Volume 6, Issue 9, September 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Server Side Approach to Mitigating XSS Attacks Using Regular Expression and Specification Based Detection Technique

Christopher Njogu*, Michael Kimwele, George Okeyo
Department of Computing, Jomo Kenyatta University of Agriculture and Technology,
JKUAT, Kenya

Abstract— Cross Site Scripting (XSS) has become one of the most widespread vulnerabilities affecting web applications. XSS is an attack against web applications in which scripting code is injected into the output of an application that is then sent to a user's web browser and when the code is executed it can transfer sensitive data to the attacker. In this paper a framework was developed to detect any occurrences of XSS vectors within the web site users' inputs and neutralize the inputs using regular expressions and specification based detection techniques before data is passed to the web server for further processing. The evaluation was done through experiments and it has shown that the proposed framework is effective and efficient. The results indicate that the approach detects most of well-known XSS attacks.

Keywords— cross-site scripting, filtering, regular expressions, vulnerability, web

I. INTRODUCTION

Cross-site scripting is injection of unauthorized script code into a web page. A successful XSS attack embeds malicious script code in the untrusted HTML output which will in turn be executed on a victim's web browser to transfer sensitive information to the attackers.

Cross-site scripting is rated as third on the top ten web application vulnerability [1]. According to [10] which is involved in web application firewalls states that over 6% of top 1,000 web sites had a successful XSS attack and [9] accounts 160% of web application are still vulnerable to XSS. According to the research done by IBM X-Force [11] records in 2013 that attackers have successfully exploited vulnerable web applications with attacks like cross-site scripting and SQL injections.

According to [2] browser manufacturers focus more on correct interpretation and rendering of HTML codes and other scripting languages rather than trying to mitigate XSS attacks.

Despite the presence of many XSS mitigation approaches both client and server-side XSS attacks the discovery of XSS vulnerability is still widespread. Thus, there is a need to improve existing solutions or develop novel attack detection techniques. In this paper, we develop a server side approach that will detect and neutralize malicious code in the user input using regular expressions and specification based detection techniques before it is passed and processed by the web server.

The rest of the paper is organized as follows: Section II talks about the related WORK that has been done by other researchers, Section III notes down the types of XSS, Section IV presents our proposed mechanism with examples, explanations, and outcomes of various experiments based on that; and finally, Section V concludes the paper with future directions of research.

II. RELATED WORKS

XSS attack bypasses the security mechanisms laid down in the server-side and also mechanisms that are integrated in the modern day browsers. Attackers exploit the vulnerability and they are able to extract important and confidential data and use them in a variety of actions on behalf of the victim; they can monitor and send user actions to the attacker. This kind of activity done by the attacker compromise the security mechanisms laid down and can lead to great losses in case of a financial transaction [1].

There are three ways of protecting against XSS attacks while using the deployment location as the main classifier for existing solutions available in the academic and business world. Cross-site Scripting is essentially an input filtering failure. We can categorize the anti-XSS tools under: server-side, client-side and hybrid approaches.

ModSecurity is a Web Application Firewall that monitors inbound and outbound traffic to identify XSS attacks, SQL injection, and other vulnerabilities. ModSecurity and other web application firewalls which are server side approaches inspect the web applications at the HTTP layer and they report attacks rather than vulnerabilities. They lack the application context to differentiate between malicious activity and expected output. ModSecurity does not do sanitization [2], [12]. The disadvantages of the WAF is that they scan all incoming traffic regardless of the fact that the request might be a static page in which no form of processing will occur and thereby creating a notable amount of delays to the overall response time of the application. Secondly, once it detects malicious contents in the users input data, it drops the request rather than neutralizing and forwarding the data to the web server and finally web application administrators have to be constantly maintaining several rules which eventually will become complex and difficult to maintain over a long period of time.

