CHAPTER 4

PERFORMANCE EVALUATION OF XSS PREVENTOR IN ASSOCIATION TO ENCRYPTORS IN XSS DETECTION

4.1 INTRODUCTION

In 2016, XSS is a critical issue that occur even in internet companies like Yahoo, Google which encountered malicious hits. Yahoo was hit up with a stored XSS type and Facebook was leveraged via images that are malicious to a redirected user account. Cross-site scripting attack fools the security managers of the web application company when they use firewall and third party libraries to safeguard during the encoding process of the input of the user and the deployment phases. The protection methods do not guarantee 100% free XSS but a care could be provided using secure input handling using encoding and validation techniques.

Kerschbaum (2007) presented an algorithm that acts as a gateway which receives data in a form of query URL or posting the requested data which handles forgery attacks obtained through end user requests. Kirda et al (2006) created a method that contains manual and automatically generated rules for detecting stored and reflected XSS using Noxes, a personal web firewall. This method expects a few user interactions and proper customization in order to preserve information leakage.

Shar et al. (2012) proposed a server side tool named SaferXSS that uses both detection and prevention methods since detection processes are not purely successful until a prevention method is employed.
Encryption techniques turn useful information to a meaningless one such that the content is of no use until it could be transformed for future use by a cipher key. Associating these sorts of techniques to a prevention method will be advantageous where the hackers can be deviated to spend some time in the decryption process of the hashing algorithms.

In this chapter, an API named ESAPI (source: www.owasp.org) is adopted to fit in the ‘XSS Handler phase’. The algorithm requires inspiration from both the web page and the browser for the effective prevention of the XSS. A customised browser is specifically updated for the usage and to correct XSS. An XSS detector designed in association with encryptions achieves high prognostic accurateness and interesting scenarios. Promising results are found during experiments conducted with the five data sets obtained from XSS Archive.

4.2 PROBLEM STATEMENT

The objective of the proposed work is to prevent XSS by employing a method which eliminates the XSS vulnerabilities present and provides a benign page for the end user at the browser view. The input and the expected output for the research problem is a web page.

4.3 SELECTION OF ENCRYPTORS FOR XSS DETECTION

The key concept of encryption algorithms is to provide confidentiality to the data stored on computers or servers. These algorithms assure the following key features namely, authentication, integrity and non-repudiation. The role of hash function in the algorithm is highly appreciable. These functions generate an output named HV or MD. Novel functionalities of the hash function include the following:
Length of the output is less when compared to the given input.

The computation speed is high for any given input.

View of the output cannot reveal the possible input.

A change in output is reflected if an alteration is done in the input.

The strength of the encryption process is directly proportional to the key size, where if the key size increases a complexity grows too. Rivest (1992) replaced a hash function and named as MD5 algorithm which processes any length input towards a fixed length output containing 32 hex characters.

A family of hash functions can be grouped as Secure Hash Algorithm (SHA). SHA-1 resembles MD5 algorithm and SHA-2 has two similar hash functions with SHA-256 with 256 words and SHA-512 with 64 words.

From the constraint analysis of the hash functions, SHA-256 and MD5 are found suitable for ‘Encryption’ in XSS Detection phase thereby playing the detector role.

4.4 FRAMEWORK FOR HANDLING XSS ATTACKS USING ENCRYPTORS

For a given web page, the proposed method prevents XSS attacks, if any maliciousness is found, using two phases namely the XSS Detector phase and the XSS Handler phase. The detailed workflow of the proposed method is outlined in Figure 4.1.
Figure 4.1 Workflow of the XSS Preventor using Encryptors
4.4.1 XSS Detector Phase

The XSS Detection includes an eXtensible Markup Language (XML) which acts as an internal database. The database holds varied details including HV or MD based on the encryption algorithm used and acts as a repository to hold the contents. The XML tree structure is shown in Figure 4.2.

Figure 4.2 XML Tree of the Database

When the URL of the web page is provided as input, the source is extracted and the HV or MD value is obtained using the encryptor algorithm. The value is then compared with the XML repository. If a match is found, the web page is retrieved from the repository, else the control is moved towards the XSS Handler phase. The brief procedure involved for string comparison (SC) using encryptors is given as algorithm in Table 4.1.
Table 4.1 Algorithm for Comparison using Encryptors

COMPARISON (Input, Output)

// Input: Uniform Resource Locator (URL) ; Output: HTML Web Page

// D: Database consists of Message Digest (MD) value, file content in
//@ XML file of secured web pages

