Text-Mining based Predictive Model to Detect XSS Vulnerable Files in Web Applications

Mukesh Kumar Gupta *, Mahesh Chandra Govil †, Girdhari Singh ‡
Department of Computer Science & Engineering
Malviya National Institute of Technology, Jaipur-302017, Rajasthan, INDIA
Email: *mukeshgupta@skit.ac.in, †govilmc@yahoo.com, ‡girdharisingh@rediffmail.com

***Abstract*—This paper presents a text-mining based approach to detect cross-site scripting (XSS) vulnerable code files in the web applications. It uses a tailored tokenizing process to extract text-features from the source code of web applications. In this process, each code file is transformed into a set of unique text-features with their associated frequencies. These features are used to build vulnerability prediction models. The efficiency of proposed approach based model is evaluated on a publicly available dataset having 9408 labelled source code files. Experimental results show that proposed features based best predictive model achieves a true average rate of 87.8% with low false rate of 12.3% in the detection of XSS vulnerable files. It is significantly better than the performance of existing text-mining approach based model that achieves a true average rate of 71.6% with false rate of 33.1% on the same data set.**

***Keywords*—web application security, cross-site scripting vulnerability, text-mining, text features, prediction model, unigram**

I. Introduction

The development of secure web applications is very resource intensive and has become the most critical issue in the software engineering [1]. The vulnerability prediction model detects the possible vulnerable code files in the web applications. It helps to save the resources of software companies by concentrating on the code segment predicted to be vulnerable and results in production of secure web applications. The recent security statistical reports reveal that approximately 55% of assessed web applications have a security vulnerability [2]. In 2013, Open Web Application Security Project (OWASP) [3] and Common Vulnerabilities and Exposures (CWE) [4] reported Cross-Site Scripting (XSS) as one of the most serious security vulnerabilities in web applications.

Cross-site scripting (XSS) is an application-level code injection type security vulnerability. It occurs, whenever a server program (i.e. dynamic web page) code uses an unrestricted input via HTTP request, database, or files in the response generator statement (i.e. output statement) without any validation. The main cause of XSS vulnerabilities is the weakness present in the source code of the web application. It may be due to the weaknesses of the programming language, improper input validations, or ignorance of security guidelines by developers. This weakness allows a malicious user to steal sensitive information (i.e. cookie, session) and performs other malevolent operations.

978-1-4673-6540-6/15/$31.00 ©2015 IEEE

Researchers have proposed various approaches based on static and dynamic analysis to detect XSS vulnerabilities in source code of web applications. Static analysis based detection approach uses a set of predefined rules to detect vulnerabilities in source code without executing it. These approaches are easy to implement, but produce too many false positive results. Dynamic analysis based approach uses the complex analysis to produce more accurate results. However, these approaches require large set of test cases to ensure any false negative results. False positive means approach detects a vulnerability whereas actually no vulnerability is present. In case of false negative, an approach claims to identify a certain class of vulnerabilities and does not detect a vulnerability, where one is present. The false negative results are more dangerous because these lead to a false sense of security.

Recently, machine-learning based vulnerability prediction models [5] [6] [7] [8] are gaining popularity in web security space, as these models provide an efficient way to identify vulnerable code from the application source codes. Actually, these models provide probabilistic remarks of vulnerable source code segments. Further, it helps to save the efforts of developer by concentrating on the code segment predicted to vulnerable. Existing XSS vulnerability prediction approaches have many limitations (e.g. not working for object-oriented code, too many false results), which are discussed in details in the related work section of this paper. To minimize those limitations, this paper presents a novel approach to extract text-features from the source code files and use them to build vulnerability prediction models.

The rest of the paper is organized as follows. Section 2 discusses prior prediction works related to the detection of the cross-site scripting vulnerability in web applications. Section 3 describes the extraction and selection of the best text-features to build vulnerability prediction model. Section 4 provides the details of the data set, experimental setting, and performance measures, which are utilized to evaluate the performance of the proposed approach. Section 5 discusses the experimental results and compares the proposed approach with existing text-mining approach. Finally, Section 6 concludes the paper noting and mentions future research directions.

