*I.J. Information Technology and Computer Science,* **2016, 4, 86-95**
Published Online April 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2016.04.10

# Automated Client-side Sanitizer for Code Injection Attacks

**Dnyaneshwar K. Patil**
Department of Computer Engineering, VIIT, SPPU University, Pune, India
E-mail: dnyaneshwar11.patil@gmail.com

**Dr. Kailas R. Patil**
Department of Computer Engineering, VIIT, SPPU University, Pune, India
E-mail: kailas.patil@viit.ac.in

***Abstract***—Web applications are useful for various online services. These web applications are becoming ubiquitous in our daily lives. They are used for multiple purposes such as e-commerce, financial services, emails, healthcare services and many other captious services. But the presence of vulnerabilities in the web application may become a serious cause for the security of the web application. A web application may contain different types of vulnerabilities. Cross-site scripting is one of the type of code injection attacks. According to OWASP TOP 10 vulnerability report, Cross-site Scripting (XSS) is among top 5 vulnerabilities. So this research work aims to implement an effective solution for the prevention of cross- site scripting vulnerabilities. In this paper, we implemented a novel client-side XSS sanitizer that prevents web applications from XSS attacks. Our sanitizer is able to detect cross-site scripting vulnerabilities at the client-side. It strengthens web browser, because modern web browser do not provide any specific notification, alert or indication of security holes or vulnerabilities and their presence in the web application.

***Index Terms***—Web application, Cross-site scripting, Vulnerability, Sanitizer.

## I. Introduction

Security is the important factor to be considered in the web engineering. A web application may contain different types of vulnerabilities. For example: if a web application is vulnerable, it may contain vulnerabilities like Injection, Broken Authentication and Session Management, Cross-Site Scripting (XSS), Insecure Direct Object References, Security Misconfiguration, Sensitive Data Exposure, Missing Function Level Access Control, Cross-Site Request Forgery (CSRF), Using Components with Known Vulnerabilities, Unvalidated Redirects and Forwards. Among these vulnerabilities, Cross-site scripting is among top 5 web application vulnerabilities [4]. In general, cross-site scripting may happen due to insertion of untrusted script code into a web page. For preventing cross-site scripting attacks existing systems contain Sanitizers like Xss Sanitizer Plugin, Jsoup Sanitizer and Haskell-xss-sanitize. Xss Sanitizer Plugin has used the OWASP ESAPI library to sanitize request parameters. Xss Sanitizer Plugin is able to detect XSS attacks, but they did not specify exactly which type of XSS may be detected by their sanitizer. Next Jsoup Sanitizer is allowing known-safe tags and attributes and values through into the cleaned output. This Jsoup Sanitizer works only with whitelist provided to it. Haskell-xss-sanitizer uses Tagsoup for parsing HTML, but it does not maintain all white spaces. This research work implements an idea by considering the limitations of the existing cross-site scripting sanitizers. Our technique considers all possible scripts for cross-site scripting vulnerabilities. According to a survey [6] conducted by Cenzic Inc. 96 percent of tests web applications in 2013 have at least one or more serious security vulnerability. The application layer is continuously targeted by attackers as a soft way for attack. 99 percent of vulnerabilities found in their tested web applications in year 2012 and 96 percent of vulnerabilities found in the year 2013. A median of these vulnerabilities per web application is 13 for year 2012 and 14 for year 2013 respectively. Cross-site scripting is the topmost vulnerability among web applications. Most of the web applications are vulnerable due to unawareness of web application developers about security practices. Current browsers are having the extensions for detecting specific vulnerability attacks, but none of the browser having all in one solution for the detection of all these web vulnerabilities. Our proposed system makes a path for developing all-in-one solution for the detection of the web application vulnerabilities. In summary, we make the following three contributions to enhancing web security:

(1) We study the web application vulnerabilities and identified their security mechanisms with limited solutions.
(2) We build robust and client-side based security mechanism to protect web applications from cross-site scripting vulnerabilities.
(3) Our evaluation result shows the effectiveness of the system.

 

In the remaining sections of this paper, Section 2 describes the background of the web application working scenario and different types of cross-site scripting vulnerabilities, Section 3 describes the motivation for choosing this research work, Section 4 describes research works on cross-site scripting Vulnerabilities and their preventive measures. Section 5 describes our observations on this entire topic of the cross-site scripting vulnerabilities, Section 6 describes proposed system with its architecture, Section 7 describes Implementation details, Section 8 describes the results of the implemented system, Section 9 describes the limitations of the implemented system in discussion and section 10 describes a conclusion about this research work on cross-site scripting vulnerabilities.

