[]{"a", "b", "strong", "i", "em"};

private final String[] vAllowedEntities = new String[]{"amp", "gt", "lt", "quot"};

private final boolean stripComment = true;

private final boolean encodeQuotes = true;

private final boolean alwaysMakeTags = true;

ArrayList<String> a_atts = new ArrayList<String>();

Input = Request(s)

Step 3 Create Matcher

Final Matcher m = P.matcher(s)

Processing steps-

Step1: Invoke Escape comments(s)

Matcher m = P1.matcher(s)

If m.find ()

Invoke quote replacement(s)

Step 2: Make a call to checkTags(s)

Matcher m = P2.matcher(s)

Make a call to checkTags(s)

If m.find ()

Invoke Process Tags(s)

Process for P3 (P_start Tag) and P4 (P_end Tag)

Step 3: a. If the tag is opening tag

Extract the name, body and ending of start tag(P_StartTag) by group(1),group(2) and group(3)

check the presence of this tag in Allowed tag list

If present

Check the body of the tag

If body contains attributes whether quoted or unquoted

then add these name and values in param name and value list respectively

b. Invoke allowedAttributes (tag name, param name)

Check the presence or absence of this tag in disallowed tag list(in vallowed map)

If this tag is allowed

Extract the attribute name which has been retrieved from body of tag

If attribute is associated with this tag

and if it is present in vProtocolAtts array

then invoke processparamprotocol (param value)

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

c. Invoke decodeEntities(s) // It will decode the encoded entities as unicode, hexadecimal to decimal

d. Validate these entities by invoking validate entity and check entity methods.

Step 4: return to process tag(s)

If this tag name is present in vSelfclosingTag Array
end it with " /"

Else If it needs closing tag end it = " "

Else close the tag

Step 5: Invoke processRemove blanks(s)

Output- Sanitized Request

IV. IMPLEMENTATION DETAILS

Hardware used can be summarized as PROCESSOR-AMD Phenom II 945 X4 3.0 GHz, RAM- 8 GB DDR3, HDD- 500 GB SATA and software required are Database-Oracle 10gXE(Xpress Edition), Web Server-Glassfish v3 & Tomcat Server, Java-JDK 6, IDE-Netbeans6.9.1,Text Editor-Edit Plus v3.20.411.

Flow diagram of Process can be shown by Figure 3-

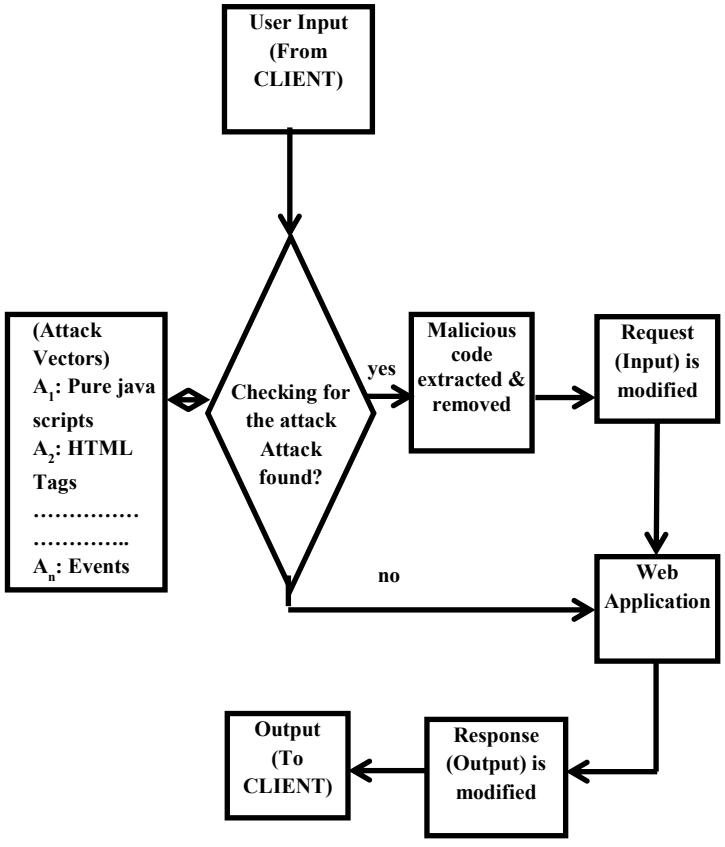

Fig-3: Flow Diagram of XSS Attack Detection & Prevention Process

V. EXPERIMENTAL RESULTS

Our framework can be applied on existing web applications as well as on manually crafted web applications. We crafted a web application for testing the framework; also we collected some web applications from a development organization. We are taking an example of XSS Attack by using META tag for testing crafted web application by us. Below the attack code has been written. Figure 4 shows the vulnerability of web application to this attack code and Figure 5 shows the web application protected by our framework.

