Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS) 4(6): 875-882
© Scholarlink Research Institute Journals, 2013 (ISSN: 2141-7016)
jeteas.scholarlinkresearch.org

S^2MXS^2: Server Side Approach to Mitigating XSS Attacks Using Regular Expression

[1]Benjamin B. C, [1]Oladeji F. A, [1]Okolie C. C, [2]Alakiri H. O, [1]Olisa O

[1]Computer Science,
University of Jos, Nigeria.
[2]Computer Technology,
Yaba College of Technology.

Corresponding Author: Benjamin B. C

Abstract

The most dreaded web application attack called Cross Site Scripting (XSS) attacks are still on the increase despite the research efforts being made. Usually, hackers upload XSS vectors into any vulnerable web site and wait for innocent victims who visit these sites. These victims are then attacked and exploited by the hacker's XSS vectors. Several existing techniques require technical adjustments on client side browsers and server side environment variables, while other techniques try to nullify the effects of XSS on users viewing dynamic contents. Mitigating XSS from server side can guarantee a better result than any other technique because users are not required to make any configurations on their browsers and no XSS vector will find its way to the client side. In this research, a framework was developed, which is based on pattern matching using regular expressions. This framework will detect any occurrence of XSS vectors within the data collected from users and nullify them before passing it over to the web application for further processing. This implies that the web application may not store or process any XSS vectors. This framework was implemented using a PHP object-oriented prototype model that can be easily integrated into existing web application. Evaluation of the framework was done using a web based PHP social network application and the results of our experiment shows that the proposed system is highly efficient in mitigating XSS attacks while maintaining a negligible runtime overhead on the web server. The purpose of this research is to design a simple XSS attack Filter framework that can be easily integrated into an existing web application which gives this research the potentials of generally reducing the rate of occurrences of XSS attacks on web applications.

Keywords: cross-site scripting, XSS, web application, PHP Filter, firewall, regular expression.

INTRODUCTION

Over the last decade, there has been an increase in the importance of securing web applications as software developers have found web based applications as an ideal approach to their software design. This has led to the increase of web-based enterprise applications that deal with sensitive financial, medical, and other sort of sensitive records. These web applications dynamically interacts with its users such that, users enter text inputs. Sometimes inputs are captured from URLs and sent in through header variables within web browsers. These input sources act like the access doors of the application. Hackers look forward for vulnerabilities through these sources and once they find any, they capitalize their attacks on it. Successful XSS attacks could mean compromising huge financial data, authentication information, loss of confidentiality and loss of company reputation that could lead to lawsuits (Alaa, Fadi & Mohammed, 2012) and (Martin, Bjorn & Joachim, 2008). Cross site scripting is rated the 3rd on the web application vulnerability top ten ranking (Hossain, 2011) and about 160% of web applications are still XSS vulnerable (FireHost, 2012). Most of these attacks can be mitigated if codes are written securely (Hossain, 2011). It is observed that most web application programmers concentrate more on how to add features and functionalities to their applications than they would concentrate on writing secured codes. On the other hand, some programmers and their clients are not knowledgeable about XSS attacks thus approach this attacks benignly. Recent researches show some steps that users need to follow to ensure that they are not victim of XSS attacks. Users are also warned not to disable scripting on their browsers, as this will knockoff several features on several web sites (Cisco, 2006). Also, users are not very knowledgeable about technicalities involved in mitigating XSS attacks, which implies that they can contribute little in the fight against XSS attacks (Alaa, Fadi & Mohammed, 2012).

Browser manufacturers focus more on correct interpretation and rendering of HTML codes and other scripting languages rather than trying to mitigate XSS attacks (Riccardo & Sekar, 2012). This is why client side approach remains a complementing technique while noting that XSS is a server side vulnerability (Riccardo & Sekar, 2012) and would be best handled from the server side.

In this paper, an XSS attack detection technique based on pattern matching using regular expressions was developed. This technique automatically detects XSS vectors and removes them before forwarding the users input to the web application for further processing. This report considered HTML, JavaScript, VBScript, XML namespace, Cascade style sheet (CSS) especially on IE browsers and Mozilla bind, which binds XML Binding Language (XBL) to DOM elements. This report also considered the fact that script obfuscations creates a big hole on several XSS filters and have strictly adopted UTF-8 character set. This is because UTF-8 character has more support starting from browsers down to database servers. This ensures the correct interpretation of users input.

