Indian Journal of Science and Technology, *Vol 8(30), DOI: 10.17485/ijst/2015/v8i30/87182, November 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Removing Cross-Site Scripting Vulnerabilities from Web Applications using the OWASP ESAPI Security Guidelines

Isatou Hydara*, Abu Bakar Md Sultan, Hazura Zulzalil and Novia Admodisastro

Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang - 43400 UPM, Selangor, Malaysia; ishahydara@gmail.com

Abstract

Software security vulnerabilities are present in many web applications and have led to many successful attacks on a daily basis. These attacks, including cross-site scripting, have caused damages for both web site owners and users. Cross-site scripting vulnerabilities are easy to exploit but difficult to eliminate. Most solutions provided only focus on preventing attacks or detecting the vulnerabilities. Very few research works have addressed eliminating these vulnerabilities from the web applications source codes. In this paper, we propose an approach to remove cross-site scripting vulnerabilities from the source code before an application is deployed. We make use of the OWASP cross-site scripting prevention rules as guideline in our approach. The proposed approach is, so far, only implemented and validated on Java-based Web applications, although it can be implemented in other programming languages with slight modifications. Initial evaluation results have indicated promising results.

Keywords: Cross-Site Scripting, Software Security, Vulnerability Removal

1. Introduction

As technology grows, more and more people are relying on Web applications to accomplish their daily transactions, be it business, personal or otherwise[1]. Businesses and organizations also have been relying heavily on Web applications to provide many of their services. The explosive growth of the Internet and the attractive services it provides have made conducting these daily transactions online a too good opportunity to miss by anyone. However, as Web applications become very important to the success of businesses and organizations, their securities have become more complex[2]. As a result, a lot of security issues have emerged over the years due to the increasing number of threats that plague Web applications[1].

Web applications security testing has, therefore, become a crucial issue to the software security industry as well as governments, businesses, and organizations. Study of the major security threats in web applications has shown that XSS vulnerabilities are among the top ten vulnerabilities, as reported by the Open Web Application Security Project (OWASP)[3].

Security is lacking in most web applications that are in used today and the software engineering community is now focusing on remedying this problem by ensuring security is integrated as early as possible in the Software Development Lifecycle (SDLC) of any web application. However, security vulnerabilities can still remain in web applications despite efforts of integrating security from the beginning of development. That is where the practice of testing the security level of the product comes into play.

*Author for correspondence

Cross-site scripting (XSS) vulnerabilities are among those vulnerabilities found in web applications and can be exploited through XSS attacks when such applications are deployed and running online. Injecting malicious scripts where these applications accept user inputs can result to serious security breaches such as cookie theft, account hijacking, manipulation of web content and theft of private information.

Many research activities have been conducted to address problems related to XSS vulnerabilities since their discovery. According to this systematic review[4], most of the approaches focused on preventing XSS attacks[5–8] and a fewer number focused on detecting XSS vulnerabilities[9–12] in web applications during software security testing. Few research activities have addressed their removal[13,14].

There are three types of XSS attacks namely reflected, stored and DOM (Document Object Model)-based[13,15]. Reflected XSS is executed by the victim's browser and occurs when the victim provides input to the web site. Stored XSS attacks store the malicious script in databases, message forums, comments fields, etc. of the attacked server. The malicious script is executed by visiting users thereby passing their privileges to the attacker. Both reflected and stored XSS vulnerabilities can be found on either client side or server side codes. On the other hand, DOM-based XSS vulnerabilities are found on the client side. Attackers are able to collect sensitive or important information from the user's computer.

In this paper, we propose an approach for the removal of XSS vulnerabilities in web applications. This work is built on a previous approach[14] and extends it by including all the three types of XSS as well as following all the 7 rules stated in the OWASP's XSS prevention rules[16,17]. The rest of the paper is organized as follows: Section 2 describes the proposed approach, which is evaluated in Section 3. The concluding part is in Section 4.

2. Proposed Approach

The proposed approach can be used to secure web applications containing XSS vulnerabilities. Once the XSS vulnerabilities are detected in the source code, the removal process can be done. The OWASP's Enterprise Security API (ESAPI) security mechanisms[18] are followed to remove the detected XSS vulnerabilities. The lines of code where the XSS vulnerabilities are located are identified. Then, we determine which of the ESAPI escaping rules can be applied to replace those lines of code without compromising their functionality. Finally, we generate the secure codes of the escaping statements and put them in place of the vulnerable statements, using the OWASP XSS prevention rules discussed in the next subsection.

2.1 OWASP's XSS Prevention Rules

XSS attacks are carried out by injecting special characters such as '', "", ';', '<' into user inputs. Typically, when a web program references user inputs in its HTML outputs, it expects that client-side browsers treat those inputs as only data. But, injected characters cause these browsers to interpret them as code. Therefore, an XSS exploit occurs by illegally changing the data context to code context. OWASP[16,17] has established XSS prevention rules to follow to ensure that any user input referenced in an HTML output is only treated as data.

The rules ensure that the appropriate escaping mechanisms are used on data inputs based on the HTML context the input is referenced. They help to ensure injected especial characters are not executed thereby disabling XSS vulnerabilities. In addition, using these rules poses no negative effects on the referenced data even if the HTML output is not vulnerable to XSS. The XSS prevention rules are summarized below, for more details on these rules please refer to OWASP[16,17]:

- **Rule#0:** Do not reference user inputs in any other cases except the ones defined in Rule#1 to Rule#7.
- **Rule#1:** Use HTML entity escaping for the untrusted data referenced in an HTML element, for example, <body> htmlEscape(untrusted_data) </body>, where "htmlEscape()" is the HTML entity escaping method.
- **Rule#2:** Use HTML attribute escaping for the untrusted data referenced as a value of a typical HTML attribute such as name and value, for example, <input value='htmlAttrEscape(untrusted_data)'>, where "htmlAttrEscape()" is the HTML attribute escaping method.
- **Rule#3:** Use JavaScript escaping for the untrusted data referenced as a quoted data value in a JavaScript block or an eventhandler, for example, <bodyonload="x='javascriptEscape(untrusted_data)'">, where "javascriptEscape()" is the JavaScript escaping method.
- **Rule#4:** Use CSS escaping for the untrusted data referenced as a value of a property in a CSS style, for example,<table style= "width:

cssEscape(untrusted_data)">, where "cssEscape()" is the CSS escaping method.

- **Rule#5:** Use URL escaping for the untrusted data referenced as a HTTP GET parameter value in a URL, for example, <a href='http://www.site.com?name=urlEscape(untrusted_data)'>, where "urlEscape()" is the URL escaping method.
- **Rule#6:** Use a specific sanitizer for tainted data that contains HTML. Encoding such data is difficult since all the tags in the input can be broken. Therefore a library that can parse and clan HTML formatted text is needed. An example is the OWASP Java HTML Sanitizer[19].
- **Rule#7:** Prevent DOM-Based XSS. The only rule focusing mainly on DOM-based XSS. it has five sub-rules under it.

There is a fundamental difference between Reflected and Stored XSS when compared to DOM based XSS. Reflected and Stored XSS are server side execution issues while DOM based XSS is a client (browser) side execution issue. However, all of this code originates from the server, therefore it is the application developer or owner's responsibility to make it secure from XSS, regardless of the type of XSS flaw it is[17]. There are five sub rules under this rule.

- **RULE #7-1:** HTML Escape then JavaScript Escape Before Inserting Untrusted Data into HTML Subcontext within the Execution Context.
- **RULE #7-2:** JavaScript Escape Before Inserting Untrusted Data into HTML Attribute Subcontext within the Execution Context.
- **RULE #7-3:** Be Careful when Inserting Untrusted Data into the Event Handler and JavaScript code Subcontexts within an Execution Context.
- **RULE #7-4:** JavaScript Escape Before Inserting Untrusted Data into the CSS Attribute Subcontext within the Execution Context.
- **RULE #7-5:** URL Escape then JavaScript Escape Before Inserting Untrusted Data into URL Attribute Subcontext within the Execution Context.

2.2 The ESAPI API

OWASP has released the Enterprise Security API (ESAPI)[18] to the security community, which is a tool that can be used to enforce the XSS prevention rules, discussed in the previous section, in vulnerable web applications. ESAPI helps to enforce security in both developed and developing web applications. It is available in different programming languages such as Java, .NET, and PHP. In our approach, we make use of the ESAPI implemented for Java since our test subjects were developed in Java. ESAPI provides many security mechanisms including authentication, validation, encoding, encryption, security wrappers, filters, and access control to mitigate various web security issues. Hence, it is easier for developers to implement any of these mechanisms in their applications with the same API.

ESAPI needs to be installed into an application for its security controls to be used. Details on its installation and configuration procedures can be found in the documentation downloadable from: code.google.com/p/owasp-esapi-java/. The ESAPI Java documentation for the escaping or encoding APIs can be downloaded from: owasp-esapijava.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Encoder.html. After it is properly installed, ESAPI could be used to secure data that was not validated in an application. It helps to encode the data before it is referenced in an HTML output.

The XSS prevention rules to enforce and the corresponding HTML context determine which appropriate encoding API is to be used. For example, Rule #1 applies when the context is HTML element. Therefore, the following escaping API is used: ESAPI.encoder().encodeForHTML(untrusted_data) [14].

2.3 Removing XSS Vulnerabilities

This approach has two steps. The first step is to locate the lines of code vulnerable to XSS attacks and identify the HTML context in which the vulnerable code is referenced. This step also identifies which escaping mechanism to use according to the HTML context and the appropriate XSS prevention rule. The second step is the source code replacement for the vulnerable codes. Then, the ESAPI escaping rules are applied to replace those vulnerable lines of code without compromising their functionality. The details of the algorithms used in these steps are provided in[14].

The removal of all XSS vulnerabilities detected in the tested programs can be done using these algorithms. Since only the vulnerable codes are escaped, there is not much modification done in the programs that are tested. The main contributions of this work are:

- The removal of detected XSS vulnerabilities of all three types from the source code of web applications.

- The automation of the XSS vulnerabilities removal approach.

3. Evaluation

The above approach was implemented in a prototype tool used to evaluate it for effectiveness and performance. Source codes of complete open-source Java-based large Web applications were used as experimentation data. They were downloaded from the Source Forge site (sourceforge.net), as they are freely available.

3.1 Experiment Data

Five Java-based open source Web applications were used as test data to evaluate the effectiveness of the proposed approach. These applications were also used as test subjects in[14] with the actual number of XSS vulnerabilities stated for each application. Table 1 shows the description of the applications. Events and Classifieds have also been used as benchmarks in related works[20,21]. Roomba has been used in[22]. PersonalBlog and JGossip have been used in[23]. Library classes are not included in the LOC counts. The numbers of XSS vulnerabilities in each application are shown in Table 2. These vulnerabilities include all the three types of XSS vulnerabilities.

Table 1. Test Subjects

Java Application	Lines of Code	Description
Events	3818	An event management system.
Classifieds	5745	An online system for advertising and shopping.
Roomba	3438	A room booking system for small to medium-sized hotels.
PersonalBlog	17149	An online blogging system where users can publish their blogs.
JGossip	79685	A large-scale forum application for discussing different topics.

Table 2. Number of XSS Vulnerabilities

Java Application	Lines of Code	No. Actual XSS Vul.
Events	3818	*20*
Classifieds	5745	*19*
Roomba	3438	*129*
PersonalBlog	17149	*2*
JGossip	79685	*31*

3.2 Results and Discussion

All the XSS vulnerabilities existing in the tested applications were successfully removed using the proposed approach. The results are shown in Table 3. In addition, the DOM-based XSS vulnerabilities existing in the tested applications were also successfully removed. These DOM-based XSS vulnerabilities were not included in the previous approach we are extending and comparing our work with[14].

Table 3. Experiment Results

Java Application	Lines of Code	No. Actual XSS Vul.	XSS Vul. Removed
Events	3818	*20*	*All*
Classifieds	5745	*19*	*All*
Roomba	3438	*129*	*All*
PersonalBlog	17149	*2*	*All*
JGossip	79685	*31*	*All*

Our approach was able to remove 100% (201/201) of all the identified vulnerabilities. Manual inspection has proved that all the actual vulnerabilities were part of the potential vulnerabilities reported using our approach as in[14]. The modified applications were tested again after the actual XSS vulnerabilities were removed, with the same XSS attack vectors used in the detection module. This time the attacks were not successful. Therefore our approach is effective in removing detected XSS vulnerabilities in the tested applications.

4. Conclusion

This paper presented an approach for XSS removal using the OWASP XSS prevention rules as well as the ESAPI security API. The approach can remove detected XSS vulnerabilities in Java-based web applications. Cross-site scripting is a major security problem for web applications. It can lead to account or web site hijacking, loss of private information, and denial of service, all of which victimize the site users. Preliminary evaluation results have shown that our approach is able to remove all the three types of XSS vulnerabilities in Java-based web applications. The same approach can also be extended in other programming languages. Further analysis will be conducted to test the approach on a wider range of real world web applications.

5. Acknowledgement

This work was supported by the Malaysian Ministry of Education under the Fundamental Research Grant Scheme (FRGS 08-02-13-1368).

6. References

1. McGraw G. Software Security: Building Security. Boston, MA: Addison Wesley Profesional; 2006.
2. Fogie S, Grossman J, Hansen R, Rager A, Petkov PD. XSS Attacks: Cross Site Scripting Exploits and Defense. In: Forgie S, editor. Burlington, MA: Elsevier, Inc/Syngress Publishing, Inc; 2007.
3. OWASP. Cross-site Scripting (XSS) - OWASP. 2014. Available from: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
4. Hydara I, Sultan ABM, Zulzalil H, Admodisastro N. Current state of research on cross-site scripting (XSS) - A systematic literature review. Inf Softw Technol. Elsevier B.V.; 2013; 58:170–86. Available from: http://dx.doi.org/10.1016/j.infsof.2014.07.010
5. Sharma P, Johari R, Sarma SS. Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. Int J Syst Assur Eng Manag. 2012 Sep 26; 3(4):343–51. Available from: http://link.springer.com/10.1007/s13198-012-0125-6
6. Sun Y, He D. Model Checking for the Defense against Cross-Site Scripting Attacks. 2012 IEEE International Conference on Computer Science and Service System; 2012. p. 2161–4. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6394855
7. Van Gundy M, Chen H. Noncespaces: Using randomization to defeat cross-site scripting attacks. Comput Secur. Elsevier Ltd; 2012 Jun; 31(4):612–28. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0167404811001477
8. Scholte T, Robertson W, Balzarotti D, Kirda E. Preventing Input Validation Vulnerabilities in Web Applications through Automated Type Analysis. 2012 IEEE 36th Annual Computer Software and Applications Conference; 2012. p. 233–43. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6340148
9. Agosta G, Barenghi A, Parata A, Pelosi G. Automated Security Analysis of Dynamic Web Applications through Symbolic Code Execution. 2012 Ninth International Conference on Information Technology - New Generations; 2012. p. 189–94. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6209165
10. Al-Amro H, El-Qawasmeh E. Discovering security vulnerabilities and leaks in ASP.NET Websites. International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec); Kuala Lumpur: IEEE; 2012. p. 329–33.
11. Van-Acker S, Nikiforakis N, Desmet L, Joosen W, Piessens F. Flash over: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. ASIACCS'12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security. Seoul: ACM; 2012. p. 12–3.
12. Duchene F, Groz R, Rawat S, Richier J-L. XSS vulnerability detection using model inference assisted evolutionary fuzzing. 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation; 2012. p. 815–7. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6200193
13. Bathia P, Beerelli BR, Laverdière M. Assisting programmers resolving vulnerabilities in Java web applications. CCIST 2011: Communications in Computer and Information Science. 2011. p. 268–79.
14. Shar LK, Tan HBK. Automated removal of cross site scripting vulnerabilities in web applications. Inf Softw Technol. Elsevier B.V.; 2012 May;54(5):467–78. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0950584911002503
15. CWE. CWE - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (2.5). The MITRE Corporation. 2014. Available from: http://cwe.mitre.org/data/definitions/79.html
16. OWASP. XSS (Cross Site Scripting) Prevention Cheat Sheet. OWASP. 2014. Available from: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
17. OWASP. DOM based XSS Prevention Cheat Sheet. OWASP. 2014. Available from: https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
18. OWASP. OWASP Enterprise Security API - OWASP. 2014 Sep 19. Available from: https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API
19. OWASP. OWASP Java HTML Sanitizer. OWASP. 2015. Available from: https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project
20. Halfond WGJ, Orso A. AMNESIA: Analysis and Monitoring for Neutralizing SQL-injection Attacks. Proc 20th IEEE/ACM Int Conf Autom Softw Eng; 2005. p. 174–83. Available from: http://doi.acm.org/10.1145/1101908.1101935
21. Halfond WGJ, Orso A, Manolios P. WASP: Protecting web applications using positive tainting and syntax-aware evaluation. IEEE Trans Softw Eng. 2008; 34(1):65–81.
22. Liu H, Kuan Tan HB. Testing input validation in Web applications through automated model recovery. J Syst Softw. 2008; 81:222–33.
23. Martin M, Lam MS. Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking. 17th Conference on Security Symposium. CA, USA: USENIX Association Berkeley; 2008. p. 31–43. Available from: https://www.usenix.org/legacy/event/sec08/tech/full_papers/martin/martin.pdf