A. Attack by using META tag-

Code: <METAHTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');"><META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');"> This code is written in Guest book and submitted by the user which then stores on server, when another user of this web application click on review link user get malicious code executed in his browser as there is no proper server side protection is available for XSS Attack.

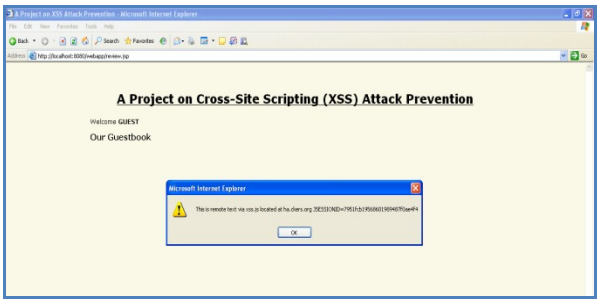

Fig- 4: Web Application Vulnerable to the XSS Attack

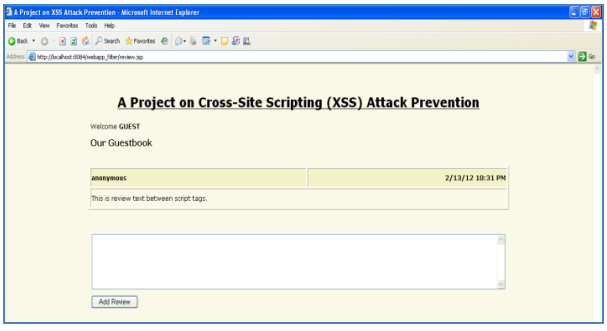

Fig- 5: Web Application Secured by Our Framework

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

Now we can show the testing result of collected web applications from development organization.

B. Testing of Online Recruitment web application- Figure 6(a), 6(b) & 6(c) show the XSS Attack vulnerability to web application and prevention of attack by proposed framework.

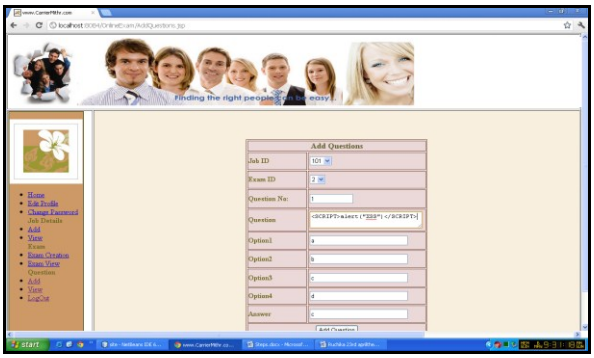

Fig- 6(a): A Person Add an Attack Code in Question Field and Submits the Form

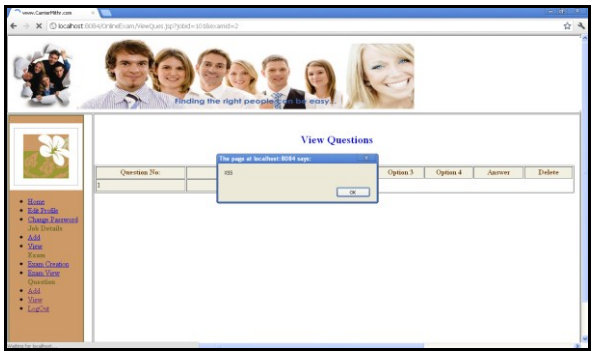

Fig- 6(b): When Someone Views This Question, an Alert Pops-Up, as Script Executes in Browser (shows vulnerability)

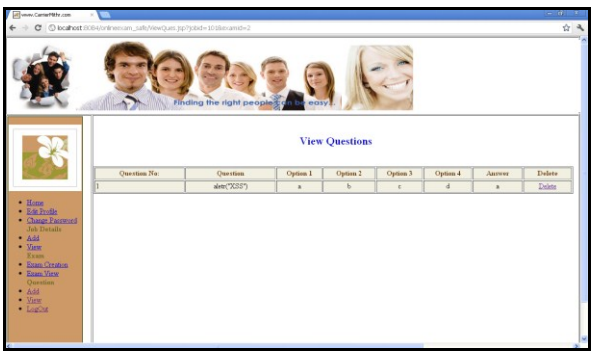

Fig--6(c): Web Application Integrated With Our Framework (Secured)

C. Testing of Get Placed web application- Figure 7(a), 7(b), and 7(c) shows the XSS Attack vulnerability to web application and prevention of attack by proposed framework.