This paper is divided into several sections as follows: Section one gives an introduction into XSS attacks and justifies the approach of the technique used by this research. Section two reviews related works while section three describes the XSS attack filter framework and presents the implementation of the framework. Section four discusses and evaluates the experimental results of the framework and section five concludes the paper.

LITERATURE REVIEW

The most used approaches in tackling XSS attack problems are usually seen as either server side or client side approaches. Most existing server side approaches are designed using the approach of a firewall called Web Application Firewall (WAF) (Jim, 2009). The flaws noted with this approach is that, it scans all incoming traffic regardless of the fact that the request might just 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. Another flaw is that 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. Another challenge is that the system administrators have to be constantly maintaining several rules, which eventually becomes complex and difficult to maintain over a long period of time. Other approaches to WAFs tend to use clustering techniques to perform some sort of learning and detection by observing and grouping data based on similar properties and anomalies. This will leave the web application with a full load of vectors since the system will pass through learning stage before detecting and blocking these vectors. Another approach that has been implemented on the server side is adding boundary to the dynamic contents in the HTTP response called the "boundary injection" (Hossain, 2011). These boundaries are combined HTML and JavaScript open and close comment tags surrounding the dynamic contents within the HTTP response of the web server. This technique will fail when an attacker adds the open and close comment tags before the XSS vectors. This will force the boundary to close just before the XSS vectors, allowing the XSS vectors to execute on the victims browser.

On the client side, browser manufacturers and programmers are applying a lot of techniques. Some examples of browser manufacturers' filters are, XSSAuditor by Google Chrome and IE8 Filter. These are being designed from a curative perspective since it is established that XSS attacks are server side attacks. The major flaws of XSSAuditor are that it is designed to mitigate basic XSS attacks (Riccardo & Sekar, 2012). This is true because 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. Also this technique is not application specific and cannot provide optimal solution for every individual web application as it can even create its own vulnerabilities like the IE8 filter (Eduardo & David, 2010). However, it is good to note that these techniques, if carefully implemented, can serve as complement technique to our proposed system.

Overview of XSS Attacks

XSS attackers have different aims for carrying out XSS attacks. This aims determines how they carry out the attack. Also the vulnerabilities imminent on any web application can determine how XSS attack will occur. Usually, hackers upload XSS vectors into a web site and wait for innocent victims who visits the site and make some sort of request for data. The web server sends the victim XSS infected scripts which carries out illegal exploits on the victim by sending victims data to the hacker in the background. Figure 1.0 below illustrates a typical XSS attack using a sequence diagram.

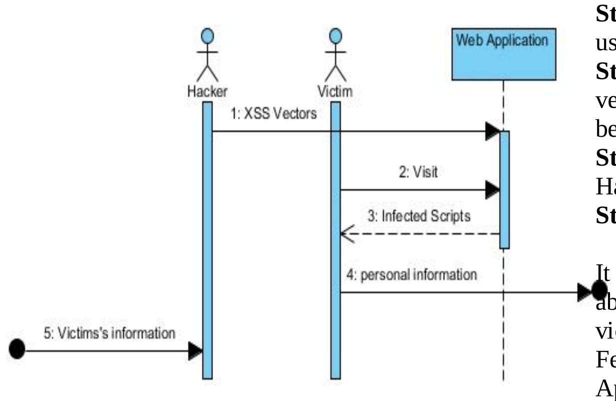

Figure 1.0 Sequence diagram of a simple XSS attack

Figure 1.0 is explained as follows: -

Step 1. A Hacker visits a web application and deposits XSS attack vectors.

Step 2. A Victim visits the infected web application using any web browser.

Step 3. The infected web application sends XSS vectors into the Victim's web browser, which then begins to exploit the Victim private data.

Step 4. The Victim's data is transmitted to the Hacker's server.

Step 5. The Hacker receives the Victim's data.

It is however, clear that if the web application was able to detect the XSS vectors and neutralized it, the victim would have been protected from this attack. Few sample XSS vectors contributed by Open Web Application Security Project, (OWASP, 2013), which are sourced from several XSS researchers and victims are shown in table 1.0 below. If any of the attack vector evades the application security, a pop up will appear on the browser showing the letters XSS.

