ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol4, Issue 6, June 2017

Evaluating Machine Learning Algorithm on Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications

[1] Kishan Babu T D, [2] Dr. Jayanna H S

[1, 2] Cyber Forensics and Information Security, Dept. of Information Science and Engineering
Siddaganga Institute of Technology, Tumakuru, Karnataka, India.

[1] tdkishanbabu@gmail.com, [2] jayannahs@gmail.com

***Abstract*— In this paper, the prediction and analysis of cross-site scripting (XSS) security vulnerabilities in web application's source code is demonstrated. Cross-site scripting (XSS) is a security vulnerability that affects the web applications and it occurs due to improper or lack of sanitization of user inputs. There is no single solution that can effectively mitigate XSS attacks. More research is needed in the area of vulnerability removal from the source code of the applications before deployment. Security inspection and testing require experts in security who think like an attacker and locating vulnerable code locations is a challenging task. Alternatively, there are also vulnerability prediction approaches based on machine learning techniques which showed that static code attributes such as code complexity measures are cheap and useful predictors. The main focus is on prediction of XSS vulnerabilities and extracts the relevant features to classify vulnerable source code file from benign one. Attack prevention and vulnerability detection are the areas focused in this study.**

***Index Terms*— Cross-site scripting vulnerability, Input validation, Machine learning, Web application security.**

I. INTRODUCTION

In the present day lifestyle, people depend on the web applications for their daily activities such as social communication, medical services and banking transactions etc., an illegal HTTP requests, cookies, session hijacking, redirecting to phishing links and creating malicious websites, installing ransomwares other illegal activities can be performed due to the security vulnerabilities present in these web based applications. Based on the statistical survey reports [2] on these vulnerabilities we can say that 55% include the vulnerable websites. Cross Site Scripting (XSS) is considered as a major vulnerability in web application and it has also been reported by Open Web Application Security Projects (OWASP) and Common Vulnerability Exposure (CWE) in
2013 [2]. This attack mainly occurs due to the flaws present in the source code which allows the user's input data to exactly appear on the server's output statement without any validation. For predicting the XSS vulnerability in the web application's source code the method followed till now were static and dynamic analysis techniques [1]. A set of predefined static rules are used without executing in the static analysis technique. Complex analysis techniques with the execution of program are done in the dynamic technique to give a much accurate results. Based on the research work of [4] [5], we can state that the software metrics and static code attributes are the basic factors for constructing a machine learning model for analysis and prediction of vulnerabilities in web applications.XSS Vulnerability can be divided into three types: Persistent XSS, in this the malicious script originates from the database of the web application and this attack normally occurs in forums, blogs and in social networking sites. Reflected XSS, in this the malicious input originates due to the request of the victim this normally happens due in error messages, alerts and greetings. DOM-based XSS (Document Object Module), in this the client-side browser stores the vulnerability but not the server-side and Here invalid user inputs are used and the DOM structure is obtained dynamically. In this paper, we follow the procedures to initially extract the basic features and then to extract the context features present in the source code of the web application and then to build machine learning based prediction model to identify the vulnerabilities present in the given source code. This approach follows with the several other works carried out till the time which includes the use of context information present in the source code to detect the vulnerabilities. The implementation is carried out with a prototype which helps in automatic extraction of the required features and to detect and identify the safe files from the vulnerable ones.

The flow of this paper is as follows, Section II describes the background and motivation involved which provided the basic framework. Section III explains the earlier works related to XSS. In Section IV the procedure to extract required features is explained. In Section V requirements like the data set being used, prior experimental setting, and

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol4, Issue 6, June 2017

how the performance is measured are discussed in detail. In Sections VI the results obtained from the experiment are explained. In the last section, conclusion of the paper is stated and also the future enhancements which can be made are discussed.

II. BACKGROUND AND MOTIVATION

This section explains about the XSS (Cross-site Scripting) Security vulnerabilities and to analyse the drawbacks present in the existing approaches.
.

A. Cross-Site Scripting (XSS) Vulnerabilities

This is a type of injection attacks mainly in the application level and here the scripts from a malicious attacker are injected into trusted web sites [1]. XSS attacks arise when an attacker uses a web application to send malicious code, mostly in the form of a browser side script. An attacker uses XSS to send malicious scripts to the legitimate users. It thinks that the script came from a trusted sender these malicious scripts can read the cookies, tokens, or any other credentials of the benign users.

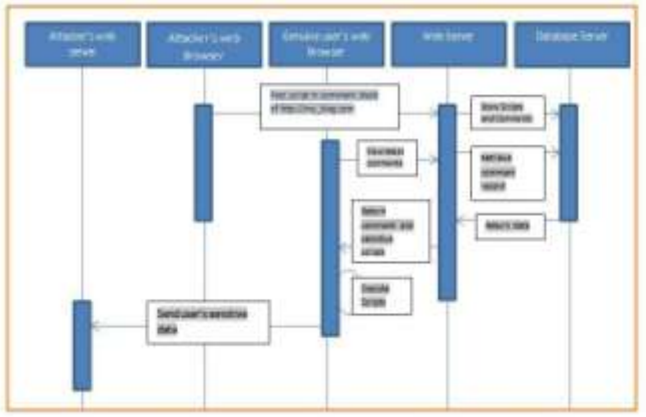
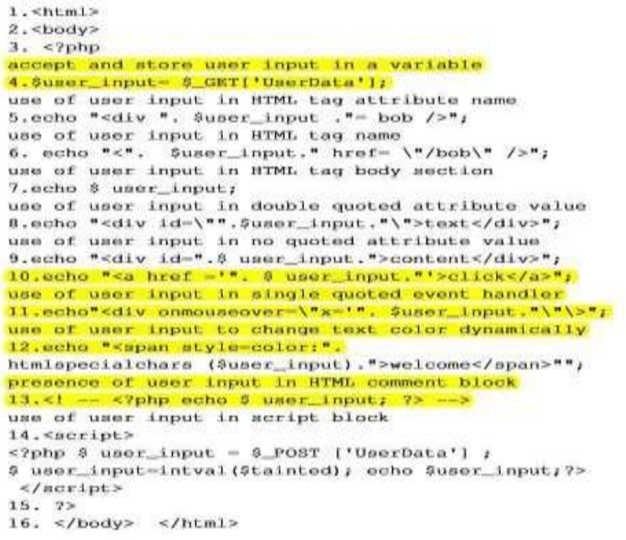


Fig.1 Sequence of steps involved in performing a stored XSS attack.

Fig. 1 represents the sequence of steps involved in performing stored XSS attack. Attackers injects a malicious script which will display an alert dialog box saying that the database has been hacked, for this they make use of comment fields, search fields or any other fields where the can insert text easily. Once a benign user logs in with his valid account, this alert message pops up and the same thing will appear on the server side. Once this executes successfully it will be stored into the database and will reflect on to the page of all the valid users. In this way the stored XSS attack can be performed.

B. Limitations of Vulnerability Detection Approaches

There are several methods to identify the Cross Site Scripting vulnerability in source code of PHP web applications. The existing static vulnerability detection method focuses just on the static rules. They use the standard PHP built-in sanitized functions (HTML special characters and HTML entities etc.) [6]. The Fig. 2 explains a sample PHP code which contains the vulnerability, in this the user-input is referenced in the output-statement with different context (e.g. HTML attribute, comment, JavaScript, URL etc.).In the below sample code, in statement 4 denotes the user input is assigned to a user defined PHP variable “$user_input”. In statement 12, user input is used to change the colour of the text. According to the standards of the sanitized function, this statement is having the vulnerability. Some of the other vulnerabilities present in this code sample are, user input is referenced in the body_anchor_NQ_Attr_Val, JavaScript block and comment blocks are present in line number 10, 11 and 13 respectively. These require special context-sensitive filters to mitigate the XSS Vulnerabilities. Several State_of _the_art_techinques include, In paper [9], to identify the HTML entities in XSS vulnerability they have used pattern matching techniques, and the authors have concentrated on only JAVA based web applications. In paper [10], the main focus is given for the context-mismatched sanitisation and inconsistent multiple sanitisation to identify the XSS vulnerabilities.



```
1.<html>
2.<body>
3. <?php
accept and store user input in a variable
4.$user_input= $_GET['UserData'];
use of user input in HTML tag attribute name
5.echo "<div ". $user_input ."= bob />";
use of user input in HTML tag name
6. echo "<".  $user_input." href= \"/bob\" />";
use of user input in HTML tag body section
7.echo $ user_input;
use of user input in double quoted attribute value
8.echo "<div id=\"".$user_input."\">text</div>";
use of user input in no quoted attribute value
9.echo "<div id=".$ user_input.">content</div>";
10.echo "<a href ='". $ user_input."'>click</a>";
use of user input in single quoted event handler
11.echo"<div onmouseover=\"x='". $user_input."\"\>";
use of user input to change text color dynamically
12.echo "<span style=color:".
htmlspecialchars ($user_input).">welcome</span>"";
presence of user input in HTML comment block
13.<! -- <?php echo $ user_input; ?> -->
use of user input in script block
14.<script>
<?php $ user_input = $_POST ['UserData'] ;
$ user_input=intval($tainted); echo $user_input;?>
 </script>
15. ?>
16. </body>  </html>
```

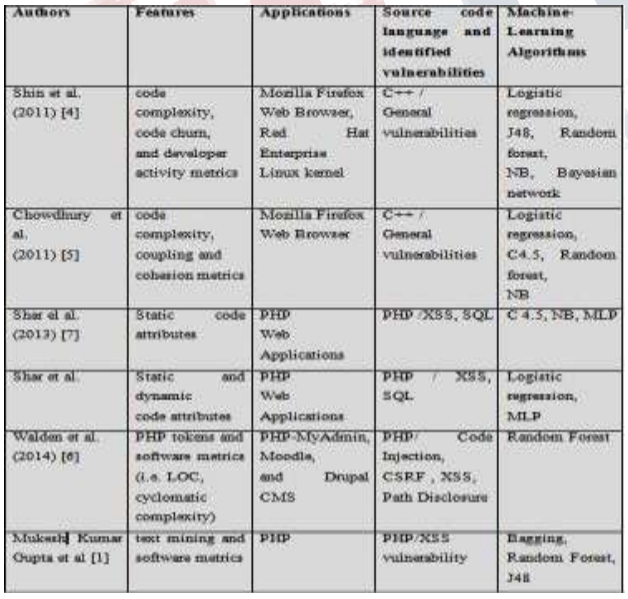
Fig. 2 User-input referenced in different HTML contexts in the PHP source code

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol4, Issue 6, June 2017

III. RELATED WORK

The prior vulnerability prediction methods to classify the vulnerable files from the safe ones are discussed in this section. The below Table-I shows different existing approaches based on several standards i.e., application, source code and machine-learning classifiers.
The author in the paper [5] focused on cohesion, complexity, and coupling metrics to detect the weakness in Mozilla Firefox browser. In the same way the author Shin et al. in paper [4], has used code churn, developer activity metrics and code complexity to segregate common vulnerable files from the safe files. Being opposite to this, Shar et al. [13], defended that simple code packages has several XSS vulnerabilities. According to authors in [11], the main source of XSS vulnerability is the use of invalid inputs. Sanitization code, input and output were derived from the dynamic and static analysis techniques. Machine learning models were built using the sensitive statements.
Author Walden et al. [6], explains the text mining features and software metrics. He noticed that this feature provides significantly superior results in detecting XSS vulnerability. In paper [1], the authors Mukesh Kumar Gupta et al. focused on the text mining and software metrics to predict XSS vulnerability. The proposed approach follows the above mentioned techniques [1, 6] to predict the vulnerabilities present in the PHP source codes.



Authors	Features	Applications	Source code language and identified vulnerabilities	Machine-Learning Algorithms
Shin et al. (2011) [4]	code complexity, code churn, and developer activity metrics	Mozilla Firefox Web Browser, Red Hat Enterprise Linux kernel	C++ / General vulnerabilities	Logistic regression, J48, Random forest, NB, Bayesian network
Chowdhury et al. (2011) [5]	code complexity, coupling and cohesion metrics	Mozilla Firefox Web Browser	C++ / General vulnerabilities	Logistic regression, C4.5, Random forest, NB
Shar el al. (2013) [7]	Static code attributes	PHP Web Applications	PHP /XSS, SQL	C 4.5, NB, MLP
Shar et al.	Static and dynamic code attributes	PHP Web Applications	PHP / XSS, SQL	Logistic regression, MLP
Walden et al. (2014) [6]	PHP tokens and software metrics (i.e. LOC, cyclomatic complexity)	PHP-MyAdmin, Moodle, and Drupal CMS	PHP/ Code Injection, CSRF , XSS, Path Disclosure	Random Forest
Mukesh Kumar Gupta et al [1]	text mining and software metrics	PHP	PHP/XSS vulnerability	Bagging, Random Forest, J48

Table-I Comparisons between the existing approaches

IV. PROPOSED METHOD

Initially, the user input context features in the output-statement are extracted. Then, the basic features which illustrate the characteristics of sanitization input and output routines are extracted. Once extracting these two features, tokenizing process is done using the PHP built-in Zend engine's lexical scanner. Thereafter, feature set is constructed by combining basic and context features. At the end, a number of machine learning algorithms are used to construct several prediction models. The below Fig.3 illustrate the phases involved to categorize vulnerable PHP source code from the safe ones.

A. Main Feature Extraction Algorithm

Algorithm 1 presents an organized procedure by which the relevant features can be extracted from the PHP source codes. This involves two major steps, namely; 1. Extraction of basic code features. 2. Determine the user input context present in the output statement. At first, PHP codes that are present in the HTML block is extracted by using HTML DOM Parser.

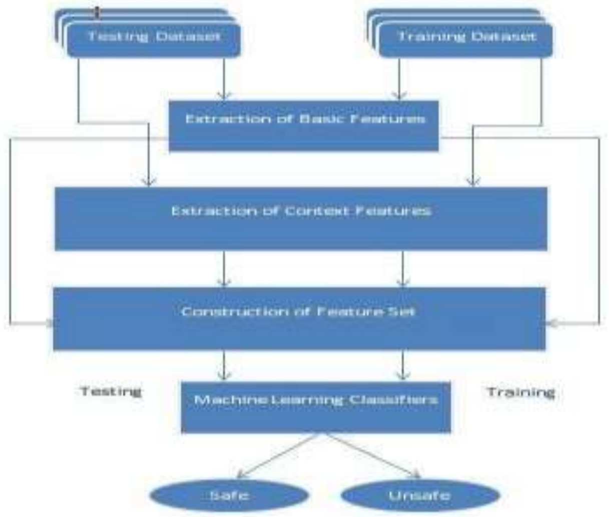

Testing Dataset
Training Dataset
Extraction of Basic Features
Extraction of Context Features
Construction of Feature Set
Testing
Machine Learning Classifiers
Training
Safe
Unsafe

Fig.3 Proposed vulnerability prediction approach

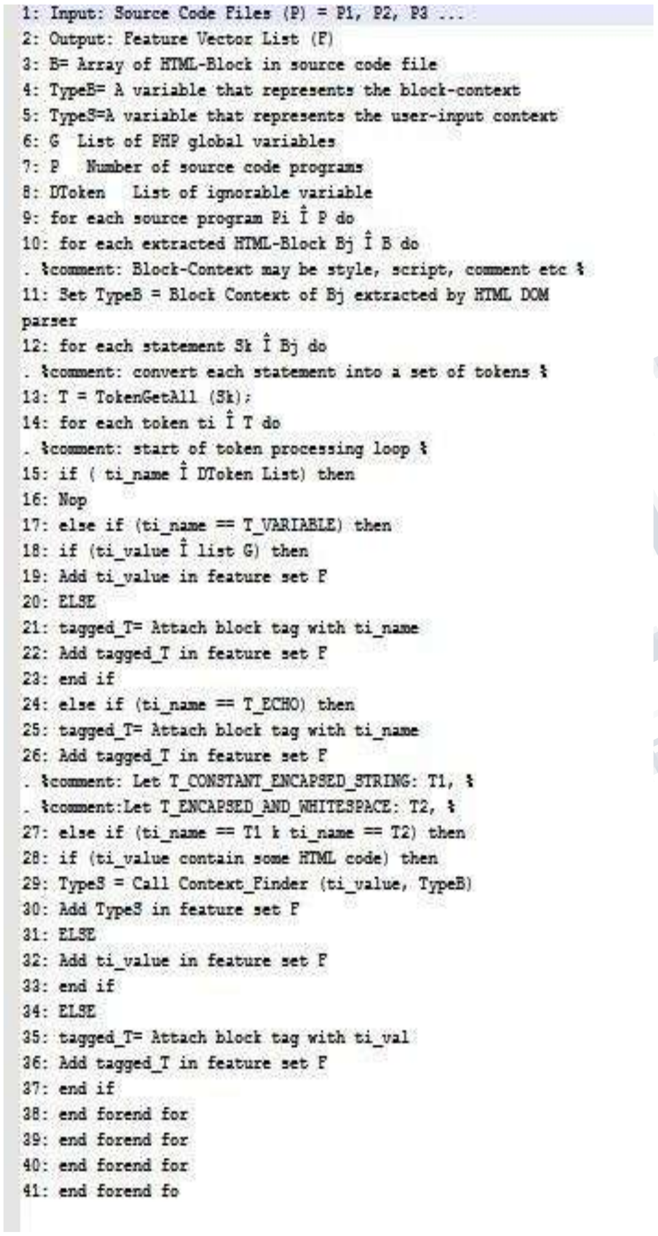
The HTML blocks such as comment, script, style etc. are the block contexts for the PHP codes. Then the extracted HTML-Block contexts are tokenized using the PHP Zend Engine Lexical Scanner. The feature set includes the tagged tokens as the basic code features. Next, the PHP codes in HTML tag are processed. The Context finder algorithm is used to extract the user input context.

IFERP
connecting engineers... developing research

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol4, Issue 6, June 2017

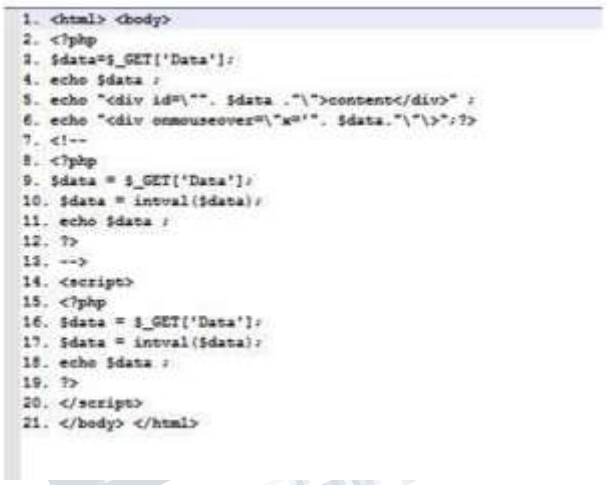
Later, the Block-context tag is concatenated with the user input context and we consider the tagged contents in the feature set. The HTML comment statements are removed by pre-processing the source code during the feature extraction process. As these statements do not deliver any meaningful information they are not used in building the prediction model.



```
1: Input: Source Code Files (P) = P1, P2, P3 ...
2: Output: Feature Vector List (F)
3: B= Array of HTML-Block in source code file
4: TypeB= A variable that represents the block-context
5: TypeS=A variable that represents the user-input context
6: G  List of PHP global variables
7: P   Number of source code programs
8: DToken   List of ignorable variable
9: for each source program Pi Î P do
10: for each extracted HTML-Block Bj Î B do
. %comment: Block-Context may be style, script, comment etc %
11: Set TypeB = Block Context of Bj extracted by HTML DOM
parser
12: for each statement Sk Î Bj do
. %comment: convert each statement into a set of tokens %
13: T = TokenGetAll (Sk);
14: for each token ti Î T do
. %comment: start of token processing loop %
15: if ( ti_name Î DToken List) then
16: Nop
17: else if (ti_name == T_VARIABLE) then
18: if (ti_value Î list G) then
19: Add ti_value in feature set F
20: ELSE
21: tagged_T= Attach block tag with ti_name
22: Add tagged_T in feature set F
23: end if
24: else if (ti_name == T_ECHO) then
25: tagged_T= Attach block tag with ti_name
26: Add tagged_T in feature set F
. %comment: Let T_CONSTANT_ENCAPSED_STRING: T1, %
. %comment:Let T_ENCAPSED_AND_WHITESPACE: T2, %
27: else if (ti_name == T1 k ti_name == T2) then
28: if (ti_value contain some HTML code) then
29: TypeS = Call Context_Finder (ti_value, TypeB)
30: Add TypeS in feature set F
31: ELSE
32: Add ti_value in feature set F
33: end if
34: ELSE
35: tagged_T= Attach block tag with ti_val
36: Add tagged_T in feature set F
37: end if
38: end forend for
39: end forend for
40: end forend for
41: end forend fo
```

Algorithm 1: Feature vector preparation

B. Extracting the features from PHP source code
The features extracted are explained in this section below, From the given source code in Fig. 4 the feature extraction methodology involves extracting Script block-context, HTML_ELEMENT and Comment. Later the tokenisation of each and every extracted block is done, which helps to build the feature set. The extracted features are represented in Table-II.



```
1. <html> <body>
2. <?php
3. $data=$_GET['Data'];
4. echo $data ;
5. echo "<div id=\"". $data ."\">content</div>" ;
6. echo "<div onmouseover=\"x='". $data."\"\>";?>
7. <!--
8. <?php
9. $data = $_GET['Data'];
10. $data = intval($data);
11. echo $data ;
12. ?>
13. -->
14. <script>
15. <?php
16. $data = $_GET['Data'];
17. $data = intval($data);
18. echo $data ;
19. ?>
20. </script>
21. </body> </html>
```

Fig. 4 Sample PHP source code

Code Line	Proposed Approach Features (F3)
1	
2	
3	T_VARIABLE_HTML_Element $_GET_HTML_Element
4	T_ECHO_HTML_Element, T_VARIABLE_HTML_Element
5	T_ECHO_HTML_Element, HTML_Element_ H_TAG_DQ_Attr_Val, T_VARIABLE_HTML_Element
6	T_ECHO_HTML_Element, HTML_Element_ HTAG_Event_DQ_Attr_Val, T_VARIABLE_HTML_Element
7	
8	
9	T_VARIABLE_comment_block, $_GET_comment_block
10	T_VARIABLE_comment_block, $_GET_comment_block, T_VARIABLE_comment_block
11	T_ECHO_comment_block , T_VARIABLE_comment_block
12	
13	
14	
15	
16	T_VARIABLE_script_block, $_GET_script_block
17	T_VARIABLE_script_block, intval_script_block, T_VARIABLE_script_block
18	T_VARIABLE_script_block, T_VARIABLE_script_block
19	
20	

Table II. Extracted Features

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol4, Issue 6, June 2017

.

V. EXPERIMENTAL SETTINGS AND DATASET

The publicly available GIT repository [15] as the dataset for the experimentation, this comprises of the unnatural test case generator. This has 4352 PHP sample unsafe codes which includes all types of vulnerabilities.

A. Experimental Setting:

For performance evaluation 10-fold cross validation technique in WEKA machine learning tool is used. Training and testing datasets are divided randomly into 90% and 10 % respectively so that to form a disjoint set. Finally the root relative squared error rates are considered to measure the performance of the system being proposed.

B. Performance measures

Here different machine learning models are built using the WEKA tool [16]. Several performance measures like accuracy, precision, recall and F-measures are used to estimate the performance taking the vulnerable source files as input

VI. RESULTS

This method uses the tagged tokens and also considers the user input in the output statements. Here the XSS vulnerable files can be detected very easily. Given the source code of the PHP application, we can easily detect whether the source code is safe or vulnerable. Different classifiers help us to analyse the error rate which helps us to evaluate the performance of this model. Classifiers like Bagging and Logistic are used to analyse the error rates. The tainted methods and sanitisation of the url helps to detect the vulnerability.
The result from the Table III says that the Bagging classifier's performance is better when compared to the other classifiers.