(1) 
(2) Filecontent ← Extract the source code in Input
(3) Flag ← 0
(4) MD ← Compute Hash Value or MessageDigest5 for the Filecontent
(5) for i=1 to total number of files in D and step up by 1
(6) 
(7) If (Input = URL in D and MD = (value(MD[i]) in D)
(8) 
(9) Flag ← 1
(10) GOTO A1
(11) }
(12) Else
(13) 
(14) Flag ← 0
(15) } 
(16) } 
(17) A1: If (Flag =1) 
(18) Output ← Display the content from database 
(19) Else 
(20) Output ← Call the Prevention Task with Input 
(21) }

4.4.2 XSS Handler Phase

The XSS Handler process corrects certain XSS attacks with the help of prevention rules. OWASP specifies the user input as data and provides suggestions for possible avoidance of XSS vulnerabilities and clinches that there is no harm in using the ESAPI. Escaping the referenced, untrusted data even if the data is not vulnerable is being performed with ESAPI. The approach proposes rules ranging from Rule#1 to Rule #5 through the API’s. The escaping rules are presented in Table 4.2.
Table 4.2 Escaping Rules for XSS Prevention

<table>
<thead>
<tr>
<th>Rule No.</th>
<th>Untrusted Data Reference</th>
<th>Source of XSS</th>
<th>Invoked Escaping API Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In HTML element</td>
<td>HTML Elements like div, body</td>
<td>ESAPI.encoder().encodeForHTML()</td>
</tr>
<tr>
<td>2</td>
<td>As a value of a HTML Attribute</td>
<td>HTML Attributes likeName=&quot;Untrusted Data&quot;</td>
<td>ESAPI.encoder().encodeForHTMLAttribute()</td>
</tr>
<tr>
<td>3</td>
<td>As a quoted data value in JavaScript or an Event Handler</td>
<td>Java script or event handlers like onload(), onclick()</td>
<td>ESAPI.encoder().encodeForJavaScript()</td>
</tr>
<tr>
<td>4</td>
<td>As a value of property in a Cascaded Style Sheet(CSS)</td>
<td>Style Tag</td>
<td>ESAPI.encoder().encodeForCSS()</td>
</tr>
<tr>
<td>5</td>
<td>As a HTTP GET parameter value in an URL</td>
<td>Anchor Tag</td>
<td>ESAPI.encoder().encodeForURL()</td>
</tr>
</tbody>
</table>

When an input is produced to this phase, the web page is parsed to build a hierarchical tree. Every node in the tree is checked for element, attribute, style script, and URL tag and they undergo SC process and the ESAPI rules are invoked. After the procedure gets over, the altered document is passed to the end user and the changes are updated to the XML database. The procedure involved for handling the prevention is given as algorithm in Table 4.3.
### Table 4.3 Algorithm for Handling XSS Prevention

PREVENTION(Input)

// Input: Uniform Resource Locator (URL); Output: HTML Web Page

// D: Database consists of Message Digest values of secured web pages

1. {  
2. Filecontent ← Extract the source code of the Input  
3. Parse the Filecontent to a Tree T  
4. n ← Count the number of nodes  
5. Construct a new tree P with a root node as of Filecontent  
6. for i = 1 to n and step up by 1  
7. {  
8. For every (T[i]) //check every tag  
9. {  
10. if “element” : p[i] ← Perform ESAPI HTML Element process for T[i]  
11. if “attribute” : p[i] ← Perform ESAPI HTML Attribute process for T[i]  
12. if “script” : p[i] ← Perform ESAPI Script process for T[i]  
13. if “style” : p[i] ← Perform ESAPI Style process for T[i]  
14. if “url” : p[i] ← Perform ESAPI URL process for T[i]  
15. otherwise : p[i] ← T[i]  
16. }  
17. }  
18. Generate HTML Document GD by merging all P[i]’s  
19. Compute the new hash value of GD  
20. Store the GD and its details in D
4.5 EXPERIMENTS AND RESULTS

4.5.1 Browser

A Java enabled browser is capable for the conduct of the work. The browser is able to process the given URL and display the benign content as output. Both XSSed and Non XSSed data can be given as input to the browser.

4.5.2 Dataset

Experiments are done for about 500 samples which include 250 XSS and 250 Non XSSed data as discussed in section 1.4 and Section 3.4.1. The pre-processed datasets are then redirected to the detector phase.