Table 1.0 XSS vector samples

Name	XSS Vector	Description
XSS locator	';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//"; alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>	Explores possibilities for XSS vulnerabilities. JavaScript function fromCharCode converts ASCII numbers back the characters, then pops an alert showing the XSS
Image XSS	<IMG SRC="javascript:alert('XSS');"> or <IMG SRC=javascript:alert('XSS')> or <IMG SRC=JaVaScRiPt:alert('XSS')> or <IMG """><SCRIPT>alert("XSS")</SCRIPT>"> or <IMG SRC=# onmouseover="alert('xxs')">	HTML image tag to evade XSS filters. Attacks manipulate case, quote, malformed and default image source tags shown respectively
UTF-8 encoding	<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40; &#39;&#88;&#83;&#83;&#39;&#41;>	Evading filters using text format encoding. This might not work on all browsers.
List-style-image	<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS</b>	Embedding XSS vector into CSS

If any of the attack vectors in table 1.0 evades a web application security, a pop up will appear on the browser showing the letters XSS.

XSS attacks are basically classified into three, which is based on the mode in which the attack was carried out (Martin, Bjorn & Joachim, 2008). These are as follows: -

- **Reflective or Non-Persistent attack**

This is said to occur when a browser immediately outputs the XSS vectors as part of its HTTP response to a HTTP request. Typically, there might be no form of storage before the attack is carried out.

- **Stored XSS or Persistent attack**

This attack occurs when the vectors are injected into the web application and are stored for later exploits. This could attack victims as soon as they interact with the web application or it can be used to aid other forms of attacks.

- **Document Object Model based XSS attack**

These attacks are usually variation of reflective attacks. Most researchers will classify this type of attack as reflective attacks. However it is good to note that this attack most of the time does not originate from the attackers. It originates as a result

of careless coding which result to some sort of logical error, which ends up exploiting victims in some negative way.

The most used approaches in tackling XSS attack problems are usually seen as either server side or client side approaches. Most existing server side approaches are designed using the approach of a firewall called Web Application Firewall (WAF) (Jim, 2009). The flaws noted with this approach is that, it scans all incoming traffic regardless of the fact that the request might just 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. Another flaw is that 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. Another challenge is that the system administrators have to be constantly maintaining several rules, which eventually becomes complex and difficult to maintain over a long period of time. Other approaches to WAFs tend to use clustering techniques to perform some sort of learning and detection by observing and grouping data based on similar properties and anomalies. This will leave the web application with a full load of vectors since the system will pass through learning stage before detecting and blocking these vectors. Another approach that has been implemented on the server side is adding boundary to the dynamic contents in the HTTP response called the "boundary injection" (Hossain, 2011). These boundaries are combined HTML and JavaScript open and close comment tags surrounding the dynamic contents within the HTTP response of the web server. This technique will fail when an attacker adds the open and close comment tags before the XSS vectors. This will force the boundary to close just before the XSS vectors, allowing the XSS vectors to execute on the victims browser.

On the client side, browser manufacturers and programmers are applying a lot of techniques. Some examples of browser manufacturers' filters are, XSSAuditor by Google Chrome and IE8 Filter. These are being designed from a curative perspective since it is established that XSS attacks are server side attacks. The major flaws of XSSAuditor are that it is designed to mitigate basic XSS attacks (Riccardo & Sekar, 2012). This is true because 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. Also this technique is not application specific and cannot provide optimal solution for every individual web application as it can even create its own vulnerabilities like the IE8 filter (Eduardo & David, 2010). However, it is good to note that these techniques, if carefully implemented, can serve as complement technique to our proposed system.

Attack Detection Framework

This report considered the possible sources of input data into a web application and how it interacts with the objects within the web application. This is because the primary source of XSS attacks are the users' input data and the aim of this paper is to design a framework that can detect and mitigate XSS attacks. Having made close study and taken observations, this report considered adopting the Tainted Object Propagation Problem (TOPP) model (Benjamin, 2006) . This approach suggests the following: -

A. Source descriptors of the form (M, N, P), which represents the ways in which inputs can be collected from web application users. M represents the source method; N represents the number of parameters and P represents the access path, which will be applied to the parameter N in order to obtain the user's input.

B. Sink descriptors of the form (M, N, P), which will represent the unsafe ways in which data may be accepted and used within a web application. M represents the sink method, N represents the parameter and P represents the access path applied to the argument N.

C. Derivation descriptor, which takes the form (M, N_s, P_s, N_d, P_d), which focuses on how data propagates between objects within the web application. It has a derivation method, M, a source object giving by argument number N_s, with its access path P_s and a destination object giving by argument N_d with its access path $P_{d.}$ This implies that when method M is called, the object derived by applying P_d to $N_d$is derived from applying P_s to N_s.