Complementary to mitigating XSS on the server-side, there are several client-side solutions. A strictly client-side mechanism for detecting malicious XSS vectors its located on the web browser. Google developed XSS Auditor which is an extension in Chrome web browser. XSSAuditor looks at the script to be executed to determine if it is part of the web page script or from the response content. If from the response content, XSSAuditor blocks it. However in some cases the response might contain vectors that will alter the web page script and capitalize on it for its attacks. In such a case XSSAuditor fails [2].

Some solutions apply hybrid approaches which have incorporated both the client and the server side approaches.

Browser-Enforced Embedded Policies (BEEP) works by checking if the web server for a given execution of a script by the browser there is a cryptographic hash in the web server whitelist. If the hash is found the script is considered trusted and executed otherwise the script is not executed or rendered. The disadvantage of BEEP is that it does not restrict other types of content that may have potential XSS attack vectors. These are scripts which are run by CSS's and plugin contents e.g. Adobe Flash, PDF and Java applets [5]

A. Overview of XSS Attacks

These days a lot of personal and corporate information is stored online in various internet storage services (e.g. Dropbox, Google drive, cloud), in various internet portals and websites. More financial and e-commerce operations are performed using the internet than in the past. These are the circumstances that have led to an increased amount of crime in the internet. One of the most common and popular type of threats in the web application domain is the cross-site scripting (XSS) attack [1].

According to [8], an XSS attack involves three entities: a website with XSS vulnerability, a legitimate user of the website and finally the attacker. The goal of the attacker is to exploit the legitimate website user by stealing his or her credentials and sensitive information. Figure 1 below illustrates a typical XSS attack using a sequence diagram.

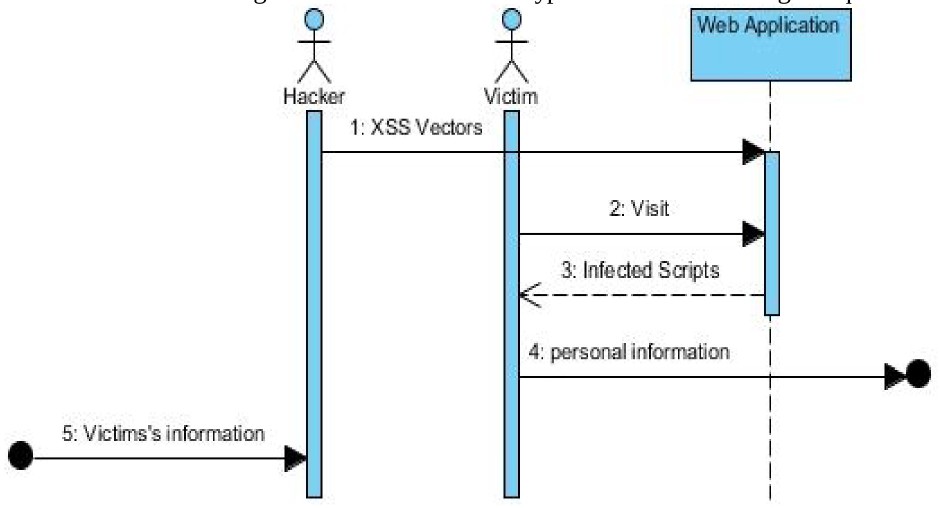


Figure 1: A simple XSS attack sequence

Figure 1 is explained as follows:

- A Hacker visits a web application and deposits XSS attack vectors.
- A Victim visits the infected web application using any web browser.
- The infected web application sends XSS vectors into the Victim's web browser, which then begins to exploit the Victim private data.
- The Victim's data is transmitted to the Hacker's server.
- The Hacker receives the Victim's data.

If the website is not XSS vulnerable, it would either discard the malicious payload, or detect the XSS vectors and neutralize them. However, if the site is XSS vulnerable, then, the user's browser would end up executing hacker/ attacker injected code in the page returned by the web site and send the users data or information to the hacker.