II. Related work

Walden et al.(2014) [8] pointed out that vulnerability prediction research area is not as mature as other related research fields e.g. fault, defect prediction. It results in a limited number of research papers on vulnerability prediction are present in the literature. Chowdhury et al.(2011) [9] used complexity,

cohesion and coupling metrics for predicting vulnerability-prone files in Mozilla Firefox. Similarly, Shin et al. (2011) [10] utilized code complexity, code churn, and developer activity metrics for discriminating general vulnerable files from benign ones. They investigated that the complex code programs are more prone to vulnerability and, predicated 80% known vulnerable files with less than 25% false positives.

In contrary, Shar et al.(2012) [11] claimed that simple and tiny code program has many XSS vulnerabilities, which resembles with our observations. Therefore, vulnerability prediction models based on those metrics are not efficient to predict XSS vulnerability. Authors [7] considered that the use of user-input in the output-statement without any validation is the main reason of XSS vulnerabilities. They extracted input, output, validation and sanitization code constructs through static and dynamic analysis. Further, they classified these code constructs in various categories and considered as features to build machine-learning models for identification of vulnerable statements. They have used an open source Pixy tool for taint analysis, but the tool itself has many limitations. First, the Pixy tool is unable to perform the taint-analysis of object-oriented code, so their approach cannot work with object-oriented code. Second, PHP 5.0 has around 3000 functions, but their approach considered only 300 built-in functions and not considered many functions(e.g. floatval), whose presences can prevent XSS attacks. Due to this, their approach is not working with latest version of PHP web applications.

Text-mining based feature sets are basic feature set that are extensively used in literature to build prediction models to solve problems in different domains (i.e. sentiment classification, fault prediction & defect prediction) but have not been used much to build XSS vulnerability prediction model. Recently, Scandariato et al.(2014) [6] proposed first text mining based machine-learning models for predicting vulnerable files from the source code of software applications. They considered source code as text and characterized each source code file as a term frequency vector. In [8], Walden et al.(2014) compared the software metrics and text mining features (i.e. unigram) based approaches and observed that text-mining features provide significantly better performance in prediction of XSS vulnerabilities. Their approach does not have any limitations in terms of the type of code (e.g. object-oriented code) as well as the source language version. We have observed many cases for which feature vectors are same for vulnerable and non-vulnerable files. Due to this, their approach has limitations in term of false negative and false positive results. Our approach is an extension of their works.

III. Proposed vulnerability prediction approach

The proposed approach proceeds as follows. Each sample code file is transformed into a corresponding token file through the proposed tokenizing process. Then, each source code file is characterized as a set of text-features with their associated frequencies. Further, a feature selection method is applied to extract most relevant features and use them to build vulnerability prediction models. Figure 2 depicts the steps, which are followed in the proposed approach to classify vulnerable source code file from benign one. The following subsections illustrate feature extraction and feature selection process to extract relevant features for building XSS prediction models.

A. Feature Extraction

Feature extraction is an important and essential step for building the predictive models. The proper set of features provides better performance of predictive models. As mentioned earlier, the XSS vulnerability occurs whenever use-input is referenced in an output-statement without any validation. Hence, the features that represent input, output and user-input validation routines are important for vulnerability prediction. The proposed approach uses a tailored tokenizing process to extract and build text-feature sets from the source code files. It begins by tokenizing each PHP source file into a set of tokens. Each token has a token-id, token name and string which represents the language code construct. Then, each token is pre-processed for removal of PHP commented lines, constant strings and pure HTML codes. These source codes are irrelevant for vulnerability prediction, as they do not contribute any meaningful information for discriminating vulnerable code from benign ones. The rest tokens are processed as follows.