4.5.3 Experimental Setup

The web page given to the browser acts as an input to the encryption algorithm namely SHA-256, MD5 which computes the HV and MD value respectively. The computed value is compared with the value in the XML database. If both the values are same, the altered content available in the database is executed by the browser else application of prevention rules and updating the database along with the new MD or HV values takes place. The contents of the XML database is depicted in a tabular form excluding the fields, namely, actual content and altered content after applying MD5 algorithm String Comparison (MD5-SC) in Table 4.4.
Table 4.4 Contents of the XML Database with MD5-SC

<table>
<thead>
<tr>
<th>URL Name</th>
<th>Message Digest or Hash Value</th>
<th>Lines of Code (LOC)</th>
<th>HTML Element Count</th>
<th>HTML Attribute Count</th>
<th>URL Count</th>
<th>Java Script Count</th>
<th>CSS Count</th>
<th>Time (in Secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://in.hotels.com/">http://in.hotels.com/</a></td>
<td>6dfbc33c359f58443f849d29098ac3a0</td>
<td>1606</td>
<td>845</td>
<td>1177</td>
<td>220</td>
<td>78</td>
<td>0</td>
<td>132</td>
</tr>
<tr>
<td><a href="http://suven.hubpages.com/hub/Importance-of-Food">http://suven.hubpages.com/hub/Importance-of-Food</a></td>
<td>cabf07ce6242b09e0d2c6aacc1ac202f</td>
<td>284</td>
<td>533</td>
<td>618</td>
<td>45</td>
<td>35</td>
<td>0</td>
<td>74</td>
</tr>
<tr>
<td><a href="http://www.webmd.com/diet/">http://www.webmd.com/diet/</a></td>
<td>3ad465b83c530c23a4aa6f82702acf18</td>
<td>2455</td>
<td>1435</td>
<td>2352</td>
<td>425</td>
<td>54</td>
<td>3</td>
<td>312</td>
</tr>
<tr>
<td><a href="http://www.enjoy.net.mm/">http://www.enjoy.net.mm/</a></td>
<td>f8e049f09bad6f48f565e45a4f2e8f41</td>
<td>735</td>
<td>735</td>
<td>917</td>
<td>174</td>
<td>10</td>
<td>2</td>
<td>122</td>
</tr>
<tr>
<td><a href="http://www.cdc.gov/heartdisease/">http://www.cdc.gov/heartdisease/</a></td>
<td>fb8a6fa47322ebbcc946a71d73148d88</td>
<td>337</td>
<td>570</td>
<td>672</td>
<td>202</td>
<td>26</td>
<td>3</td>
<td>74</td>
</tr>
</tbody>
</table>
The XML database content is presented as a table without the page source content after applying SHA 256 algorithm for String Comparison (SHA-SC) in Table 4.5.

<table>
<thead>
<tr>
<th>URL Name</th>
<th>SHA1 Hash Value</th>
<th>Lines of Code (LOC)</th>
<th>HTML Element Count</th>
<th>HTML Attribute Count</th>
<th>URL Count</th>
<th>JavaScript Count</th>
<th>CSS Count</th>
<th>Time (in Secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://indianfood.about.com/">http://indianfood.about.com/</a></td>
<td>78a1fe6662c170acf68fd84b5d274171</td>
<td>327</td>
<td>535</td>
<td>751</td>
<td>185</td>
<td>29</td>
<td>2</td>
<td>68</td>
</tr>
<tr>
<td><a href="http://en.wikipedia.org/wiki/calorie">http://en.wikipedia.org/wiki/calorie</a></td>
<td>0db8eb1a445bef111392dd31f443aeff3</td>
<td>416</td>
<td>815</td>
<td>1052</td>
<td>301</td>
<td>9</td>
<td>2</td>
<td>87</td>
</tr>
</tbody>
</table>

Table 4.5 Contents of the XML Database with SHA-SC
<table>
<thead>
<tr>
<th>Hash Value</th>
<th>Sequence</th>
<th>Length</th>
<th>Total</th>
<th>Level</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a628c5729735e65e9fc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="https://www.ibm.com/developerworks/library/wa-secxss/">https://www.ibm.com/developerworks/library/wa-secxss/</a></td>
<td>a23ca3f3528207747a9ab036d96dc69b900d95a3c71cd59d39460af8152d06d4</td>
<td>785</td>
<td>844</td>
<td>908</td>
<td>174</td>
</tr>
<tr>
<td><a href="http://www.enjoy.net.mm/">http://www.enjoy.net.mm/</a></td>
<td>db95b2a5d61712817941783874e412422974ae0238084e7313f4deabe09fca0f</td>
<td>735</td>
<td>735</td>
<td>917</td>
<td>174</td>
</tr>
<tr>
<td><a href="http://www.cdc.gov/heartdisease/">http://www.cdc.gov/heartdisease/</a></td>
<td>9e095c0dd6e7607c7b1518611756cfec117b6aa0f37f156eb4cfdd7b461acb0e</td>
<td>337</td>
<td>570</td>
<td>672</td>
<td>202</td>
</tr>
<tr>
<td><a href="http://indianfood.about.com/">http://indianfood.about.com/</a></td>
<td>738ac55c50734074099da9deff8c6ff3eb0af6b75ae64099ec077df11e3cc2e</td>
<td>327</td>
<td>535</td>
<td>751</td>
<td>185</td>
</tr>
<tr>
<td><a href="http://en.wikipedia.org/wiki/calorie">http://en.wikipedia.org/wiki/calorie</a></td>
<td>8289e3133841de3ba978f9fe9e75499fd852432c3d290c74012b4f1f5dfc362e</td>
<td>416</td>
<td>815</td>
<td>1052</td>
<td>301</td>
</tr>
</tbody>
</table>