XSS attacks are divided into three different groups or categories based on the mode in which the attack was carried out. These are as follows: Non-persistent XSS attack, Persistent XSS attack and DOM-based XSS attack.

1) Non-persistent or reflected XSS attack

This kind of vulnerability is the most common type. The vulnerability holes show up when data provided by a web user is used immediately by the server-side scripts to generate a page of results for the user. If the user supplied data is not validated this will allow a client side code to be injected into the dynamic page. The attacker uses a get weak link inside the web application and they append a malicious content to the link of the victim site and propagate it using email or other messaging medium. The malicious content is not saved in the victims' database so this makes it hard to identify by just checking on the saved data [1].

2) Document Object Model Based attack

It is the same as non-persistent vulnerability with one important difference. The difference is that the processing of the vulnerability is done by a JavaScript library within the browser rather than on the server. If the malicious script is placed in the hash part of the URL, it is not even sent to the server, meaning that server-side protection techniques fail in that instance [1].

3) Persistent or stored XSS attack

This vulnerability type allows the most powerful kinds of attacks to be carried out. The vulnerability exists when data provided to the web application by the user is first stored persistently on the server database and its later displayed to users in a web page without the content being encoded using any HTML entities. This kind of attack is the most significant than the other two types since the attacker will only inject a malicious script just once and it could potentially hit a large number of other users. This kind of attack was used against MySpace website and Mcaffe website [1].

III. THE PROPOSED APPROACH: EXPERIMENTS AND OUTCOME

Defending against persistent cross-site scripting attacks input filtering and sanitization should be performed to the input so as to avoid the input from being executed by a browser and exploiting the users. To avoid xss attacks developers should always sanitize user's input before storing it in the database. The conceptual framework used to detect, mitigate and filter persistent xss vulnerabilities is designed based on the results of various experimentations.

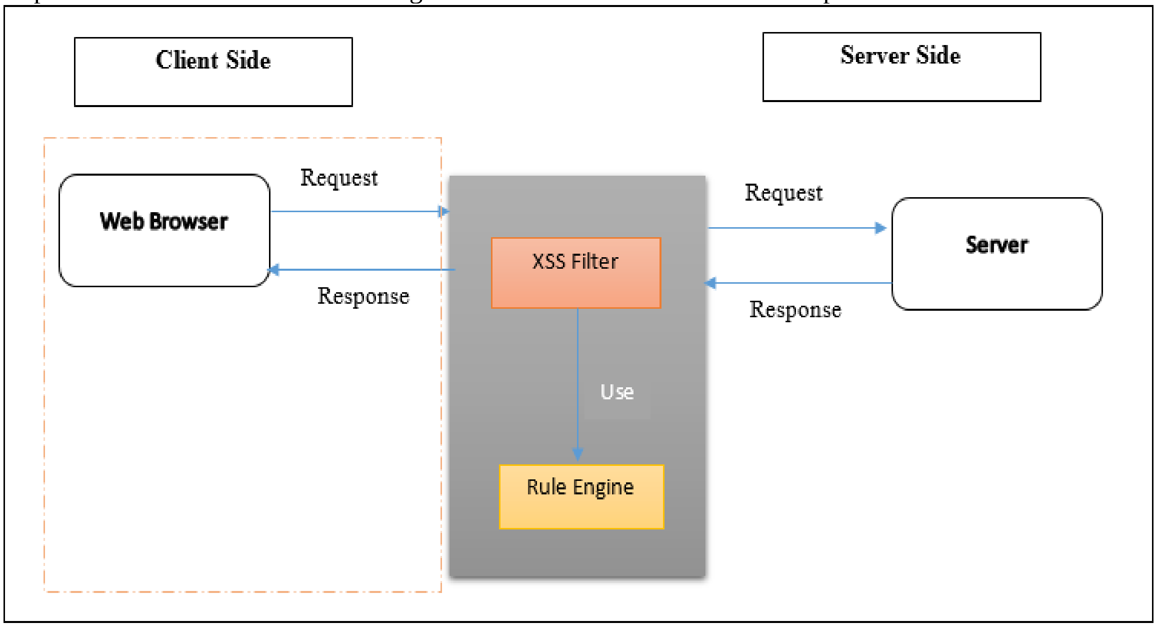
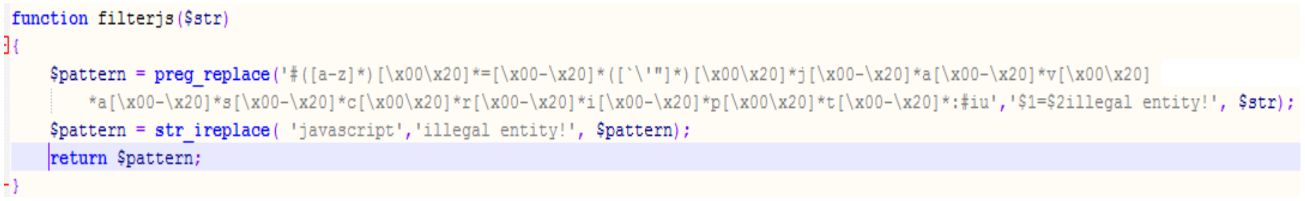