1) **Input Related Features:** The user-input is the main source for exploiting XSS vulnerabilities as the malicious-user can insert malicious code only via direct or indirect sources. Therefore, the code constructs used to implement input access logics are important for characterizing vulnerable code files. In the PHP code, external inputs are received via. global variables (e.g. $_GET, $_COOKIE, $_POST etc), database functions (i.e. my_sql_fetch_array) or file built-in functions(i.e fgets). These codes are represented by T_VARIABLE and T_STRING tokens in tokenized file. For these tokens, corresponding token strings are included in our feature set. For user-defined local variables their token names (i.e. T_VARIABLE) are considered in feature set, as these variables may have different names, but do not provide any difference in vulnerability point of view. The existing approach treated all variables(i.e. global or local variables) in same manners and included T_VARIABLE along with variable names in the feature set.
2) **Output Related Features:** Output statements use external input to generate the dynamic response. In tokenized file, the output statements are represented by T_ECHO, or T_PRINT token names. Our feature set contains these token names without their string values.
3) **Sanitization Related Features:** In PHP, some inbuilt functions are able to prevent XSS attacks. We have analyzed and found various PHP inbuilt functions (i.e. string manipulation, encryption, escape, input validation etc) or standard sanitisation functions (i.e. htmlspecialchars, htmlentities etc), which work as a sanitisation function.These functions either minimize the probability of XSS vulnerability or in some cases completely remove XSS vulnerability. T_STRING token are generated in the token file corresponding to each function. These functions have different capabilities in term of sanitisation of user input, so our approach considered token string correspond to each T_STRING token. Existing text mining approach [8] considered all function and identifier name in the

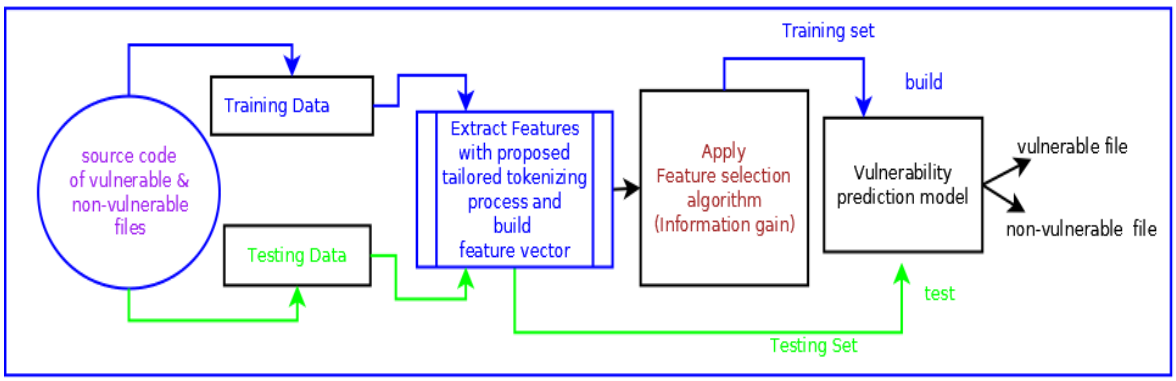

Fig. 1. Flow graph of proposed vulnerability prediction approach

same ways and extracted only one feature T_STRING for many code constructs, which produce an ambiguity in discriminating the vulnerable code from benign ones.

4) **Context Related Features:** Some PHP statements contain HTML tags or HTML statements contain PHP codes in the form of constant string to produce interactive web application. These code constructs indirectly represent an HTML context (e.g. style, script etc.) and require special filters to prevent XSS attacks. In the tokenizing process, these code constructs are represented by T_CONSTANT_STRING and T_INLINE_ HTML tokens respectively. We have considered only those HTML statements in which PHP code is present and ignored other statements. As well as, in the case of T_CONSTANT_ENCAPSED_STRING token, only those token strings are considered, in which either HTML opening or closing tag is present. Further, a set of delimiters (e.g. ” ” = :) are used to get text words presents in the corresponding token strings and included in our feature set.

5) **OOPS and other features:** For other tokens, we have considered only token name in our feature set without their string values.