## II. Background

Web application is the software that is able to run in a web browser. Such web applications can be developed with the help of programming languages (for example: HTML, CSS and JavaScript, etc.) that are supported by the web browser. These programming languages rely on the web browser for rendering web applications. Due to the ubiquitous nature of the web browser web applications are becoming more popular. Another reason for the popularity of the web application is its attractive graphical user interface. The main reason for becoming popular of the web application is that to maintain its adaptability excepting the trouble of installing the software on strongly millions of web client computers. Web application borrows itself towards multi-tiered perspective by its occurrence. Figure 1 shows a web application with its working on the client-side mechanism and server-side mechanism. The client-side mechanism can be used by web browser for rendering web application. It may contain JavaScript, Flash, etc. and by using this Client-side mechanism user can use a web browser for searching the content on the web or to do his intended work. Web users may use multiple web browsers like Mozilla Firefox, Google Chrome, Safari and many more for making requests to the web server [17].

Web browser works at the interface between web application user and web server. Web user enters a URL into the address bar of the web browser for making requests to the web server or web user can use the search engine for making requests to the web server and using web application. In between web browser and web server once the web user enters a keyword into the search engine at that time web browser generates HTTP request and sends it to the web server. Here security of the generated request depends on the HTTP headers used by the web application developers as well as policies used by the web application developers. So it is necessary to focus on the Client-side mechanism to make stronger protection for the web application to save important data from cyber criminals. Continuous growth in web application development without considering its vulnerable status is an important factor for web security. Web application is useful for e-commerce services, financial organizations, governmental websites and social media like Facebook, Twitter, etc., all these service providers and information users require online security for their important information, but due to different types of vulnerabilities present in the web application, an attacker can easily access this valuable information of user or of the organization. Now we are coming to web application vulnerability that is one of the top 5 web application vulnerabilities. It is cross-site scripting vulnerability.

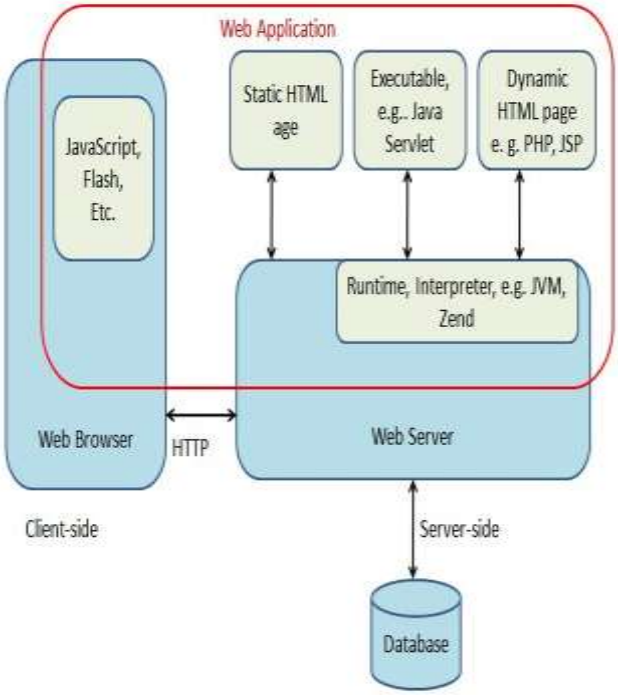


Fig.1. Web application working scenario

Cross-site scripting vulnerabilities are common vulnerabilities in most of the web applications. Following are the types of cross-site scripting attacks:

### *2.1 Stored XSS Vulnerabilities*

Stored cross-site scripting vulnerability is the most powerful type of the XSS attack. When web application user provides information to the web application that information is stored permanently on the server and later displayed on the webpage by the web application without encoding it with entity encoding of the HTML language. Stored XSS vulnerability is also known as second order vulnerability [3].

Figure 2 shows a mechanism for stored XSS. Unstrusted data accepted from the web user through web browser may be stored on the server-side database permanently. In this scenario, if the user gives executable script as an input it will be stored on the server-side database permanently and will be executed always whenever request come to that webpage. A real world example of this vulnerability is Samy Myspace Worm.

 

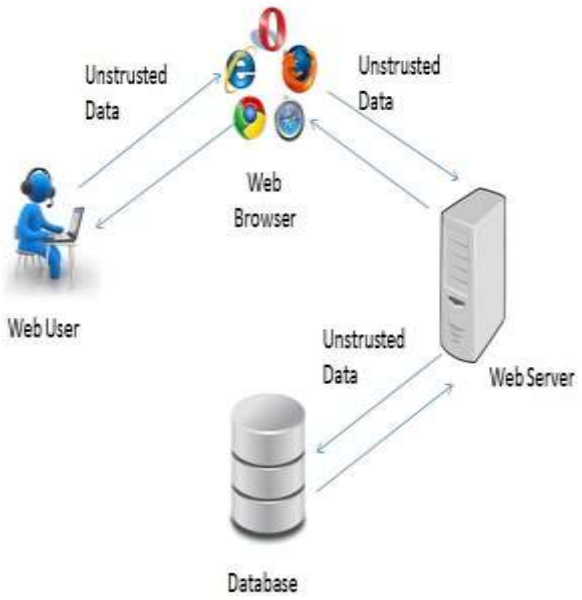


Fig.2. Stored XSS

### *2.2 Reflected XSS Vulnerabilities*

When data provided by the web application user are used for reflection by the server according to the requested web page for generating the expected result, then this mechanism can become sources for the reflected XSS type of the vulnerability. It can be used for denial of service attacking. For example, consider the following case: By using <meta> tag .php page can be reloaded

```
<META HTTP−EQUIV=Refresh CONTENT=”1; URL= h t t p : / / www. somethingonursite.com/ururl . php?>
```

As shown in the above script in PHP, particular page will be refreshed after each second. So it will become as an infinite loop for refresh requests which will cause database server down due to flooding of requests. In this way denial of service attack may happen on web application.