Figure 2: Conceptual Framework

Figure 2 shows the conceptual framework to prevent persistent XSS attacks. The figure depicts that user inputs are taken from the web browser as untrusted data which must go through filtering process to get a sanitized or clean data. The clean data is then passed for further processing and ultimately stored to the database to generate sanitized output once the data is called or pulled from the database.

4) JavaScript Filter

This filter module runs through the entire content that is passed to it, once it detects any JavaScript code that is malicious within the data it removes it. This filter contains a pattern that should not appear in the user input. If a string matches this particular pattern it is marked as invalid and it is replaced with "Illegal entity". The figure 3 below shows the function that was used

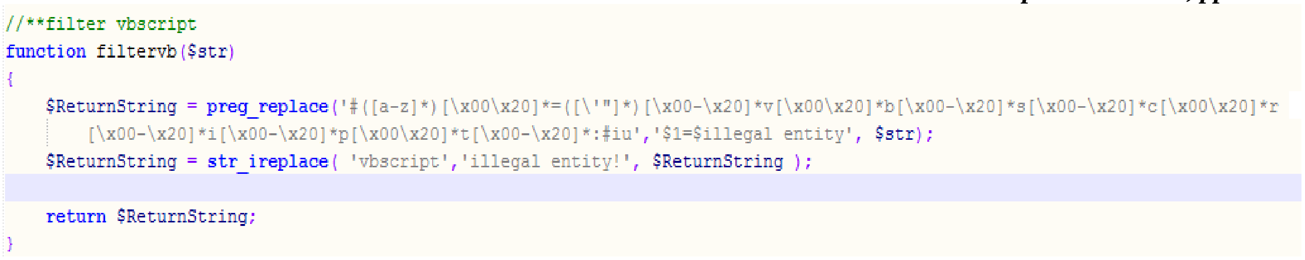


```
function filterjs($str)
{
    $pattern = preg_replace('#([a-z]*)[\x00\x20]*=[\x00-\x20]*([`\'"]*)[\x00\x20]*j[\x00-\x20]*a[\x00-\x20]*v[\x00\x20]
        *a[\x00-\x20]*s[\x00-\x20]*c[\x00\x20]*r[\x00-\x20]*i[\x00-\x20]*p[\x00\x20]*t[\x00-\x20]*:#iu','$1=$2illegal entity!', $str);
    $pattern = str_ireplace( 'javascript','illegal entity!', $pattern);
    return $pattern;
}
```

Figure 3: JavaScript Filter Method

5) VBScript Filter

The VBScript is a Microsoft scripting language that has been overlooked by some researchers who have worked previously on XSS attacks and mitigations. Microsoft operating system is widely used and it comes with a preinstalled Internet Explorer browser. It is therefore important to include VBScript sanitization. This module checks for any VBScript attack vectors and removes them. If any string matches with the pattern from the user input it is replaced by "illegal entity". The figure 4 below shows the function that was implemented.