Example

Consider the source code program given in figure 2, to highlight the difference between propose a feature extraction approach with existing text-mining based XSS prediction approaches. In this code, the user-input is validated through *htmlspecialchars* function and make line 6 as a non-vulnerable statement. In line 7, HTML tag is present in the PHP code and in line 9, HTML statement contains PHP code. These combinations reveal the HTML contexts and in-spite the use of PHP built-in function(i.e. htmlspecialchars) these statements are vulnerable to XSS attacks. For example, the attack vector presents in figure 2 can exploit the XSS vulnerability. These statements require special filters to prevent XSS attacks. Some of the extracted features with their frequencies by Walden's approach [8] (F1) are :

T_INLINE_HTML: 3, T_OPEN_TAG: 2, T_VARIABLE($tainted) :3, T_STRING:2, T_VARIABLE($data): 4, ...

```
1.<html>
2.<body>
3.<?php
4.$tainted=$_GET['UserData'];
5.$data=htmlspecialchars($tainted,ENT_QUOTES);
6.echo "Hello $data";
7.echo"<span style=color:".$data.">Hey</span>";
8.?>
9.<h1 style=color:<?php echo $data;?>>Hi</h1>
10.<h1>Hello World!</h1>
11.</div>
12.</body>
13.</html>

Attack Vector:
UserData="green onmouseover= alert(/Meow!/)"
```

Fig. 2. : Code listing of XSS vulnerable program

In the existing text-mining approach, the user-defined variable names are considered as a different feature that are not useful from vulnerability point of view. Also, T_STRING feature is considered for all strings (e.g. ENT_QUOTES, htmlspecialchars, identifier etc) in their feature set. However, ENT_QUOTES is a parameter and 'htmlspecialchars' is a sanitization function in PHP language, but both are considered in the same category. In addition, no discrimination is present in the feature vector for different HTML codes . In contrary, propose approach extracts following features (F2):

T_OPEN_TAG: 2, T_VARIABLE: 6, $_GET: 1, htmlspecialchars:1, ENT_QUOTES:1, T_ECHO: 2, <span: 2, style: 1, color: 1, ...

In the proposed approach, all user-defined variables are treated equally and global variables are treated differently. For each T_STRING token corresponding token-string is considered in our feature set. In the sample code, the "¡span style= color:" string represents a "style" HTML context. The existing approach has not considered their effect, which is very important for vulnerability prediction. Table 1 summarizes the extracted text-features corresponding to each source code line listed in figure 2 through different approaches.

TABLE I. COMPARISON OF EXTRACTED UNIGRAM FEATURES

Code Line Number	Walden Unigram Features (F1)	Proposed Unigram Features (F2)
1	T_INLINE_HTML	<html >
2	-	<body >
3	T_OPEN_TAG	T_OPEN_TAG
4	T_VARIABLE($tainted),=, T_VARIABLE($_GET), [,],;	T_VARIABLE, $_GET, T_CONSTANT_ENCAPSED_STRING
5	T_VARIABLE($data),=, T_STRING,(, T_VARIABLE($tainted), T_STRING,),;	T_VARIABLE, htmlspecialchars, T_VARIABLE ,ENT_QUOTES
6	T_ECHO, T_ENCAPSED_AND_WHITESPACE, T_VARIABLE($tainted), ;	T_ECHO, T_ENCAPSED_AND_WHITESPACE, T_VARIABLE
7	T-ECHO, ., T_VARIABLE($data),.,	T_ECHO, <span, style, color, T_VARIABLE, T_CONSTANT_ENCAPSED_STRING
8	T_CLOSE_TAG	T_CLOSE_TAG
9	T_INLINE_HTML, T_OPEN_TAG, T_ECHO, T_VARIABLE($data), ;, T_CLOSE_TAG	<h1, style, color, T_OPEN_TAG,T_ECHO, T_VARIABLE,T_CLOSE_TAG, </h1>
10	T_INLINE_HTML	<h1 ></ h1 >
11	-	
12	-	</body>
13	-	</html>

B. Feature Selection

Researchers have applied various feature selection techniques to select the important features from the large size feature vector. It eliminates irrelevant and noisy features to improve the performance of predictive models. In this paper, we extracted prominent features with Information Gain feature selection techniques and constructed ProminentIG (F3) feature set.

IV. DATASET AND SETTINGS

In this section choice of data set, experimental setting and performance measures used to evaluate the predictive model are discussed.