### *2.3 DOM-based XSS Vulnerabilities*

DOM-based XSS vulnerabilities can occur in web page client-side script itself. Suppose JavaScript accesses a URL request parameter and takes that information to write some HTML to its own page which is not encoded using HTML entities, then DOM-based XSS vulnerability will be present there. This written data will be reinterpreted by web browsers that can include additional client-side script [4]. For example: We have web application as

```
http://yourwebsite.com/entity5/worldencapsultaed/forigendata/returnpage11.php
```

an attacker can write code for DOM-based attack for above mentioned url as

```
http://yourwebsite.com/entity5/worldencapsultaed/forigendata/returnpage11.php?<script>alert( “DOM −BASED XSS ATTACK”)</script>
```

So the above code will generate DOM-based XSS attack for that particular webpage.

## III. MOTIVATION

Web applications are becoming popular and ubiquitous in our daily lives due to their importance in the current era of digital world. It is necessary to use web browser for web application user to access the web application. And these web applications are useful in following important fields: Online Banking, Government Services, Social Media Websites, e-commerce. All these fields are important for maintaining fast online transactions with their intended purposes, but these services must have a secure mechanism to handle their services for the web application users. But today’s web applications are most vulnerable to cross-site scripting. On the other hand web application, web browser and web developers are the motivating factors for this research work. Because if web application has not implemented security policies for prevention of the XSS attack, then it will be vulnerable to the XSS attacks. Browser is also important to consider because it is responsible for executing untrusted code provided by the user. Further, how these factors are important for web security are explained as:

1) Web Application: Before deploying web applications on the server, it must implement security policies for avoiding attacks like cross-site scripting. Otherwise that web application may be vulnerable to the cross-site scripting attacks as well as vulnerable to other possible attacks. Therefore, web application should have security policies for avoiding cross-site scripting attacks.
2) Web Browser: The web browser is the medium for accessing web applications. When a user enters input to the web application, web browser executes it, if it contains executable scripts otherwise treats that input as plain text.

For example: suppose we took URL form vulnerable web site

 

```
http://public−firing−range.appspot.com/reflected/parameter/body ?q=a
```

This web page has source code as

```
<html>
<body>
a
</body>
</html>
```

When a user enters script in the URL field, it will be executed directly at the client-side by the web browser.

```
http://public−firing−range.appspot.com/reflected/para meter/body?q=a<script>alert(”U r Attacked ”)</script>
```

it will directly affect the source code of the web page as given below:

```
<html>
<body>
a<script> alert(“ U r Attacked”)</script>
</body>
</html>
```

Here browser does not consider URL as the only URL but it treated that URL as executable code. Hence browser does not have a specific mechanism for treating user given input on the basis of their contents. On the other hand, there are multiple web browsers available for accessing web applications like Mozilla Firefox, Google Chrome, Internet Explorer and so on. But all of these available web browsers do not provide any specific alert related to web application vulnerability to the user.

3) Web Developer: A web developer may use multiple web technologies for developing web applications like HTML, JavaScript, CSS, VBScript, PHP and many more. But Web developers are developing web applications continuously without considering the factor of security of the web application and this may become cause for an attacker to steal sensitive information of the web user or the valuable information about the organization. There are alternative security practices available for developing web applications, but web application developers are not aware about these security practices. So it is necessary for web application developers to pay attention towards secure practices for developing web applications which will reduce risks of web application vulnerabilities. Cross-site scripting is the most effective vulnerability among web vulnerabilities. Existing solutions for XSS are weak for protecting web applications. Next section gives brief idea about cross- site scripting vulnerabilities.