D. Sanitization descriptor is of the form (SM, N_s, P_s, N_d, P_d), which specifies the methods that will sanitize data objects and stop the propagation of tainted data object within the web application. SM is representing a method acting on a source object giving by argument number N_s, with its access path P_s yielding an untainted data object and a destination object giving by argument N_d with its access path P_d.

Considering the TOPP model above, A suggests the general method in which data is generally accepted by web applications, B shows that A is unsafe and since B is unsafe, then N_s, P_s of C is unsafe since it may be derived from B and the outcome of the derivation, N_d, with its access path P_d are tainted i.e. infected. Therefore this report proposes a D, which is required to sanitize and produce and untainted $N_{s,}$ with an access path P_s.

Considering the TOPP model above, A suggests the general method in which data is generally accepted by web applications, B shows that A is unsafe and since B is unsafe, then then N_s, P_s of C is unsafe since it may be derived from B and the outcome of the derivation, N_d, with its access path P_d are tainted i.e. infected. Therefore this report proposes a D, which is

required to sanitize and produce and untainted $N_{s,}$ with an access path P_s.

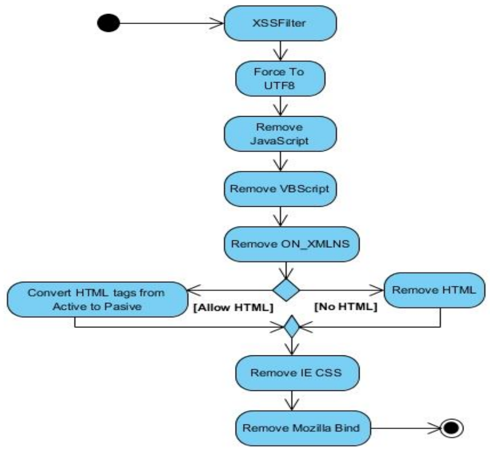

Figure 1.1 State diagram of the designed framework

A. Force To UTF8

This module simply checks the incoming data to see if it conforms to Unicode Transformation Format 8 (UTF-8). If it does not, it will be converted to UTF-8 so as to ensure that the vectors do not use other formats to obfuscate and evade the filter. It will also save processing overhead, as each module does not need to work using all other character formats. UTF-8 was considered because most of its compatibility with browsers down to database servers.

B. Remove JavaScript

This module runs through the entire content passed to it, once it detects any JavaScript within the data, it removes it. This was implemented using the regular expression below.

$ReturnString = 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!', $SourceString);
$ReturnString = str_ireplace('javascript', 'illegal entity!', $ReturnString);

C. Remove VBScript

The Microsoft scripting language VBScript has been overlooked by several researchers who have worked previously on XSS attacks. All Microsoft Windows operating system comes with IE browser preinstalled. This means that it is important to include the sanitization of VBScripts. This module checks for VBScript vectors and removes them. The following regular expression was used.

$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...', $SourceString);
$ReturnString = str_ireplace('vbscript', 'illegal entity!', $ReturnString);

D. Remove ON_XML_NS

XML Namespace (xmlns) is used to avoid conflict of element names especially when using XML languages. However, these namespaces within documents have been used to evade XSS filters. This module ensures that it looks into the embedded elements defined based on xmlns and check for vectors. If any is found, then it is removed. This can be implemented using the regular expression
$ReturnString = preg_replace('#(<[^>]+?[\x00-\x20"\'])(?:on|xmlns)[^>]*+>#iu', '$1>', $SourceString);

E. RemoveHTML

This module if enabled by the XSS filter configuration, will remove all html tags contained in the data. In this case, a No code situation is achieved. This can be implemented as shown below

$ReturnString = preg_replace('#</*(?:applet|b(?:ase|gsound|link)|embed|frame(?:set)?|i(?:frame|layer)|l(?:ayer|ink)|meta|object|s(?:cript|tyle)|title|xml)[^>]*+>#i', '', $ReturnString);

F. Convert HTML tags from Active to passive

This function is tries to convert any HTML special character to HTML entities so as to allow users input HTML code as part of their inputs. This is not advised since some client browsers can switch it back to active HTML tags. But however some applications might require this feature.