A. Data for Experiment

We have used a publicly available GIT repository [12] that contains a synthetic test case generator. The available dataset contains 9408 samples written in PHP. It has 5600 safe and 3808 unsafe samples that are organized into different categories. Evaluation of the proposed method is performed on this dataset, as it provides various cases to evaluate the efficiency of predictive models. This dataset is available for free, open-source and contain a set of PHP source code with their vulnerability labels. It also contains the cases for object oriented source code that can be helpful to evaluate the performance of the proposed methods for object-oriented code. This dataset is better as compared to other existing data sets for evaluating the proposed approach. The other existing vulnerability data set repositories are Bugzilla, CVE , NVD etc. These contain only vulnerability information and do not provide source code to extract vulnerability prediction features. Therefore, these are inadequate to evaluate our proposed approach. Next, NIST (National Institute of Standard and Technology) dataset is also available publicly to evaluate vulnerability prediction methods, but it contains only limited samples for PHP source code (i.e. 80) which is insufficient to build an efficient vulnerability predictive model.

B. Experimental Setting:

The Ten-fold cross validation technique is used to evaluate the performance of the proposed approach. We randomly divide the data set into 90% training and 10% testing programs, such that both the sets are disjoint. We repeat all the experiments 10 times with randomly selected training and testing sets, and final performance is reported by average of the results.

C. Prediction performance measures

In this paper, we have extracted features for each data set sample and built different machine-learning models (i.e. predicator) with a machine-learning weka tool [13]. It is an open source, platform-independent, and freely available tool, which include the implementation of different machine-learning algorithms for data mining and machine learning experiments [14]. Similar to related prediction approaches, we have used several performance measures such as TPR, TNR, FPR and FNR to evaluate the predictive performance. The false positive represents the number of actual non-vulnerable files wrongly predicated as vulnerable, and true positive represents the number of actual vulnerable files correctly predicated as vulnerable by our predictive model. The true negative represents the number of actual non-vulnerable files correctly predicated as non-vulnerable and false negative represents the number of actual vulnerable (i.e. unsafe) files wrongly predicated as non-vulnerable by our predictor. The effective predictive model should have high values of TPR and TNR with low values of FNR and FPR, to assure that the model can correctly discriminate between vulnerable and non-vulnerable source code files.

V. RESULTS AND DISCUSSION

The results of various approaches with respect to evaluation measures such as TPR, TNR, FPR and FNR have shown in Table 3. The proposed approach outperform than existing text-mining approach due to the reason that the proposed approach contain the effect of HTML code in PHP statement

TABLE II. EVALUATION OF VARIOUS FEATURE SET WITH DIFFERENT MACHINE LEARNING CLASSIFIERS

Classifier	Feature Set	TP Rate	FN Rate	TN Rate	FP Rate	Average True Rate	Average False Rate
NB	Walden Feature(F1)	39.4	60.6	81.8	18.2	64.7	43.4
	Proposed Feature(F2)	77.4	22.6	64.7	35.3	69.8	30.2
	ProminentIG(F3)	78.6	21.4	65.8	34.2	70.9	29.1
JRip	Walden Feature(F1)	53	47	81.1	18.9	69.7	35.6
	Proposed Feature(F2)	57.6	42.4	94.5	5.5	79.6	20.4
	ProminentIG(F3)	76.6	23.4	90.8	9.2	85.1	14.9
J48	Walden Feature(F1)	57.8	42.2	81.7	18.3	71.6	33.1
	Proposed Feature(F2)	83.7	16.6	90.6	9.4	87.8	12.3
	ProminentIG(F3)	85.1	14.9	87.7	12.3	86.6	13.4
Random Tree	Walden Feature(F1)	52.6	47.4	80.8	19.2	69.4	36
	Proposed Feature(F2)	72.5	27.5	77.9	22.1	75.7	24.3
	ProminentIG(F3)	79.6	20.4	85.9	14.1	83.3	16.6
Bagging	Walden Feature(F1)	56.9	43.1	81.1	18.9	71.3	33.3
	Proposed Feature(F2)	84.1	15.9	89.6	10.4	87.4	12.6
	ProminentIG(F3)	85.4	14.6	89.8	10.2	88	11.9