## IV. Literature Survey

Literature survey is broadly classified into three categories:

### *4.1 Existing XSS Sanitizers*

XSS Sanitizer Plugin [8], Jsoup Sanitizer [5] and Haskell-xsssanitize [7] are the existing XSS sanitizers. XSS Sanitizer Plugin [8] has used the OWASP ESAPI library for sanitizing request parameters. This XSS sanitizer plugin automatically works for cleaning the browser from XSS code, but it does not provide information whether it detects all types of XSS or detects only particular XSS. Jsoup Sanitizer [5] is the XSS sanitizer that performs by parsing the input HTML by creating a safe sand-boxed mechanism. Later on iterating through parse tree and only permitting known secure tabs and attributes through the sanitized output. Haskell-xss-sanitize is the XSS sanitizer that allows user to accept html from untrusted sources initially filtering it through a whitelist. The whitelist filtering is comprehensive with including support of CSS style attributes. Haskell-xss-sanitize uses the TagSoup parser to parse the HTML. But this TagSoup does not maintain all white space. For Example: TagSoup is not able to distinguish between the following cases:

```
<a href=”foo”>, <a href=foo>
<a href>, <a href >
<a></a>, <a/>
```

### *4.2 String solvers for web application security*

S3 [24] and Z3-str [28] are the string solvers for web application vulnerability detection and analysis respectively. S3 [24] is the symbolic string solver based on its own constraint language. Their algorithm initially makes use of a symbolic representation in such a way that membership in a set termed by the regular expression may be encoded as equations of strings. Z3-str is nothing but the general purpose string solver. It treats strings as a primitive type that avoids the inherent shortcomings observed in many existing solvers which encode strings in terms of other primitives. Their logic of the plugin is in three sorts boolean, int and string. Strings sorted terms are having functions as replace, concatenation and sub-string. Strings sorted terms are included with the string constant and variables of arbitrary length.

### *4.3 JavaScript based vulnerability detection systems*

Yue, C. and Wang, H. [26], M. Cova, Kruegel, G. Vigna [12] and Finifter M., Weinberger J., Barth, A. [14] have considered vulnerabilities occurring due to the JavaScript programming language. Yue, C. and Wang, H. [26] presented an analysis of insecure JavaScript practices and suggested alternative JavaScript practices for it. They examined 6805 unique websites for the

 

measurement and an analysis of JavaScript. According to their analysis they found 66.4% of analyzed websites convicts unsafe practices with inclusion of JavaScript into the top level documents of their web pages. 44.4% of their measured websites used eval () function for dynamic generation and execution of JavaScript codes in their web pages. And they also found the function document.write () of the JavaScript and property of innerHTML are very popular instead of alternative secure practices for them. But they have specific solutions for avoiding web application vulnerabilities that are related to avoiding insecure JavaScript practices. M. Cova, Kruegel, G. Vigna [12] presented the solution for detection of the attacks which are possible due to execution of the online downloaded files. For implementation they have developed a system that uses machine learning techniques and a number of features to establish the features of the usual JavaScript code. Their system is also capable to detect the behavior of abnormal JavaScript code by imitating its behavior and equating it to launch prominence [4]. This solution presented by authors cannot protect web applications from JavaScript malware. Finifter M., Barth, A. [14] introduced a special solution for preventing capability leaks of the subsets of JavaScript. In this paper they proposed new technique for preventing capability leaks of JavaScript by improving statically verified JavaScript subset [14]. They explained about one-third of Alexa Top 100 web applications is exploitable by an advertisement by the ADsafe which is verified. They proposed an updated mathematically verified subset of the JavaScript which uses namespaces. It is only possible to prevent web application from capability leaks of JavaScript codes it means the authors have considered only capability leak problem of JavaScript.

### *4.4 Other web vulnerability detection systems*

Prophiler [9], SecuBat [17], [13], [15], [11], [27] are the web application vulnerability detection systems. Prophiler [9] is the filter that executes fast for detecting malicious web pages. It explains the concept of the attacks that are happening at the time of downloading and prevention techniques for it. For preventing drive by download attacks, they have built a filter named as 'Prophiler' which is used for detection of the harmful web pages. SecuBat [17] is the web application vulnerability scanner. SecuBat gives way for how to find potentially vulnerable websites. By usefulness of the SecuBat authors were able to detect many potential vulnerable websites. For validating the performance and accurateness of the SecuBat authors picked 100 interesting websites from the potential list of victims for the purpose of further analysis as well as to confirm exploitable flaws in the recognized web pages. They also mentioned all of their victims were from well known industrial companies and of vulnerable web sites about possible security problems. The only limitation of this proposed solution is we have to submit websites to this scanner means it is not based on the client-side approach. [13] Proposes solution for analysis of the websites of the design flaws that are visible to the user. User visible security design flaws may contain flaws that can become a risk for web user. Further authors examine that the influence of user visible security by examining websites from 214 United States commercial institutes. They intentionally chose commercial web applications because of their high demand for security [13]. After experimentation they found lots of faults which may direct web clients to make worse security permissions. According to their survey, 76 percent of their examined websites containing a minimum one design fault which indicates that these design flaws have not understood widely even experts who have information about security and responsibility of security. Therefore finally they implemented solution to recover from these security design flaws which are user visible design flaws. This paper detecting only design flaws which are user visible. [15] Explains the concept of web password habits of web users. It gives protection to the password given by the user to his system and which stored on a web browser [15]. This system is having client-side approach, but related to the protection of passwords that are stored in web browsers. [11] Is the research work over security flaws in GUI logic. As per their perspective for achieving security at the end point, conventional security techniques are incapable if the integrity of HCI is compromised by third party. Authors are totally focusing on the vulnerabilities, which are only relate to GUI logic means they have implemented their solution with the specific consideration of the problem. [27] is the research work related to browser saved passwords. According to their perspective web application users are facing problems with the intimidating challenges of forming, memorizing and using safe as well as strongest passwords for maintaining their important assets on respective web applications. They have suggested that their system can be implemented in other global browser. They have implemented a different approach for the protection of browser saved passwords rather than the conventional password manager systems. [23] explains the analysis of existing malware detection systems. [20] is the system that maintains security for personal information. [16] is the practical approach by applying a mathematical formulation of web vulnerabilities. [25] is the research work related to reduction of denial of service attacks using web service filters. [18] is the system for analyzing the relationship between customer and organization on the Internet. [10] is the novel technique for web page classification on the basis of a specific domain. [21] is the system that predicts the navigation of the user by using weighted association rules. [22] is the research work related to the reliability assessment of the web application. [19] is the system prevents web applications from forgery attacks.