G. Remove IE CSS

Microsoft added an extra method called expression to IE browser, which can allow users execute codes within a style property like so: <span style="width: expression(alert('XSS'));"></span>. This module will check through to locate such vectors and remove them. This can be implemented using the code below.

```
$tentedString =
preg_replace('#(<[^>]+?)style[\x00-
\x20]*=[\x00-
\x20]*[`\"']*.*?expression[\x00-
\x20]*\([^>]*+>#i', '$1>',
$SourceString);

$ReturnString =
preg_replace('#(<[^>]+?)style[\x00-
\x20]*=[\x00-
\x20]*[`\"']*.*?behaviour[\x00-
\x20]*\([^>]*+>#i', '$1>',
$SourceString);

$ReturnString =
preg_replace('#(<[^>]+?)style[\x00-
\x20]*=[\x00-\x20]*[`\"']*.*?s[\x00-
\x20]*c[\x00-\x20]*r[\x00-\x20]*i[\x00-
\x20]*p[\x00-\x20]*t[\x00-
\x20]*:*[^>]*+>#iu', '$1>',
$SourceString);
```

H. Remove Mozilla Bind

Mozilla Firefox added a binding feature (-moz-binding) on their browsers. This binding ties a DOM element to an external XML Binding Language (XBL). This is a serious security issue because an attacker can dynamically control the DOM element on a victim's browser by injecting XSS vectors containing Mozilla binds. This module ensures that such binds are detected and removed from the users input data.

```
$ReturnString = preg_replace('#([a-
z]*)[\x00-\x20]*=([\"']*)[\x00-\x20]*-moz-
binding[\x00-\x20]*:#Uu',
'$1=$2nomozbinding...', $SourceString);
```

This framework was implemented using PHP programming language, and Object oriented programming approach (OOP). The test web application has a design of a basic social media web application with chatting features. The test web application was designed to have XSS loop holes and also implemented the JavaScript "document.write()" function. The Use Case diagram of the prototype test application is shown below in figure 1.2. This report deployed the test web application on an Ubuntu 12.04LTS operating system running apache web server version 2.2.21, MySQL server 5.5.9 and PHP version 5.3.6.

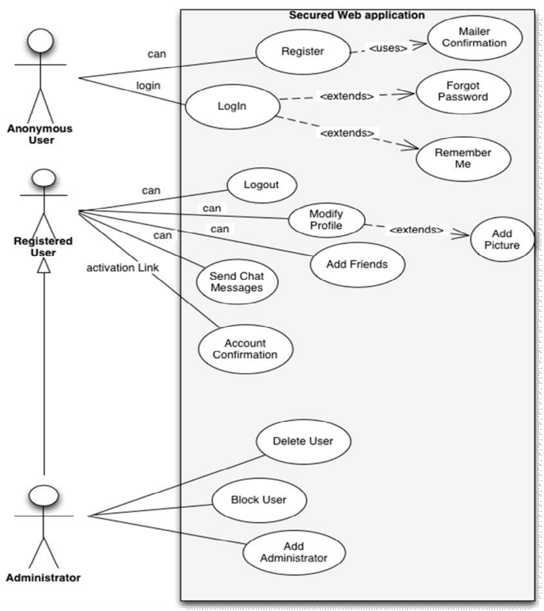

Figure 1.2 Use case diagram showing the test application of the framework

DISCUSSIONS OF FINDINGS

In order to evaluate the new system, this report identified and used some XSS vectors from XSS Cheat Sheet (OWASP, 2013), which are widely used to evaluate XSS.

However these are just a few vectors since it is not possible to capture and test all vectors within this report.

A. XSS locator

Input

';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//";
alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//--
></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>

Output

';alert(String.fromCharCode(88,83,83))//';alert(String.fromCharCode(88,83,83))//"
alert(String.fromCharCode(88,83,83))//"alert(String.fromCharCode(88,83,83&

Input

'';!--"<XSS>=&{()}

Output/Result

'';!--"&lt;XSS&gt;=&{()}

This vector locates and informs the attack that XSS attack is possible. When this vector is injected, and in most cases where a script is vulnerable with no special XSS vector requirements the word "XSS" will popup. However, in the case of the newly designed system, there was no popup and the corresponding output shows that the vector was nullified.

B. Image XSS using JavaScript directive

Input

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

Output/Result

&lt;IMG SRC="illegal entity!alert('XSS')"&gt;

This vector tries to inject itself into the system while hiding inside a HTML image tag. However, Internet Explorer 7.0 or greater has by default disabled this, other browsers still support it. This vector still did not escape through as the WebFilter module detected it and neutralized it by translating it to "&lt;IMG SRC="illegal entity!alert('XSS')"&gt;".