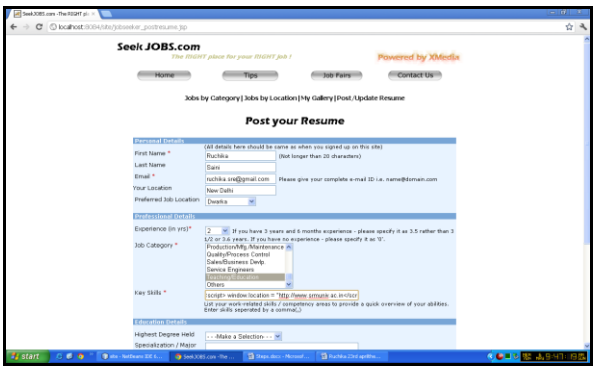

Fig- 7(a): Job Seeker Post an Attack in Key Skill Field during Form Submission

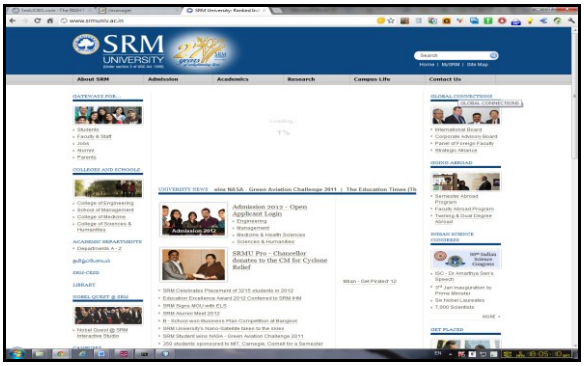

Fig- 7(b): Employee logins & Redirected to a Website in Place of Showing Records of Job Seeker (without framework)

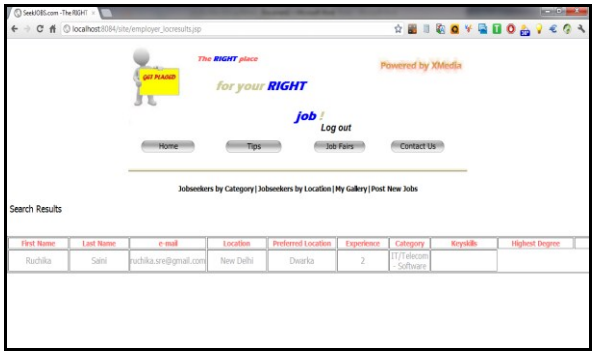

Fig- 7(c): Employee Can Now See the Record of Jobseeker and Attack Become Unsuccessful (Framework added to web app)

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

We compared our framework with some existing solution [15] for XSS attack Prevention, shown by table 1. Second comparison is based on securing java based web application by using construction frameworks [16 and 17] which is shown by table 2.

We applied the framework on collected web applications (table 3) as Online Recruitment web application, Get Placed web application, Java Boutique, My Dream Home, Real State Forum, Online Classic Music and Video Sharing web application for vulnerability testing and attack prevention. Out of these one application is not vulnerable and others are vulnerable, but successfully protected by applying the framework.

Table 1: Comparisons of Proposed Framework with Existing Solutions

SN	Properties	Multi Agent Scanner	EGuard Algorithm	Noxes (Tool)	Proposed Framework
1	Mechanism applied at	Server side	Server side	Client side	Server side
2	Detection and Prevention	Detection Only	Detection and Prevention	Detection and user controlled (manual) Prevention	Detection and auto Prevention
3	Need of Source code	Yes	Yes	No	No
4	Success Ratio in static analysis	Successful in detection	Not satisfactory	Good but with limitation	Good

Table 2: Comparisons of Proposed Framework with Existing Construction Framework for Java Web Applications

SN	Properties	Web4j Framework	Stripe Framework	Struts	Proposed Framework
1	Sanitization mechanism	Character Escaping	Simple XSS Filtering	Input Sanitization	Input sanitization and Output Encoding
2	Escaped character	33	13	Not applicable	More than 40
3	Can be applied to existing application	No	No	No	Yes
4	Integration Stage	Development	Development	Development	Any stage

ISSN 2249-6343
International Journal of Computer Technology and Electronics Engineering (IJCTEE)
Volume 2, Issue 3, June 2012