### *4.5 Motivational Survey*

Security header for every web application plays an important role in maintaining web application security, so it is necessary to provide security headers for web application. For checking different security headers

provided by web applications, we have calculated statistics shown in figure 3. Strict Transport Security (HSTS) header is used for HTTPS connections and it is used in 12 percent of our tested web applications. Only 8 percent web applications are using content security policy for protection from cross-site scripting vulnerabilities. X-Frame-Options headers are used for preventing web application from clickjacking attacks. X-Frame-Options are applied for 60 percent of our tested web applications. X-XSS-Protection is the security header used in 40 percent of tested web applications.

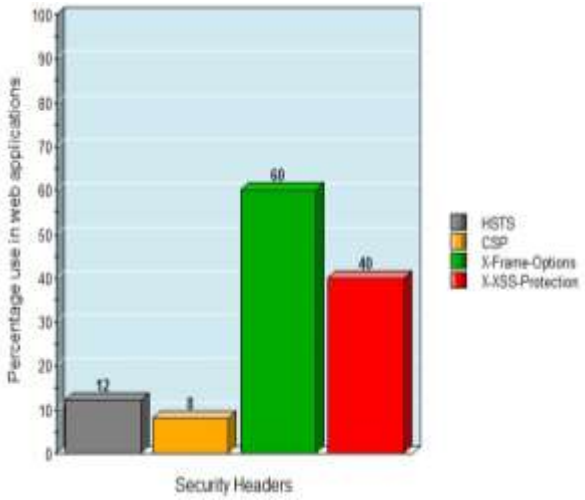


Fig.3. Security Headers Analysis

## V. Our Observations

Existing solutions for XSS vulnerabilities are very specific and these mechanisms are easily breakable. On the other hand recent web applications are consisting very complex structures, but due to security loopholes inside these structures, they are prone to various web application vulnerabilities. Web development teams are not aware about secure web development practices and they are developing web applications without considering the security factor. Therefore, it is necessary to construct all-in-one solution for web application vulnerabilities in the web applications.

## VI. Proposed System

Proposed system architecture consists of modules DOM, Input field capture, Input analyzer, Links, Text area, Sanitizer and XSS Notification.

### *6.1 System architecture overview*

Our proposed system architecture gives the exact idea about prevention of cross-site scripting vulnerabilities. DOM module will access the current webpage's DOM and that DOM will help to Input field capture module for capturing different inputs. The further Input analyzer will analyze each input field data from the input field capture module. Analyzed data will be forwarded for Links module, and Text area module. Next Sanitizer is used to sanitize user provided input with the help of Links module and a Text area module. Finally, the XSS notification module generates a notification for the user about input provided by the user.

### *6.2 System Architecture*

Figure 4 shows system architecture. Following are the modules of the proposed system.

1. DOM (Document Object Module): A programmer can build documents, navigate their structure or delete elements and contents with the help of the DOM. Anything found in an HTML or XML document can be manipulated using the DOM. It creates a DOM tree for each document.
2. Input field capture: The input field capture module accepts inputs provided by the web user. Input provided may be link or text area by the web user.
3. Input Analyzer: This module takes all input fields of the current loaded web page. Further, it categorizes inputs into links and Text area fields and forwards it to the next module according to the inputs categorization. 4. Links: The links module maintains a queue for links present on the loaded web page. Further, it feeds these links one by one to the sanitizer module for XSS vulnerability checking.
4. Text area: The text area module accepts texts entered by the web user through previous modules and maintains queue for all text area fields present on the current web page.