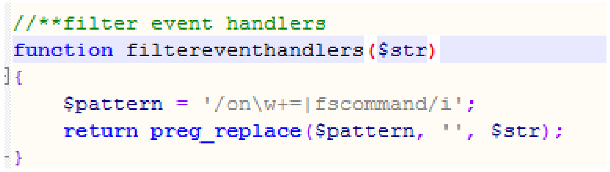
```
//**filter vbscript
function filtervb($str)
{
    $ReturnString = preg_replace('#([a-z]*)[\x00\x20]*=([\'"]*)[\x00-\x20]*v[\x00\x20]*b[\x00-\x20]*s[\x00-\x20]*c[\x00\x20]*r
        [\x00-\x20]*i[\x00-\x20]*p[\x00\x20]*t[\x00-\x20]*:#iu','$1=$illegal entity', $str);
    $ReturnString = str_ireplace( 'vbscript','illegal entity!', $ReturnString );

    return $ReturnString;
}
```

Figure 4: VBScript Filter Method

6) Event handlers filter

Event handlers are JavaScript codes that are not added inside the <script> tags. The event handlers are added inside the HTML tags and they execute JavaScript when an event happens for example pressing a button or submitting a form. The function assigned to the event handler is executed or run when an event occurs. Examples of event handlers: onMouseOver – its executed when a cursor moves over an object or area, onClick – its executed when someone clicks on a form, onError – its executed when an error occurs when either a document or image causes an error, onLoad – its executed after the window loads.

When an attacker injects XSS vector into a web application this module filters out the event handler in the input submitted by attacker by replacing it with null. Thus, the XSS payload would become invalid. The Figure 5 below shows how this filter can be used in a PHP class.



```
//**filter event handlers
function filtereventhandlers($str)
{
    $pattern = '/on\w+=|fscommand/i';
    return preg_replace($pattern, '', $str);
}
```

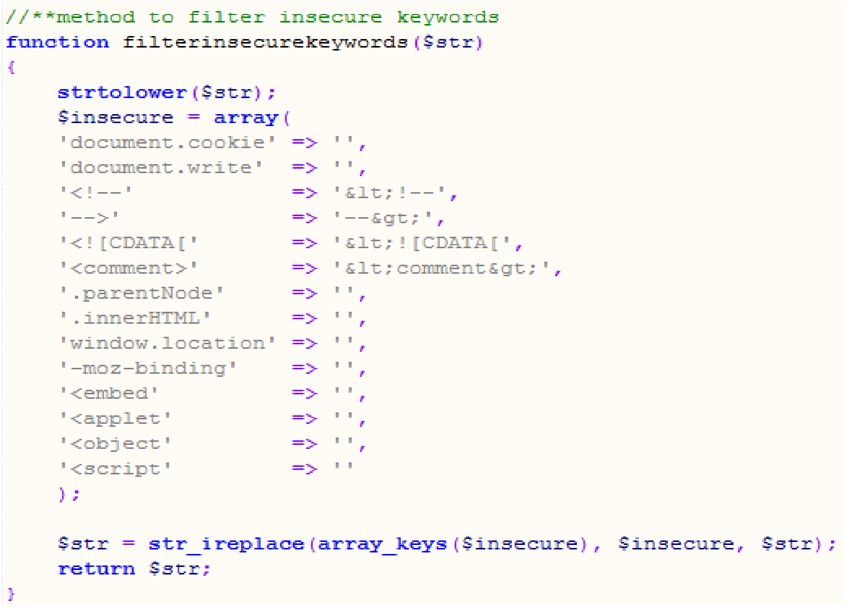
Figure 5: Event Handlers Filter Method

7) Insecure keywords filter

We have a list of keywords that are known as bad data and should be blocked by the filter is the string matches the pattern. The filter pattern should not match the user input, if it matches the keyword is replaced and the payload would become invalid. Some of the insecure keywords include:

- document.cookie
- document.write
- window.location
- innerHTML
- parentNode
- <applet
- <embed
- <script

When an attacker injects XSS vector into a web application this module filters out any insecure keyword from the input submitted by attacker by replacing it. The Figure 6 below shows how this filter can be used in a PHP class.



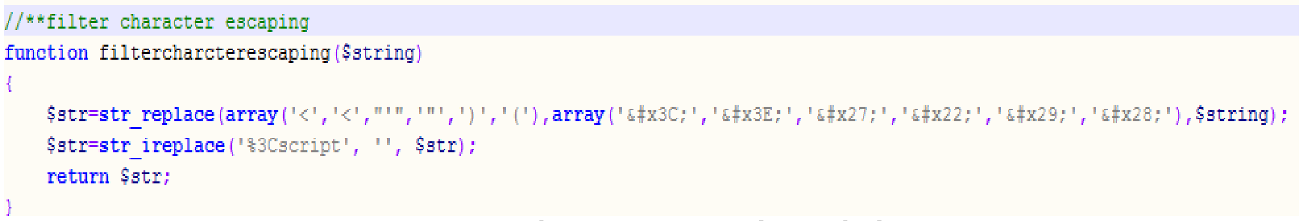
```
//**method to filter insecure keywords
function filterinsecurekeywords($str)
{
    strtolower($str);
    $insecure = array(
    'document.cookie' => '',
    'document.write'  => '',
    '<!--'            => '&lt;!--',
    '-->'             => '--&gt;',
    '<![CDATA['       => '&lt;![CDATA[',
    '<comment>'       => '&lt;comment&gt;',
    '.parentNode'     => '',
    '.innerHTML'      => '',
    'window.location' => '',
    '-moz-binding'    => '',
    '<embed'          => '',
    '<applet'         => '',
    '<object'         => '',
    '<script'         => ''
    );

    $str = str_ireplace(array_keys($insecure), $insecure, $str);
    return $str;
}
```

Figure 6: Insecure Keywords Filter Method

8) Character Escaping Filter

We need to ensure that all variable outputs in a webpage are encoded before they are returned to the end user. Preventing this types of XSS attack we substitute every special character that may be used in these attacks. We can escape the malicious characters by using the &# sequence followed by its character code. For example if a user inputs <script> alert("Attacked")</script> the special character '<', '>' this characters will be substituted with &#x3C and &#x3E respectively. This will be displayed in the webpage but the browser will not execute the script and the attack will be prevented.

The Figure 7 below shows how this filter can be used in a PHP class.



```
//**filter character escaping
function filtercharcterescaping($string)
{
    $str=str_replace(array('<','<',"'","'",')','('),array('&#x3C;','&#x3E;','&#x27;','&#x22;','&#x29;','&#x28;'),$string);
    $str=str_ireplace('%3Cscript', '', $str);
    return $str;
}
```

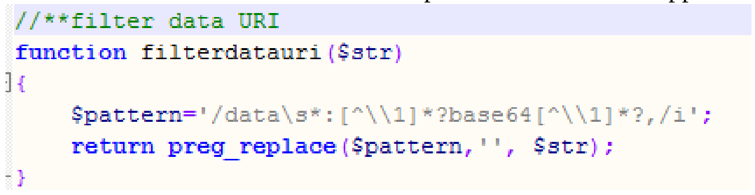
Figure 7: Character Escaping Filter Method

9) Data URI filter

A data URI is a self-contained link that contains document data and metadata encapsulated in the URI entirely but it does not include a filename. When presented with 'data:' URI with MIME (Multipurpose Internet Mail Extensions) types that trigger the save dialog like 'application/octet-stream', browser attempts to save the URI content as a file on the local file system.