Table 3- Web Applications Protected by Proposed Approach

SN	Web Application	Injection Point	Vulnerable to Attack	Attack Prevented
1	Web Application created by us	Guest book	Yes	Yes
2	Online Recruitment web app	Form field	Yes	Yes
3	Get Placed Web app	Add Question field	Yes	Yes
4	Java Boutique web app	Forms	Yes	Yes
5	My Dream Home web app	Search field	No	Not applicable
6	Real State Forum web app	Guest book	Yes	Yes
7	Online Classic Music web app	Search field	Yes	Yes
8	Video Sharing web app	Via post method	Yes	Yes

VII. FUTURE WORK

There are many aspects of this research that can be further studied and explored. We feel that further investigations are necessary on how to detect new vulnerabilities of web applications more effectively. For example, Artificial Intelligence (AI) is one of the broadest research areas which can be utilized to find new ways for pattern generation. Many technologies such as Machine Learning and Neural Networks could be applied to detect new types of web attacks. We could employ these technologies to develop more sophisticated detection approaches based on event logs with long time observation. Consequently, our framework could be extended to improve the process of XSS Attack detection by further analysing inputs and outputs of a web application system, and thus discover and prevent new types of web attacks.

REFERENCES

[1] M. Johns, B. Engelmann, and J. Posegga, "XSSDS: Server-Side Detection of Cross-Site Scripting Attacks," in Computer Security Applications Conference, 2008. ACSAC 2008. Annual. IEEE, pp. 335–344, 2008

[2] A. Klein, "DOM based cross site scripting or XSS of the third kind,"Web Application Security Consortium, Articles, vol. 4, 2005.

[3] H. Shahriar and M. Zulkernine, "Mutec: Mutation-based testing of cross site scripting," Software Engineering for Secure Systems, ICSE Workshop on, vol. 0, pp. 47–53, 2009.

[4] A. Stock, J. Williams, and D. Wichers, "OWASP top 10," OWASP Foundation, April 2010.

[5] E.Kirda, C.Kruegel, G.Vigna, and N.Jovanovic,"Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks" Dijon, France SAC'06 April 23-27, 2006.

[6] F.Valeur, G.Vigna, C.Kruegel, E.Kirda, "An Anomaly driven Reverse Proxy for Web Applications", SAC'06 April 2327Dijon, France, ACM 1595931082/ 06/0004 , 2006.

[7] O.Ismaill, M.E.Youki, K.adobayashi, S. Yamaguch, "A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability" Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA), 2004.

[8] Christopher Kruegel, G. Vigna, William Robertson, "A multi-model approach to the detection of web-based attacks", Computer Networks 48 ELSEVIER pp. 717–738-, 2005.

[9] G.A.Lucca, A.R.Fasolino et all, "Identifying Cross Site Scripting Vulnerabilities in Web Applications", Proceedings of the Sixth IEEE International Workshop on Web Site Evolution, 2004

[10]Injection Points in Real word XSS http://sandsprite.com/Sleuth/papers/RealWorld_XSS_2.html

[11] P. Vogt, F. Nentwich, N. Jovanovic, C. Kruegel, E. Kirda, and G. Vigna. "Cross site scripting prevention with dynamic data tainting and static analysis", 14th Annual Network and Distributed System Security Symposium (NDSS), 2007.

[12] E. Gal´an, A. Alcaide, A. Orfila, J. Blasco, "A Multi–agent Scanner to Detect Stored–XSS Vulnerabilities", IEEE International Conference on Internet Technology and Secure Transactions (ICITST), pp.332-337, June 2010

[13] M. James Stephen, P.V.G.D. Prasad Reddy, Ch. Demudu Naidu, "Prevention of Cross Site Scripting with E-Guard Algorithm", International Journal of Computer Applications Volume 22– No.5, pp. 30-34, May 2011.

[14] XSS Attack Vectors at http://ha.ckers.org/xss.html

[15] shar, l.; tan, h.; "methods for defending against cross site scripting attacks" ieee journal of computer, volume 44 issue 10, august 2011

[16]StripeFramework,http://stripes.sourceforge.net/docs/current/javadoc/index.html

[17] WEB4J Framework, http://www.web4j.com/UserGuide.jsp#XSS

AUTHORS-

Dr R P Mahapatra- *Professor and HOD of CSE Department SRM University Chennai NCR Campus*
Email- mahapatra.rp@gmail.com

Ruchika Saini- *pursuing MTech CSE 4th Semester from SRM University Chennai NCR Campus*
Email- ruchika.sre@gmail.com

Neha Saini- *Assistant Professor in MCA Department SRM University NCR Campus*
Email- nehasaini.7@gmail.com