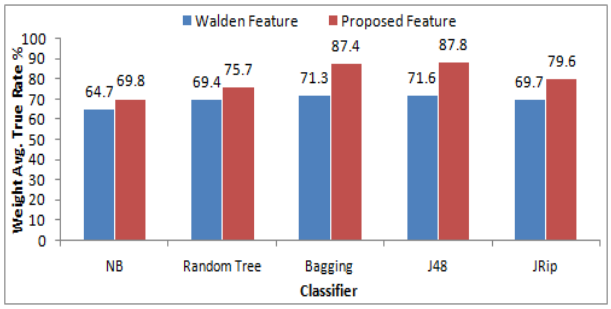

Fig. 3. Avg. True Result v/s Classifier for different Feature Set

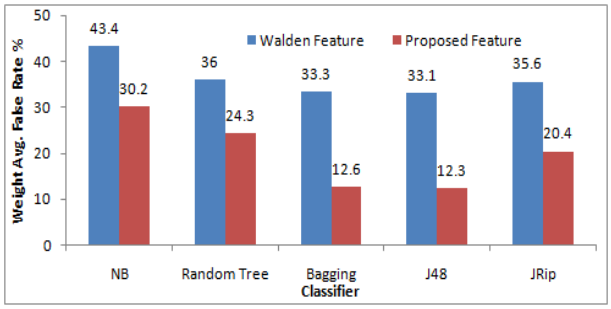

Fig. 4. Avg. False Result v/s Classifier for different Feature Set

or PHP code in an HTML statement. These code combinations indirectly represent an HTML context and considerably important information for vulnerable code discrimination. Existing approach considered unique tokens of source code files as feature to develop the machine learning models, however, these features are not sufficient as explained in the example in figure 2. To extract best features, we have applied Information Gain(IG) feature extraction method, but did not observe major performance difference of feature selection on various classifiers. Therefore, we have compared the performance of the models with all extracted attributes.

Predictive models are prepared using various classifiers as NB, Bagging, Random Tree, J48, JRip classifiers. Experimental results indicate that J48 classifier performs better as compared to other classifiers for existing and proposed feature. However, it is very close to bagging classifier as shown in figure 3.

J48 performs better than other classifier because it reduces the effect of attributes with low information gain i.e. irrelevant attributes. J48 classifier produces the best average true rate of 87.8% with proposed features, which is best among other classifiers i.e. 69.8 %, 79.6%, 75.7%, and 87.4% respectively, for NB, JRip, Random Tree, and Bagging classifiers (results as shown in figure 3). Figure 4 compares average false result with various classifiers for different feature sets. It shows J48 out performs the other classifiers. Bagging classifier produces the comparable results with the J48 classifier for proposed features and produces better results as compared to NB, random tree and JRip classifiers with the proposed feature set.

VI. CONCLUSIONS AND FUTURE WORK

Vulnerability detection is an important task in securing the web applications before and after their release. Insecure web applications may cause of stealing personal and crucial user information. This paper presented an approach for identification of vulnerable source code file through text-mining based predictive models. We have compared our approach with existing ones on a publicly available labelled data set. Experiment results show that the proposed features can detect vulnerable source code files in web applications with high TPR and TNR with low FNR and FPR.

In addition, it has been realized that the text-mining based approach does not depend on programming language version and work with object-oriented code, which shows the superiority than other existing XSS vulnerability detection approaches. In the future, we wish to analyze the effect of N-gram analysis on vulnerability detection results.

REFERENCES

[1] 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.

[2] WhiteHat Security. Web statistics report. https://whitehatsec.com/categories/statistics-report, 2014. Accessed: 2014-11-16.

[3] Dennis Groves Mark Curphey. Open web application security project. https://www.owasp.org/index.php/Main_Page, 2015. Accessed: 2015-04-22.

[4] CVE Editor Board. Common vulnerabilities and exposures (the standard for information security vulnerability names). http://cwe.mitre.org, 2014. Accessed: 2014-07-13.

[5] 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.

[6] 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, 2014.

[7] 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.

[8] 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, 2014.

[9] 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.

[10] 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, 2011.

[11] 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.

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

[13] Peter Reutemann Eibe Frank, Mark Hall and Len Trigg. Weka: Data mining tool. http://www.cs.waikato.ac.nz/ml/weka, 2015. Accessed: 2015-05-26.

[14] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.