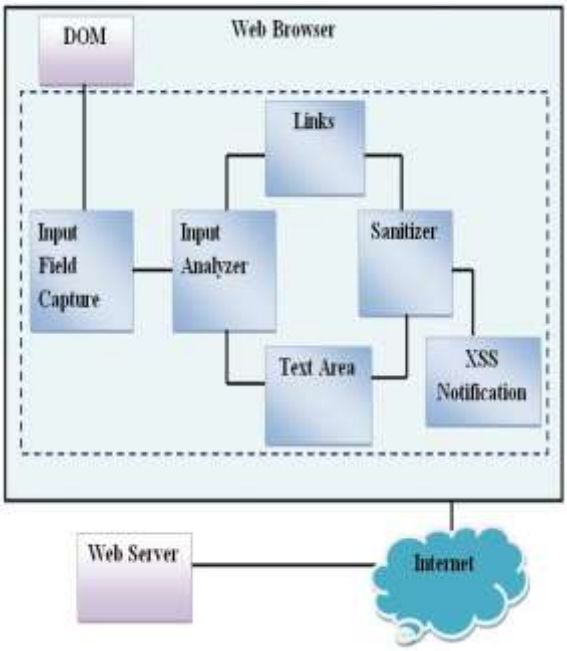


Fig.4. System Architecture

5. XSS Notification: Once the XSS vulnerability is detected in that webpage, the XSS Notification module will generate a notification message for the web user. For capturing the user's attention we

are applying a red border to the XSS vulnerable web page. These all notifications will be generated automatically when a web user will access web applications through the web browser.

### *6.3 Algorithm*

Algorithm of the proposed architecture gives an exact working scenario of the system. Input is text entered by the web user or link provided by the web user and output is notification from browser to the user about XSS vulnerability.

**Algorithm 1**: DetectXSS

(1) Initialize user request
//Take input from web user and it will either text area or link.
(2) Capture input fields
//Text area or link entered by user and forward these fields to input analyzer.
(3) Analyze input fields
//categorization of the input fields into text area and links
(4) Links or text area
If user enters link as input
Feed this link to sanitizer
Else
Feed text area to sanitizer
EndIf
(5) Sanitization
Process user entered input and generate message for XSS notifier.
(6) XSS notifier

At last notify to the user whether the current web page is vulnerable or not.

The user may request by giving input through the URL address bar. The user may also input through text box or by clicking on the link present on the web page or user may enter text in text area fields present on the web page. Once the request is initialized by web user, it will be fed to the input capture module of the system. Further that input will be analyzed through input analyzer. Input analyzer will categorize inputs into the links and text area fields. Further Sanitizer processes input fields and forwards message to XSS notifier about the status of user entered input.

## VII. Implementation Details

We have implemented a browser extension for prevention of cross site scripting vulnerability at the client-side. A total line of code is approximately 2200. We have used Jetpack framework for implementation of the system. JavaScript is the programming language used for implementation. APIs [2] used in the system are tabs, page-mod, page-worker and notifications. Tabs API is used for checking currently loaded tab in the web browser. Page-mod and page-worker API are used for running scripts in the context of web pages and for creating invisible pages and accessing its DOM.

## VIII. Results Analysis

Our implemented system gives effective results for prevention of cross-site scripting vulnerabilities. We tested our system for the inputs given by a web user. If the user provides normal input to the web application, then our system will work normally. But if the user gave executable scripts as normal input to the web application then our system generates notification about vulnerable status of web applications to the user.

### *8.1 Effectiveness*

As we explained example, in motivation for web browser that have url as:

```
http://public−firing−range.appspot.com/reflected/parameter/body ?q=a
```

when user will try to insert XSS vulnerable script in the URL at the same time our implemented system will give notification to the user about it's vulnerable status. This shows the effectiveness of our system. We have also considered following example for the checking effectiveness of our system.


Fig.5. Vulnerable web application before user input

Figure 5 shows web application that is vulnerable to the cross-site scripting attacks. Initially, when user enters a URL into the address bar it loads the web application into web browser.

Figure 6 shows how our proposed system will protect user from cross-site scripting. Initially the URL is loaded and if the user tried to insert cross-site scripting attackable scripts as an input to the web browser. It will mark web page by red colored border for user attention.