If the hash value is unique, the content is presented to the browser else generation of a hierarchical tree for the document is done. ESAPI rules
are applied and the altered source code will be of the form as shown in Figure 4.3.
Figure 4.3 Web Page with Prevention Rules

The altered code will be for browser utilization and updated content in the XML will be of the form shown in Table 4.6.

Table 4.6 Updated Content included into XML Database

<table>
<thead>
<tr>
<th>URL Name</th>
<th>Message Digest</th>
<th>Lines of Code (LOC)</th>
<th>HTML Element Count</th>
<th>HTML Attribute Count</th>
<th>URL Count</th>
<th>Java Script Count</th>
<th>CSS Count</th>
<th>Time (in Secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.paypal.com/de/cgi-bin/searchwide-search?cmd=_siti">https://www.paypal.com/de/cgi-bin/searchwide-search?cmd=_siti</a>...</td>
<td>d97fe6433fd64f4348e5e250b8a09c7</td>
<td>90</td>
<td>179</td>
<td>275</td>
<td>37</td>
<td>17</td>
<td>3</td>
<td>20</td>
</tr>
</tbody>
</table>

4.5.4 Results and Discussion

The encrypted pages using MD5 and SHA256 are taken for XSS detection and prevention. In this work, any modification in the web page will
result in a new digest value or a hash value depending on the algorithm selected. When a web page is requested by the browser, the content is mapped with the cipher value of the detection phase. If any change in the HV or MD, the XSS Handler phase is initiated.

The functionality of XSS Handler is tested for the 500 web pages that are encrypted using MD5-SC and SHA-SC and the measures like DR, FDR and FS are used. The comparative performance of the encryptors used for SC is shown in Table 4.7.

### Table 4.7 Performance Comparison of Encryptors

<table>
<thead>
<tr>
<th>Test Domain</th>
<th>DR</th>
<th>FDR</th>
<th>FS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MD5 - SC</td>
<td>SHA-SC</td>
<td>MD5 - SC</td>
</tr>
<tr>
<td>Events</td>
<td>96.30</td>
<td>96.30</td>
<td>1.27</td>
</tr>
<tr>
<td>classifieds</td>
<td>97.56</td>
<td>97.56</td>
<td>0</td>
</tr>
<tr>
<td>Roomba</td>
<td>98.04</td>
<td>98.04</td>
<td>0.66</td>
</tr>
<tr>
<td>Personal Blog</td>
<td>98.81</td>
<td>96.43</td>
<td>0</td>
</tr>
<tr>
<td>Jgossip</td>
<td>99.36</td>
<td>99.36</td>
<td>0</td>
</tr>
</tbody>
</table>

From the result obtained by the DR for various criteria both MD5-SC and SHA-SC has detected at the same rate for 4 domains whereas in the Personal Blog domain the rate of MD5-SC is higher than SHA-SC. The FDR is negligible with a value null in many cases for both of the algorithms. The FS score for MD5-SC is high for the classifieds, personal blog and Jgossip domains, whereas SHA-SC scores high for events. The FS is equal in MD5-SC and SHA-SC for Roomba domain.
On comparing the MD5 and SHA 256 algorithms, both are hash functions with a varied size of output string. It is proved that SHA 256 is stronger in security than MD5 whereas MD5 is quicker than SHA 256. Moreover the functionalities of these algorithms differ invariably depending on the nature of the hardware and software used.

From the analysis of the result, it is found that MD5-SC and SHA-SC do not show a comparable difference in their performances. The usage of the encryptors does not impact the efficiency of the XSS detector and it is used only to speed up the visualization of the content in the browser.

4.6 SUMMARY

In this chapter, XSS Handler phase had been devised with eminent encryption method that is valuable for string comparison. From the analysis, it was observed that comparison using the encryptors has no much significant difference except time in any of the data used. This clearly depicts than any one encryptor amongst the two can be used for comparison. From the experimentation, it was inferred that the ESAPI could handle the web page thereby preventing XSS attacks using MD5-SC and SHA-SC with an average detection rate of 98.01% and 97.53% respectively. This prevention method could be wisely used at the client end without any further processing.