C. Grave accent obfuscation

Input

<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>

Output/Result

&lt;IMG SRC=`illegal entity!alert("RSnake says, 'XSS'")`&gt;

A lot of web application developers are not aware of the havoc that could be caused by the grave accent character. The grave accent mark is used to hide XSS vectors and will then be correctly interpreted by the web browser, which may results to a fierce XSS attack. However the Attack Detection Framework was able to detect and nullify the attack as shown in the out

D. Malformed anchor tags

Input

<aonmouseover="alert(document.cookie)" >xxs link</a>

Output/Result

&lt;a&gt;xxslink&lt;/a&gt;

The vectors used are malformed (ill written or wrong syntax) HTML tags. This is because most algorithms will be looking out for correct HTML syntax while trying to detect the injected vector. A malformed HTML syntax will end up destabilizing the algorithm. However in the case of the newly designed framework, it was seen that the algorithm was able to detect and neutralize the XSS vector.

E. UTF-8 Unicode encoding

Input

<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>

Output

&lt;IMG SRC=illegal entity!alert('XSS')&gt;

There are several Unicode transformation formats namely UTF-1, UTF-7, UTF-8, UTF-16 and UTF-31. The format that is widely used is the UTF-8 this is both cutting across form the browser to the web server and down to the database server. Therefore, this research chooses to use UTF-8 as a standard. The filtering algorithm will always convert all data to UTF-8 standard. However vectors with Unicode encoding will end up being converted to UTF-8 and then the filter will correctly interpret it and decode any XSS vector embedded inside it.This explains the output,

&lt;IMG SRC=illegal entity!alert('XSS')&gt;

CONCLUSION

In order to defend XSS attacks, this report designed and implemented S^2MXS^2as a server side XSS attack detection and prevention system. The approach uses regular expression to detect and remove XSS vectors coming into the system from user input data, in the form of textual inputs. This report followed an Object Oriented programming (OOP) design approach, which makes it easy to be integrated into existing application. It was implemented using PHP web programming language as this report considered the fact that majority of web applications were designed using PHP. The result showed that the approach suffered from zero false negative. However, our design does not tolerate any form of programming codes and scripts as part of user inputs.

LIMITATIONS OF THE STUDY

The limitation of this study is that the framework was developed using PHP programming language and MySQL database server. This means that implementing this framework using other programming languages may be very challenging as the output of regular expression functions may from one programing language to the other. Another limitation of this study is that the framework does not tolerate any form of codes as part of user inputs. Another limitation is that the framework does not have any form of learning algorithm and does not build and manage an attack vector database.

RECOMMENDATIONS FOR FUTURE WORKS

Further research works might include the ability to tolerate codes and scripts as part of users inputs. This research may also consider adding learning algorithm into the design in the future.

REFERENCE

Alaa, H, A. and Fadi, A, O, N. and Mohammed, S, A, W. (2012) 'Web Application Security of Money Transfer Systems' Journal of Emerging Trends in Computing and Information Sciences, Vol. 3, No. 3, ISSN 2079-8407.

Benjamin, L. 2006. Improving software security with precise static and runtime analysis. PhD Thesis, Stanford University

Cisco. (2006) Understanding Cross-Site Script Understanding Cross-Site Scripting (XSS) Threat Vectors Retrieved from <http://www.cisco.com/en/US/products/cmb/cisco-amb-20060922-understanding-xss.html> [Accessed May 2013]

Eduardo, V, N. and David, L. (2010) Universal xss via ie8's xss _lters. Black Hat Europe 2010.

FireHost. (2012) Retrieved from <http://www.firehost.com/company/newsroom/web-application-attack-report-fourth-quarter-2012>

Hossain, S. (2011) S2XS2: A Server Side Approach to Automatically Detect XSS Attacks 2011 Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing

Jim, B. (March 2009) Web Application Firewalls: Defense in Depth for Your Web Infrastructure

Martin, J. Bjorn, E. and Joachim, P. (2008) XSSDS: Server-side Detection of Cross-site Scripting Attacks 2008 Annual Computer Security Applications Conference

Riccardo P and Sekar .R (2012) Protection, Usability and Improvements in Reflected XSS Filters ACM 978-1-4503-1303.

The Open Web Application Security Project (OWASP) Cross-site Scripting (XSS) Retrieved from https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet