Prediction of Cross-Site Scripting Attack Using Machine Learning Algorithms

Vishnu. B. A
PG Student
Amrita School of Engineering
Amrita Vishwa Vidyapeetham (University)
Coimbatore
vishnuba89@gmail.com

Ms. Jevitha. K. P
Assistant Professor
Amrita School of Engineering
Amrita Vishwa Vidyapeetham (University)
Coimbatore
kp_jevitha@cb.amrita.edu

ABSTRACT

Dynamic web pages are widely used by web applications to provide better user experience and to attract more web users. The web applications use the client side and server side scripts to provide dynamic behavior to the web pages. Cross-Site Scripting (XSS) attack uses malicious scripts and links injected into the trusted web pages to steal sensitive data from the victims. In this paper, we present the experimental results obtained using three machine learning algorithms (Naïve Bayes, Support Vector Machine and J48 Decision Tree) for the prediction of Cross-site scripting attack. This is done using the features based on normal and malicious URLs and JavaScript. J48 gave better results than Naïve Bayes and Support Vector Machine based on the features extracted from URL and Java Script code. All the algorithms gave comparatively better results with discretized attributes but noticeable difference in performance was seen only in the case of SVM.

Keywords

Web application security, Machine learning, Cross Site Scripting (XSS).

1. INTRODUCTION

The World Wide Web has become today's most common, cheapest and fastest communication medium in the world. Due to ease of access and the variety of services available, millions of people are using it for their day to day activities. Social networking sites, online shopping web sites, cloud storage services etc. are becoming more popular. In such a scenario, one of the ways to attract Internet customers is to have web pages that are user friendly, attractive and dynamic. Server and client side script plays an important role in providing better experience to web users. On the other hand, malicious users/attackers use these functionalities to build attack vectors that directly or indirectly attack these web users. The main aim of these attacks are to steal account credentials such as username and passwords, personal details, session cookies, get remote system access and to spread malwares.

ICONIAAC '14, October 10 - 11 2014, Amritapuri, India
Copyright 2014 ACM 978-1-4503-2908-8/14/08…$15.00.

http://dx.doi.org/10.1145/2660859.2660969

Cross Site Scripting has become one of the major attack vector used against many web sites. Open Web Application Security Project (OWASP) Top 10 for the year 2013 lists XSS as the 3rd most critical web application security attack [1]. Careless surfing, use of outdated browsing and antivirus software tools, increased activity of spammers, forwarding of links and images over social networking and internet forums, lack of preventive security measures during the development of web applications aid the wide-spread of this attack. Once an attacker is able to find vulnerability associated with a web application, he will be able to attack its web users using this vulnerable web application. Client side scripts like Java Scripts [5] are most widely used for implementing XSS attacks.

The basic fact for the success of this attack is that most web users are not aware of the actual code/URLs of hyperlinks and the scripts running behind the web pages. So the web attackers are more successful in implementing such attacks using scripts or URLs. When an attacker is interested in a particular web site, they start checking the responses sent by the server, to find the flaws associated with it. After gathering enough information, they design attack vectors. Cross Site Scripting (XSS) is a web application attack which enables the attacker/malicious user to inject malicious code or links in a vulnerable web application, which will be executed when the web page is viewed by other web users [2].

XSS attacks can be categorized into 3 classes

Reflected/Non Persistent XSS: If an attacker is able to find the vulnerability associated with an web application (such as lack of encoding schemes and user input validation methods), he may craft special links and URLs with malicious script code and spread it to internet users via social networking sites and e-mails. If a user clicks on this link, he will get a response web page which is embedded with the malicious script and can be a victim of the attack. It will happen every time user clicks on that hyperlink.

Stored/Persistent XSS: Here, the malicious input given by the attacker is stored in the server and later it is executed when web page is viewed by other users. This is the most powerful XSS attack, since the attack can spread to millions of users at the same time. One of the examples includes storing of malicious input given by the attacker as comments or opinions in the web sites. When other users view the web page containing this malicious script, their browser executes the script and can be a victim of the attack [3].

Local/DOM based XSS: Applications use client side scripts to dynamically generate HTML contents. In DOM based attack, the original client side script code runs maliciously as a result of change in the DOM environment at victim's browser. This need not require any communication with server before modifying the web page [17].

2. RELATED WORK

The Same Origin Policy was introduced by Netscape in order to restrict the resource access by the script code. According to this policy, the JavaScript code has read/write access only to the elements or windows or documents which are originated from the same origin [6]. The free, open source add-on NoScript developed by Mozilla for Firefox, Sea monkey and other Mozilla based browsers helped in selectively allowing the execution of scripts which are in the user's trusted list [7].

The malicious inputs or scripts inserted by the attacker into the web page gets executed when no proper sanitization mechanisms or HTML encoding schemes are used in the resulting web page. When proper encoding schemes are used, the execution of script inputs with opening and closing script tags can be prevented. The prevention mechanisms and techniques that can be used against XSS attacks can be found in OWASP prevention cheat sheet [4]. The integrity of web page can be taken into account for the detection of injected malicious user inputs. Tripwire [8] uses the hash value of the web page to verify its integrity. At later point of time, the hash value is recomputed and if there is any change in the pre computed hash value, it indicates the modification of web page [8].

Kirda et al [9] proposed a client side solution for the mitigation of XSS attacks. They presented a web proxy, Noxes which requires minimal user interaction and uses the rules which are manually added and automatically generated. By mitigating XSS attacks, it provides protection against information leakage from web users. Nunan and Souto et al [10] proposed machine learning approach for the automatic classification of Cross Site Scripting attacks. They extracted features from the URLs and web documents and used them for analyzing classification performance.

The client inputs are used by web applications to perform sensitive file access, database access etc. According to L. K. Shar and H. B. K. Tan [15], if the inputs given by the user are not cleansed before they are used in the output web pages or before using them for these sensitive operations, it can result in security violations and attacks. They proposed SQL injection and Cross Site Scripting vulnerability prediction using data mining. The nodes in the CFG of web application which are susceptible to SQL injection or Cross Site Scripting attacks are the sensitive sink nodes. The data dependence graph is generated for these sensitive sinks and the input sanitization methods that can be applied for the nodes of data dependency graph is taken into account for creating the static attribute vector for classification. Based on this attribute vector, they built predictive model for the prediction of SQL injection and Cross Site Scripting attack vulnerabilities. Later, they added dynamic vector and used clustering in order to improve performance and testing capability [16].

3. PROPOSED APPROACH

Our proposed approach is to use machine learning algorithms for the classification of normal and malicious web pages based on the features extracted from URL and Java Script code. Figure 1 depicts the steps involved in the proposed method. Since JavaScript and URL are mainly used for implementing and propagating XSS attack, they were used for identifying the features for the preparation of the dataset. So, the initial step is to collect normal web pages and web pages which are affected with XSS attack. The second step is to extract URL and JavaScript code from these web pages. The features are identified and the dataset is created using the collected URL and JavaScript code. This dataset is then used for training and testing the selected machine learning algorithms.

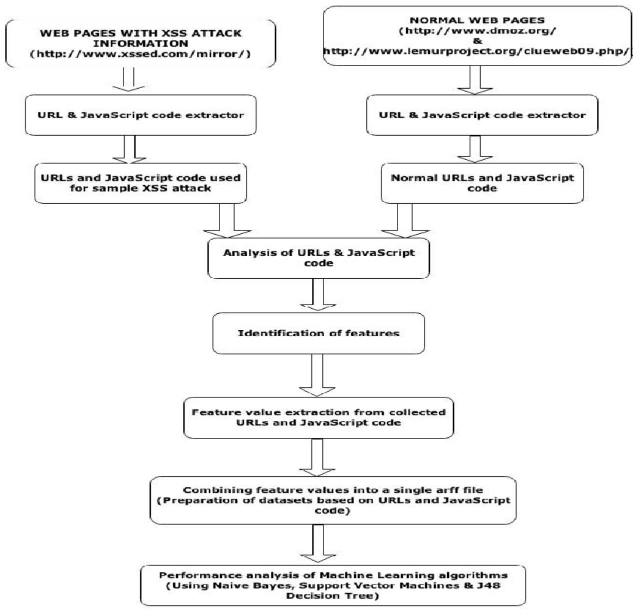

WEB PAGES WITH XSS ATTACK INFORMATION (http://www.xssed.com/mirror/)
NORMAL WEB PAGES (http://www.dmoz.org/ & http://www.lemurproject.org/clueweb09.php/
URL & JavaScript code extractor
URL & JavaScript code extractor
URLs and JavaScript code used for sample XSS attack
Normal URLs and JavaScript code
Analysis of URLs & JavaScript code
Identification of features
Feature value extraction from collected URLs and JavaScript code
Combining feature values into a single arff file (Preparation of datasets based on URLs and JavaScript code)
Performance analysis of Machine Learning algorithms (Using Naive Bayes, Support Vector Machines & J48 Decision Tree)

Figure 1: Proposed System

3.1 Web Page Collection

The web pages collected for the preparation of dataset consists of normal web pages and pages with malicious URL and Java Script. The XSSed [18] project maintains the largest online XSS vulnerable archive. Sufficient vulnerable URLs and script can be obtained from the mirror directory of XSSed. About 44,264 mirror pages were collected from XSSed mirror directory and these pages contribute to the XSS category.

Normal web pages were collected from the open directory project Dmoz [19] which is the largest human edited web directory and ClueWeb09 warc files, which is maintained by Lemur Project [20]. The pages collected from Dmoz consists of 19,646 web pages and the web pages in ClueWeb09 warc files consist about 296 web pages and these two sets of web pages contribute to the non XSS category.

3.2 Feature Identification

For the identification of the features, the normal and malicious URLs and script were programmatically extracted from the collected web pages and analyzed. The features were identified, the feature values were computed for every URL and script extracted from the normal and malicious web pages, and these values were used for the preparation of the dataset. For this work, we have used seven features for the URLs and five features for the JavaScript.

3.2.1 URL Based Features

1. Number of characters: It represents the number of characters of the extracted URL. The URL that belongs to XSS category usually contains JavaScript code. So its length will be larger than normal URL. It is considered in most of the attacks which are related with URL [10].
2. Presence of duplicate characters: It corresponds to the presence of duplicate characters like << and >> in the URL. Duplicate characters are used as part of user input to defeat certain anti XSS filters [11].
3. Presence of special characters: It corresponds to the character combination which is used as the prefix with script input to make the script code executable. These special characters includes ", "">, ">, "/> etc.
4. Presence of script tags: The URL which are used for implementing attack usually contain the direct JavaScript code for redirecting, cookie access etc. In such case, they are embedded within the opening and closing script tags. So the presence of script opening and closing tags in the URL can be considered as a useful feature.
5. Request for cookie: It corresponds to the checking of document.cookie function. If there is any request for cookies in the URL, the probability that the URL contains XSS attack is more.
6. Redirection: URLs are checked for the presence of functions window.location, window.history.back, window.navigate etc. Redirecting functions are used by attacker to redirect the victims to malicious web pages.
7. Number of keywords: These are the words which usually appear in the URL as the variable names for assigning user input values or for the spread of phishing attacks [10]. Some of them include login, signup, contact, search, query, redirect etc.

3.2.2 JavaScript Based Features

1. Number of characters: Since normal JavaScript contains multiple lines of code for manipulating user inputs, its length will be greater than that of XSS URLs which simply contains the direct script code for redirect or cookie access.
2. Number of script functions: It corresponds to the number of JavaScript functions used in the script code. Naïve code usually uses more than one JavaScript function for the manipulation of user inputs. So the number of JavaScript functions will be more in naïve script code.
3. Reference to JavaScript file: Normal script code refer to external script files more frequently whereas XSS script code uses script functions directly for implementing the attack.
4. Presence of user-defined functions: User defined functions are used in normal script code for convenience and reusability. Its presence can be taken as a feature for distinguishing normal and attack code.
5. Request for cookie: Script code for accessing and sending cookie information to other websites can be taken as an indication of information stealing and XSS attack.

4. RESULT EVALUATION

4.1 Dataset Evaluation

The dataset prepared from the extracted URLs consists of 8 attributes which includes seven URL feature values and one attribute for representing XSS, non-XSS class values. Similarly for JavaScript dataset, 6 attributes were used which includes 5 feature values and one attribute for representing XSS, non-XSS class values. The dataset for URLs consists of about 43,579 XSS instances and 13, 14,146 non XSS instances and the dataset for Java Scripts consists of about 32,256 XSS instances and 60,115 non XSS instances.

4.2 Classifier and Result Evaluation

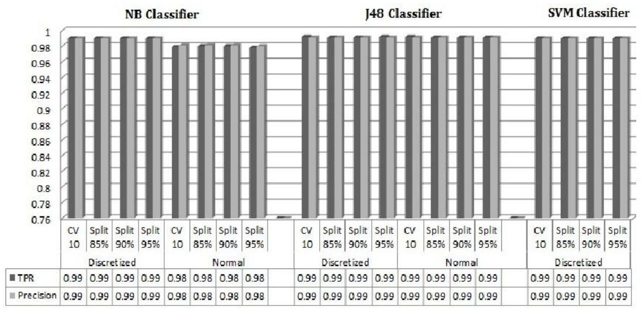
Three machine learning algorithms were used for analyzing the performance of the two datasets and they are Naïve Bayes (NB), Support Vector Machine and J48 Decision Tree algorithms. Naïve Bayes classifier is based on Bayes theorem which is used in statistics and probability theory. The basic idea of NB is that each observed sample in the training set can incrementally increase or decrease the probability that a hypothesis occur [13]. Bayes theorem calculates the probability of a hypothesis (posterior probability) based on the prior probability and class likelihoods.

Support Vector Machine is a method for linear classification and regression. In SVM, the main aim is to find the optimal hyper plane that maximizes the margin. After training, the discriminant is written as a sum of training vectors that are selected as support vectors. The instances which are not support vectors and which lies inside the margin are discarded since they carry no information [14]. J48 Decision Tree is an open source Java implementation of C4.5 algorithm. Decision learning methods are most widely used for inductive inference. Here the learned function is represented by a decision tree or it can be considered as a set of if-then rules. Each node in the decision tree represents attributes and the branch descending down represents value of the attribute. Decision tree construction is based on the entropy and information gain associated with each attributes [14].

The selected algorithms were applied on the dataset using Weka tool [12]. We used 4 test modes for training the three classifiers. 10-fold cross validation and percentage splits with 85, 90 and 95 split percentages were used for the training and testing of the datasets. In the case of 10 fold cross validation; the given dataset is divided into 10 sets each having equal number of both positive and negative samples and training is performed using 9 samples. For testing the performance of classifier, remaining one set is used. In the case of splitting, the percentage value for split can be specified. The specified percentage of instances from dataset is used for training the classifier and the remaining part is used for testing.

A number of test runs were run on these datasets by applying different filtering methods and customizing the parameters available for each classifier. The performance of each test run under each classifiers were compared based on the number of correctly and incorrectly classified instances, True Positive Rate (TPR), False Positive Rate (FPR), Precision and the time taken to build the model.

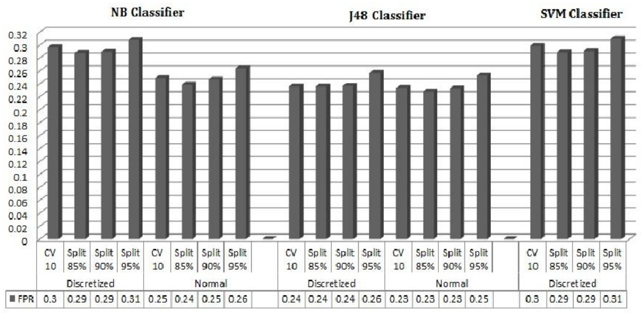
The performance of three classifiers for the XSS-URL dataset is shown in Figure 2 and Figure 3 and Table 1. In the case of XSS-URL dataset, the NB classifier, CV10 took more time than other split methods for the discretized attributes. But for normal attribute inputs, 95% split took more time to build classifier model. While comparing the TPR and Precision values, discretized attributes gave slightly higher values than using normal attributes. Lower FPR was obtained with normal attributes.



	NB Classifier								J48 Classifier								SVM Classifier			
	Discretized				Normal				Discretized				Normal				Discretized			
	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%
▪ TPR	0.99	0.99	0.99	0.99	0.98	0.98	0.98	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
▪ Precision	0.99	0.99	0.99	0.99	0.98	0.98	0.98	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99

Figure 2: Performance on XSS-URL dataset based on TPR and Precision

For the J48 classifier, the highest time requirement for building the model was for 90% split with normal attributes whereas the discretized attributes with 95% split test mode executed very quickly. Both types of attributes gave almost same values for TPR and Precision but normal attributes yield lower false positive values.



	NB Classifier								J48 Classifier								SVM Classifier			
	Discretized				Normal				Discretized				Normal				Discretized			
	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%
▪ FPR	0.3	0.29	0.29	0.31	0.25	0.24	0.25	0.26	0.24	0.24	0.24	0.26	0.23	0.23	0.23	0.25	0.3	0.29	0.29	0.31

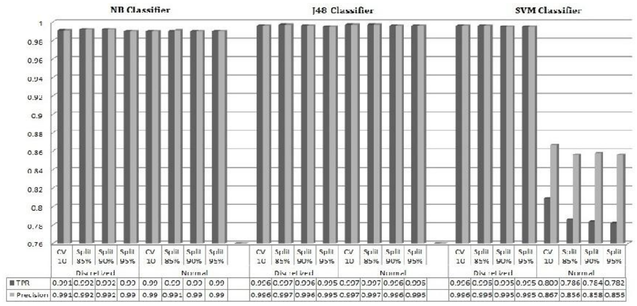
Figure 3: Performance on XSS-URL dataset based on FPR

Result for the execution for discrete attributes could not be collected for SVM since it caused error during the execution due to the hardware limitations of the system which was used for experimentation. The TPR and Precision values of SVM were almost similar to that of NB and J48, but the FPR values were high for the normal attribute inputs.

Table 1: Time Requirement for the three classifiers on the XSS-URL dataset

Filtering Method Applied	Test Mode	Time Taken To Build Model (in sec)		
		NB	J48	SVM
Discretized Attributes	CV 10	3.79	6.44	79.79
	Split 85%	0.25	6.83	88.23
	Split 90%	0.55	6.58	84.77
	Split 95%	0.27	6.38	80.42
Normal Attributes	CV 10	1.23	23.74	-
	Split 85%	1.14	23.71	-
	Split 90%	1.11	28.03	-
	Split 95%	1.28	23.7	-

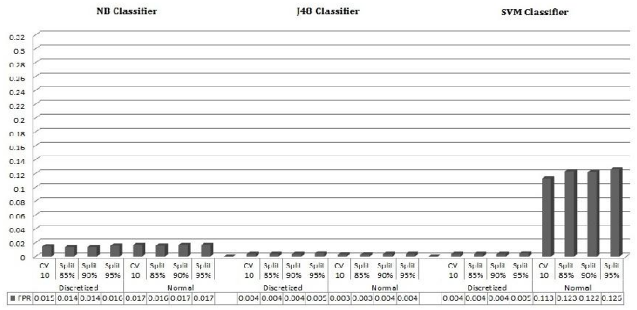
Refer Figure 4, Figure 5 and Table 2 for the results obtained for XSS-JavaScript dataset. For the XSS-JavaScript dataset, the NB classifier, discretized attributes gave slightly better results than that of normal attributes, in terms of TPR and Precision. FPR value was slightly lower when using discretized attributes. Also, the execution completed within a few seconds for all the test modes.



	NB Classifier								J48 Classifier								SVM Classifier							
	Discretized				Normal				Discretized				Normal				Discretized				Normal			
	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%
▪ TPR	0.991	0.992	0.992	0.99	0.99	0.99	0.99	0.99	0.996	0.997	0.996	0.995	0.997	0.997	0.996	0.995	0.996	0.995	0.995	0.995	0.809	0.786	0.784	0.782
▪ Precision	0.991	0.992	0.992	0.99	0.99	0.991	0.99	0.99	0.996	0.997	0.996	0.995	0.997	0.997	0.996	0.995	0.996	0.995	0.995	0.995	0.867	0.856	0.858	0.855

Figure 4: Performance on XSS-JavaScript dataset based on TPR and Precision

For the J48 classifier, the execution time for normal attributes was low. It could get the same values for both TPR & Precision as that of using discretized attributes. Its FPR was lower than the other two algorithms. The SVM classifier with normal attributes showed noticeable performance loss compared to using discretized attributes, with respect to TPR and Precision. FPR was higher than any other algorithm. Also, the execution time for normal attribute input was high.



	NB Classifier								J48 Classifier								SVM Classifier							
	Discretized				Normal				Discretized				Normal				Discretized				Normal			
	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%	CV 10	Split 85%	Split 90%	Split 95%
▪ FPR	0.015	0.014	0.014	0.016	0.017	0.016	0.017	0.017	0.004	0.004	0.004	0.005	0.003	0.003	0.004	0.004	0.004	0.004	0.004	0.005	0.113	0.123	0.122	0.125

Figure 5: Performance on XSS-JavaScript dataset based on FPR

Table 2: Time Requirement for the three classifiers on the XSS-JavaScript dataset

Filtering Method Applied	Test Mode	Time Taken To Build Model (in sec)		
		NB	J48	SVM
Discretized Attributes	CV 10	0.05	0.23	32.42
	Split 85%	0.02	0.11	32.06
	Split 90%	0.02	0.13	31.78
	Split 95%	0.02	0.11	31.76
Normal Attributes	CV 10	0.17	0.17	84.74
	Split 85%	0.06	0.06	58.33
	Split 90%	0.05	0.05	83.41
	Split 95%	0.05	0.05	82.99

5. CONCLUSION AND FUTURE WORK

While comparing the overall performance of all the 3 classifiers on XSS-URL and XSS-JavaScript, J48 was on par with other algorithms with respect to TPR & Precision, whereas it gave comparatively better performance with respect to FPR. But its time requirement for model creation was little higher compared to that of NB classifier but much less than SVM. Hence, J48 with discretized attributes can be used for the identification of XSS attack based on URL and JavaScript.

The scope of this work can be extended by identifying more features and adding them to the currently created dataset. In this work, only three machine learning algorithms were used for analysis. The experiment can be repeated by using other machine learning algorithms. Since archives with phishing and spam data are available online, similar study can be done on phishing or spam classification and can be used for extending the capability of the current dataset in representing multiple attack domains.

6. REFERENCES

1. OWASP. 2013. *Top 10 2013-Top 10 – OWASP*. https://www.owasp.org/index.php/Top_10_2013-Top_10.
2. XSSED. *What is XSS? XSSed.com*. http://www.xssed.com/xssinfo.
3. OWASP. *Cross-site Scripting (XSS) – OWASP*. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
4. OWASP. 2014. *XSS (Cross Site Scripting) Prevention Cheat Sheet – OWASP*. XSS (Cross Site Scripting) Prevention Cheat Sheet – OWASP.
5. D. Flanagan. JavaScript: *The Definitive Guide*. 6th Edition, O'reilly.
6. Ruderman, J. 2001. *JavaScript Security: Same Origin*. www-archive.mozilla.org/projects/security/components/same-origin.html.
7. Giorgio Maone. *NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience*. http://noscript.net/
8. Frenz, C.M., Yoon, J. P. 2012. *XSSmon: A Perl Based IDS for the Detection of Potential XSS Attacks*. In Systems. Applications and Technology Conference (LISAT). IEEE.
9. Kirda, E., Kruegel, C., Vigna, G., Jovanovic, N. 2006. *Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks*. In 21th ACM Symposium on Applied Computing. ACM.
10. Nunan, A.E., Souto, E., Santos, E.M., Feitosa, E. 2012. *Automatic Classification of Cross-Site Scripting in Web Pages Using Document-based and URL-based Features*. Symposium on Computers and Communications (ISCC). IEEE
11. Common Attack Pattern Enumeration and Classification. 2010. *CAPEC-245: Cross-Site Scripting Using Doubled Characters*. http://capec.mitre.org/data/definitions/245.html.
12. Bouckaert, R.R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., Scuse, D. 2013. *WEKA Manual for Version 3-6-10*.
13. Ethem Alpaydın. 2010. *Introduction to Machine Learning*. Second Edition. The MIT Press.
14. Tom M. Mitchell. 1997. *Machine Learning*. McGraw-Hill.
15. Shar, L.K., Hee Beng Kuan Tan. 2012. *Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities*. In 35th International Conference on Software Engineering (ICSE).
16. Shar, L.K., Hee Beng Kuan Tan, Briand, L.C. 2013. *Mining SQL Injection and Cross Site Scripting Vulnerabilities using Hybrid Program Analysis*. In 35th International Conference on Software Engineering (ICSE).
17. OWASP. *DOM Based XSS - OWASP*. https://www.owasp.org/index.php/DOM_Based_XSS.
18. XSSED. *XSSed Cross Site Scripting (XSS) attacks information and archive*. http://www.xssed.com.
19. *DMOZ - the Open Directory Project*. http://www.dmoz.org/.
20. *Lemur Project Home*. http://www.lemurproject.org/.