Classifiers	Root relative squared error	Relative absolute error
Logistic	94.3242%	39.2857%
Bagging	96.7842%	63.7302%

Table III. Error rates with different classifiers

VII. CONCLUSION

Vulnerability in web applications may cause theft of private and important user information. Vulnerability detection is a main task in securing the web applications before releasing. Here in this paper, the method of predicting the Cross site scripting (XSS) vulnerability in the PHP web application's source code is stated. Depending on the several factors like the Tagged _Tokens, Tainted methods, Sanitisation functions the vulnerability of XSS can be predicted. This helps for an organisation to penetrate with the developed web application and detect the vulnerabilities and then to release it to the outside market. Hence this acts as a penetration tool to detect and classify the vulnerable files from the benign ones.

REFERENCES

[1] M. K. Gupta, M. C. Govil and G. Singh, "Predicting Cross-Site Scripting (XSS) security vulnerabilities in web applications," 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE), Songkhla,
2015, pp. 162-167.

[2] WhiteHatSecurity. Web statistics report. https://whitehatsec.com/categories/statistics-report, 2013. Accessed: 2013-06-26.

[3] Isatou Hydara, Abu Bakar Md. Sultan, Hazura Zulzalil, and Novia Admodisastro. Current state of research on cross-site scripting a systematic literature review. Information and Software Technology, 58(0):170 – 186, 2015.

[4] Yonghee Shin, A. Meneely, L. Williams, and J.A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 37(6):772–787, Nov 2011.

[5] Istehad Chowdhury and Mohammad Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture, 57(3):294 – 313,
2011. Special Issue on Security and Dependability Assurance of
Software Architectures.

[6] J. Walden, J. Stuckman, and R. Scandariato. Predicting vulnerable components: Software metrics vs text mining. IEEE 25th International Symposium on Software Reliability Engineering (ISSRE), pages 23–33, Nov 2014.

[7] Lwin Khin Shar and Hee Beng Kuan Tan. Predicting sql injection and cross site scripting vulnerabilities through mining input sanitization patterns. Information and Software Technology,
55(10):1767 – 1780, 2013.

[8] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen. Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering, 40(10):993–1006, Oct 2014.

[9] Lwin Khin Shar and Hee Beng Kuan Tan. Automated removal of cross site scripting vulnerabilities in web applications. Information and Software Technology, 54:467–478, 2012.

[10] Prateek Saxena, David Molnar, and Benjamin Livshits. Scriptgard: Automatic context-sensitive sanitization for large-scale legacy web applications. Proceedings of the 18th ACM Conference on Computer and Communications Security, pages 601–614, 2011.

[11] Lwin Khin Shar, Hee Beng Kuan Tan, and Lionel C. Briand. Mining sql injection and cross site scripting vulnerabilities using hybrid program analysis. Proceedings of the 2013 International Conference on Software Engineering, pages 642–651, 2013.

[12] Aram Hovsepyan, Riccardo Scandariato, Wouter Joosen, and James Walden. Software vulnerability prediction using text analysis techniques. Proceedings of the 4th International Workshop on Security Measurements and Metrics, pages 7–10, 2012.

[13] Lwin Khin Shar and Hee Beng Kuan Tan. Predicting common web application vulnerabilities from input validation and sanitization code patterns. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pages 310–313, 2012.

[14] Ibéria Medeiros, Nuno F. Neves, and Miguel Correia. Automatic detection and correction of web application

vulnerabilities using data mining to predict false positives. Proceedings of the 23rd International Conference on World Wide Web, pages 63–74, 2014.

[15] Bertrand STIVALET Aurelien DELAITRE. Php vulnerabilities test suite. https://github.com/stivalet/PHP-Vulnerability-test-suite , 2014. Accessed: 2014-07-13.

[16] Peter Reutemann Eibe Frank, Mark Hall and Len Trigg. Weka: Data mining tool. http://www.cs.waikato.ac.nz/ml/weka, 2013. Accessed: 2013-06-26.