The use of some keywords in the user input has been blacklisted in our web application - keywords like for instance, JavaScript, alert, script, round brackets, double quotes, and colon. Generally, <script> tag must be used to execute a JavaScript. Here, attacker cannot use the <script> tag because, the application validates user input against specific keywords. Hence, to execute a JavaScript, attacker tries using a data URI. In this attempt, attacker can now inject the following payload: "<object data="data:text/html;base64,PHNjcmlw dD5hbGVydCgiWFNTIik7PC9zY3JpcHQ+"></ object>"

The following figure 8 shows how data URI attack has been prevented in our web application.



```
//**filter data URI
function filterdatauri($str)
{
    $pattern='/data\s*:[^\\1]*?base64[^\\1]*?,/i';
    return preg_replace($pattern,'', $str);
}
```

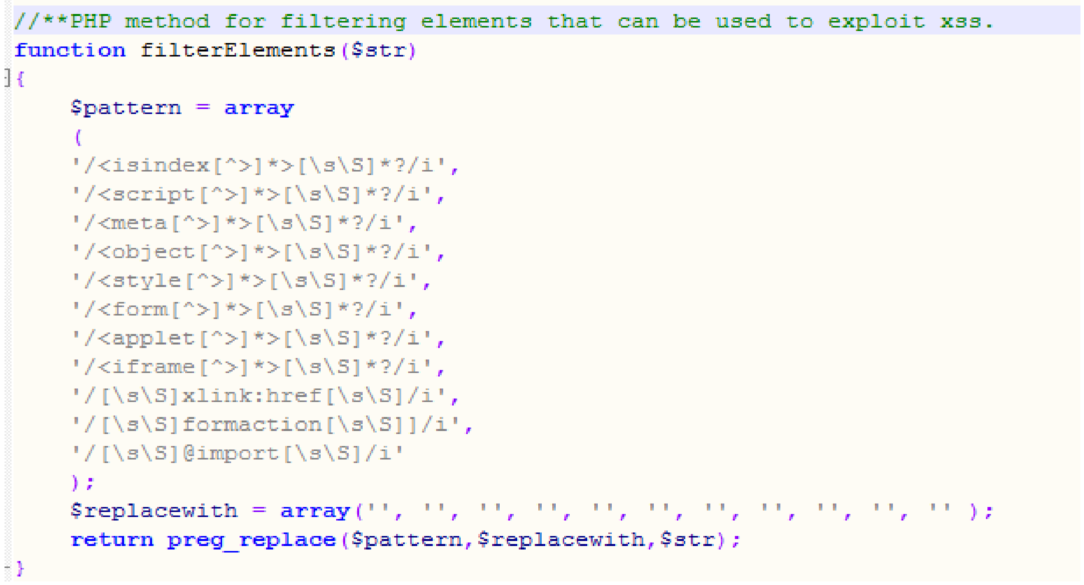
Figure 8: Data URI Filter Method

10) Filter HTML elements that maybe used as XSS payloads

HTML elements such as <isindex>, <meta>, <form>, <object>, <style>, <script>, etc, should be filtered out from user input to prevent our web application form XSS code injection. Here is an example of XSS payload filtered by this function:

<form><button formaction=javascript&colon;alert(1) >CLICKME

The following figure 9 shows how the HTML elements has been prevented in our web application.



```
//**PHP method for filtering elements that can be used to exploit xss.
function filterElements($str)
{
    $pattern = array
    (
    '/<isindex[^>]*>[\s\S]*?/i',
    '/<script[^>]*>[\s\S]*?/i',
    '/<meta[^>]*>[\s\S]*?/i',
    '/<object[^>]*>[\s\S]*?/i',
    '/<style[^>]*>[\s\S]*?/i',
    '/<form[^>]*>[\s\S]*?/i',
    '/<applet[^>]*>[\s\S]*?/i',
    '/<iframe[^>]*>[\s\S]*?/i',
    '/[\s\S]xlink:href[\s\S]/i',
    '/[\s\S]formaction[\s\S]]/i',
    '/[\s\S]@import[\s\S]/i'
    );
    $replacewith = array('', '', '', '', '', '', '', '', '', '', '' );
    return preg_replace($pattern,$replacewith,$str);
}
```

Figure 9: HTML Entities Filter Method

11) Outcome of Our Experiments

We have tested and evaluated a series of XSS attack scenarios. A collection of XSS Cheat Sheets from (OWASP, 2013). All of those were filtered and sanitized effectively. It has been proved that the proposed mechanism is effective enough in mitigating out various malicious XSS

IV. CONCLUSIONS AND FUTURE WORKS

A successful XSS attack can steal or manipulate victim's sessions and cookies, which the attacker may use to impersonate a legitimate user of a system. Hence, it is important to filter all web user input to secure any web application. Our main concern in this paper is that the applications should perform input and output filtering to achieve protection for its users. With the various experiments and investigation done, the proposed XSS attack prevention model has been found to be a very effective framework.

With time attackers devise new kinds of tricks and techniques and the attacks are getting more sophisticated. Given this fact, the most effective way to prevent and or defend against XSS attack is to code or build the web applications with security in mind and also to use proper escaping mechanisms. Given this case it is better never to trust the data coming from the user. Every bit of user data must be validated, filtered, and escaped before the output is given to the user browser.

As our future work, we plan to investigate thoroughly how our prevention framework can be extended and applied to non-persistent and other types of Cross-Site Scripting.

ACKNOWLEDGMENT

This work has been successful because of my Lord and savior Jesus Christ, my family, friends, supervisors and classmates who have given me continuous support.

REFERENCES

[1] Ceponis, J., Ceponiene, L., Venckauskas, A., & Mockus, D. (2013). Evaluation of Open Source Server-Side XSS Protection Solutions. In Information and Software Technologies (pp. 345-356). Springer Berlin Heidelberg.

[2] Benjamin, B. C., Oladeji, F. A., Okolie, C. C., Alakiri, H. O., & Olisa, O. (2013). S2MXS2: server side approach to mitigating XSS attacks using regular expression. Journal of Emerging Trends in Engineering and Applied Sciences,4(6), 875-882.

[3] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks. in Proceedings of the 2006 ACM symposium on Applied computing. 2006: ACM.

[4] Guha, S. Krishnamurthi, and T. Jim. Using static analysis for Ajax intrusion detection. in Proceedings of the 18th international conference on World wide web. 2009: ACM.

[5] M. V. Gundy and H. Chen. BEEP: Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. in WWW '07: Proceedings of the 16th international conference on World Wide Web. 2007: ACM.

[6] Van Gundy, M., & Chen, H. (2012). Noncespaces: Using randomization to defeat cross-site scripting attacks. computers & security, 31(4), 612-628.

[7] Cross-site scripting. (n.d.). What is XSS?. Retrieved July 1, 2014, from http://www.xssed.com/xssinfo

[8] Pelizzi, R., & Sekar, R. (2012, May). Protection, usability and improvements in reflected XSS filters. In ASIACCS (p. 5).

[9] Fire Host. (2012) Retrieved from http://www.firehost.com/company/newsroom/webapplication- attack-report-fourth-quarter-2012

[10] Trustwave's SpiderLabs, (2013) http://blog.spiderlabs.com/2013/08/the-web-is-vulnerable-xss-on-the-battlefront-part-1.html

[11] Gideoni, A. et al.(2014). IBM X-Force Threat Intelligence Quarterly 1Q 2014. New York, NY: IBM Corporation

[12] Ristic, I. (2010). ModSecurity Handbook. Feisty Duck