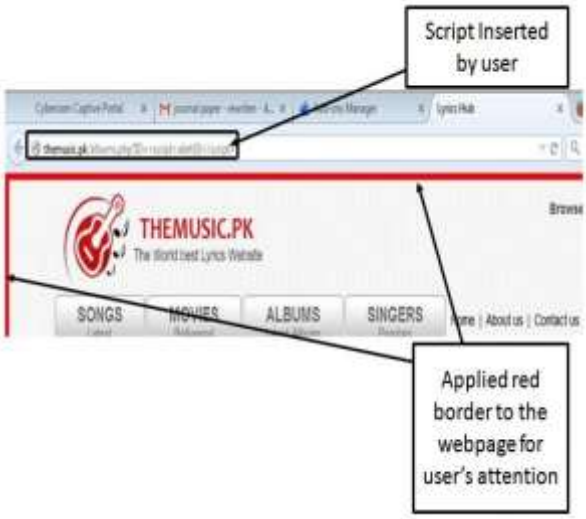


Fig.6. Vulnerable web application after user input


Fig.7. Notification

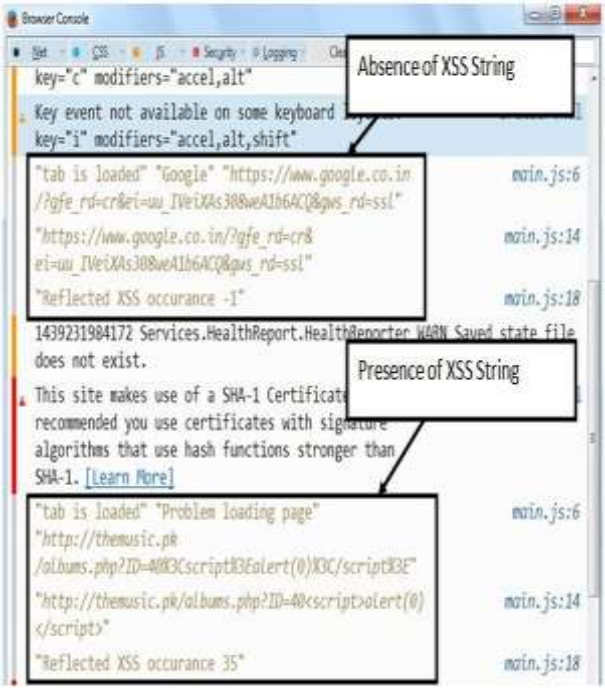


Fig.8. Occurrence of the XSS String

and it will also create notification to the user about web application's vulnerable status as shown in Figure 7. Notification to the user may be seen at the right-bottom corner of the web browser or at the right-top corner of the web browser.

Figure 8 explains the details about the exact location of the XSS attackable script in the user input. We have used Linux operating system for implementation of the system. JavaScript is the programming language for developing the system.

### *8.2 Performance Overhead*

We checked performance of the system using Dromaeo [1].

Table 1. Performance testing of our system

| **Test Names** | **Without Our System** (runs/s) | **With Our System** (runs/s) |
|---|---|---|
| Arrays | 1046.97 | 1107.37 |
| Base 64 Encoding and Decoding | 1716.55 | 1611.21 |
| Code Evaluation | 594.74 | 497.75 |
| Compute Bits in Byte | 24375.40 | 24372.80 |
| DOM Attributes | 2787.35 | 3253.39 |
| DOM Modification | 385.14 | 389.73 |
| DOM Traversal | 508.86 | 515.66 |
| Validate User Input | 814.22 | 822.92 |

Above table summarizes performance of our implemented system with a real world web browser.

### *8.3 Compatibility*

We tested our approach with 100 real world web applications. In our tested environment, it doesn't affect the working of real world web applications. None of our tested web applications have affected, this shows compatibility of the system.

## IX. Discussion

The user may give input to the web application through two ways: Links and Text area. Considering these input fields we have implemented our system. Once user gives input to the web application that input will be examined through the implemented system and final notification may be generated on the basis of the vulnerable status of the web application. This implemented system is limited to the capture and analyze user inputs from the web user. It is able to detect vulnerable scripts present in the system. User may enter script in any scripting language so we have considered this issue for implementation. Our system may become more powerful by adding features like Artificial Intelligence techniques to input capture module and input detection module of the system. This system may become a path for prevention of the all web application vulnerabilities.

 

## X. Conclusion

Existing solutions for web application vulnerabilities are specific for particular vulnerability and applicable to particular web applications. Our proposed system is the state-of-the-art solution for the detection of the cross-site scripting vulnerabilities among the web applications. In specific our system is able to detect reflected cross-site scripting as well as the stored cross-site scripting vulnerabilities. The future scope of the proposed system will be the all-in-one solution for all kinds of the web application vulnerabilities. Another perspective of the proposed system in future will be to focus on solutions for making aware of web developer about secure practices in web development. These secure practices will make stronger security for web application and that will be another solution for protecting web applications from web vulnerabilities. self protection Our proposed system may also use artificial intelligence algorithms to detect web application vulnerabilities.

## References


[1] Dromaeo javascript performance testing. Available at http://dromaeo.com/, JavaScript Performance Testing.

[2] Mozilla developer network. Available at https://developer.mozilla.org/en-US/Add-ons, Mozilla.

[3] Mozilla firefox extensions. Available at https: //addons.mozilla.org/en-US/firefox/extensions/, Mozilla Firefox.

[4] New international project on web vulnerabilities. Available at https://www.owasp.org/index.php, OWASP.

[5] Prevent xss with jsoup sanitizer. Available at http://jsoup.org/cookbook/cleaning-html/whitelist-sanitizer, JSOUP.

[6] Survey by cenzic inc. application vulnerability report.. Available at https://www.info-pointsecurity.com/sites/default/files/cenzic-vulnerability-report-2014.pdf, Vulnerability Report 2014.

[7] The xss sanitize package. Available at https://hackage.haskell.org/package/xss-sanitize, The XSS Sanitizer.

[8] Xss sanitizer plugin. Available at https://grails.org/plugin/xss-sanitizer,XSS Sanitizer Plugin.

[9] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: A fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 197–206, New York, NY, USA, 2011. ACM.

[10] Vivek Chandra and Nidhi Saxena. Article: An improved technique for web page classification in respect of domain specific search. *International Journal of Computer Applications*, 102(4):7–10, September 2014.

[11] Shuo Chen, Jose Meseguer, Ralf Sasse, Helen Wang, Yi min Wang, Shuo Chen, Jos Meseguer, Ralf Sasse, Helen J. Wang, and Yi min Wang. A systematic approach to uncover gui logic flaws for web security, 2006.

[12] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 281–290, New York, NY, USA, 2010. ACM.

[13] Laura Falk, Atul Prakash, and Kevin Borders. Analyzing websites for user-visible security design flaws. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, SOUPS '08, pages 117–126, New York, NY, USA, 2008. ACM.

[14] Matthew Finifter, Joel Weinberger, and Adam Barth. Preventing capability leaks in secure javascript subsets. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*, 2010.

[15] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA, 2007. ACM.

[16] Mohamed Ghazouani, Sophia Faris, Hicham Medromi, and Adil Sayouti. Article: Information security risk assessment a practical approach with a mathematical formulation of risk. *International Journal of Computer Applications*, 103(8):36–42, October 2014.

[17] Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic. Secubat: A web vulnerability scanner. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 247–256, New York, NY, USA, 2006. ACM.

[18] Navjot Kaur and Himanshu Aggarwal. Article: Web log analysis for identifying the number of visitors and their behavior to enhance the accessibility and usability of website. *International Journal of Computer Applications*, 110(4):25–30, January 2015.

[19] M. V. Kishore, G. Pandit Samuel, N. Aditya Sundar, M. Enayath Ali, and Y. Lalitha Varma. Article: A novel methodology for secure communications and prevention of forgery attacks. *International Journal of Computer Applications*, 96(22):5– 12, June 2014.

[20] Anuradha K. Kudlikar and Meghana B. Nagori. Article: Refinement in personalize web search system with privacy protection. *International Journal of Computer Applications*, 117(6):1–6, May 2015.

[21] Zeynab Liraki, Ali Harounabadi, and Javad Mirabedini. Article: Predicting the users' navigation patterns in web, using weighted association rules and users' navigation information. *International Journal of Computer Applications*, 110(12):16–21, January 2015.

[22] Laxmi Shanker Maurya and Anil Kumar Malviya. Article:Web application reliability assessment using error and workload data obtained from server error and access logs. *International Journal of Computer Applications*, 97(15):6–9, July 2014.

[23] Smita Ranveer and Swapnaja Hiray. Article: Comparative analysis of feature extraction methods of malware detection. *International Journal of Computer Applications*, 120(5):1–7, June 2015.

[24] Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. S3: A symbolic string solver for vulnerability detection in web applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1232–1243, New York, NY, USA, 2014. ACM.

[25] Sonali Utsai and Ram B. Joshi. Article: Dos attack reduction by using web service filter. *International Journal of Computer Applications*, 105(14):4–9, November 2014.

[26] Chuan Yue and Haining Wang. A measurement study of insecure javascript practices on the web. *ACM Trans. Web*, 7(2):7:1–7:39, May 2013.

[27] Rui Zhao and Chuan Yue. All your browser-saved passwords could belong to us: a security analysis and a cloud-based new design. In Elisa Bertino, Ravi S. Sandhu, Lujo Bauer, and Jaehong Park, editors, *CODASPY*, pages

333–340. ACM, 2013.
[28] Yunhui Zheng, Xiangyu Zhang, and Vijay Ganesh. Z3-str: A z3-based string solver for web application analysis. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 114–124, New York, NY, USA, 2013. ACM.

## Authors' Profiles

**Dnyaneshwar K. Patil:** Post-graduate student for master degree for computer engineering in Vishwakarma Institute of Information Technology (VIIT, Pune of SPPU University, interested in web security.

**Dr. K. R. Patil:** KAILAS PATIL eceived the PhD in Computer Sci- ence, National University of Singapore (NUS), Singapore, in 2014. He is currently a Professor with the Department of Computer Engineering at Vishwakarma Institute of Information Technology (VIIT), University of Pune, India. He is a Mozilla Rep in India. His research interests include information security, cloud security, and web security. He also served as a reviewer in many SCI-index journals, other journals, other conferences.

**How to cite this paper:** Dnyaneshwar K. Patil, Kailas R. Patil,"Automated Client-side Sanitizer for Code Injection Attacks", International Journal of Information Technology and Computer Science(IJITCS), Vol.8, No.4, pp.86-95, 2016. DOI: 10.5815/ijitcs.2016.04.10