International Journal of Security and Its Applications
Vol.9, No.10 (2015), pp.231-240
http://dx.doi.org/10.14257/ijsia.2015.9.10.21

Evaluating Software Metrics as Predictors of Software Vulnerabilities

Mamdouh Alenezi and Ibrahim Abunadi

College of Computer & Information Sciences, Prince Sultan University
Riyadh, Saudi Arabia
malenezi@psu.edu.sa, iabunadi@psu.edu.sa

Abstract

Web application security is an important problem in today's Internet. A major cause of this is that many developers are not equipped with the right skills to develop secure code. Because of limited time and resources, web engineers need help in recognizing vulnerable components. A useful approach to predict vulnerable code would allow them to prioritize security-auditing efforts. In this work, we compare the performance of different classification techniques in predicting vulnerable PHP files and propose an application of these classification rules. We performed empirical case studies on three large open source web-projects. Software metrics are investigated whether they are discriminative and predictive of vulnerable code, and can guide actions for improvement of code and development team and can prioritize validation and verification efforts. The results indicate that the metrics are discriminative and predictive of vulnerabilities.

Keywords: *Web Security, Software Metrics, Vulnerability Prediction*

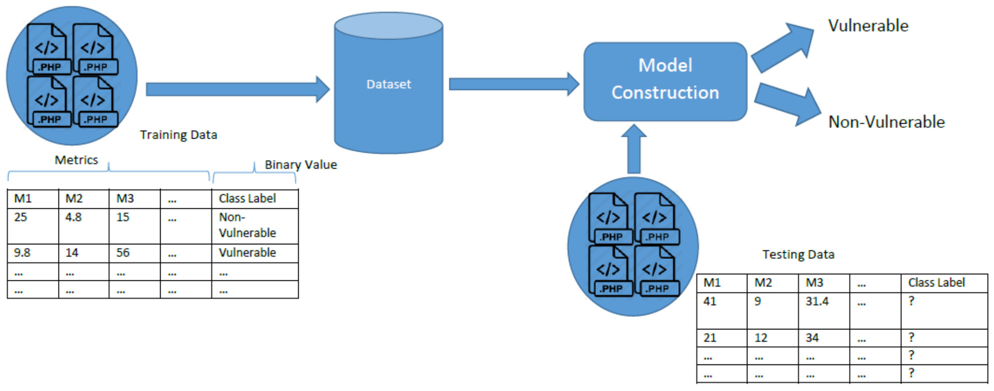
1. Introduction

Software systems have become an essential component of any critical infrastructure. The process of building secure software systems are expensive, difficult, and time-consuming [13]. Web applications are also very essential in our everyday activities for instance; they are used for communications, shopping, banking, social networking, etc. Since these web applications are highly accessible, their vulnerabilities have greater impact than vulnerabilities in other types of software. The main responsibility of the security of these applications lies with the developers. Unfortunately, most developers do not have the appropriate knowledge about secure coding [17] and most software vulnerability related studies have found that software complexity is the enemy of software security [16].

The security of most systems and networks depends up on the security of the software running on them. Most of the attacks on these systems occur because of exploitation of vulnerabilities found in these software applications. Finding and solving vulnerabilities in early stages of software composition is an important step. Building and distributing vulnerability prediction models can focus information assurance teams to spend their time and resources on the vulnerable parts of their code base.

If a single security attack succeeds, it can bring severe damage to organizations and people; especially it comprises confidential information or launches a denial of service attack. These security attacks usually happen when vulnerabilities in a system are exploited. Vulnerability in software is a weakness that allows an attacker to use the system for a malicious purpose. Detecting and finding these vulnerabilities early is very important in software engineering because it would help reduce the cost of development and help in preventing damage to the reputation of the software company. As such, methods and tools for software vulnerability prediction are invaluable. The argument for focusing on PHP is two-fold: more than twice as many open source web applications are written in PHP than Java [23], and PHP is known for its poor security reputation [24].

In this work, we compare the performance of different classification techniques in predicting vulnerable PHP files and propose an application of these classification rules. The remainder of this paper is organized as follows. Section 2 discusses software metrics. Section 3 states the proposed approach. Section 4 discusses the experimental evaluation. The empirical study is presented in Section 4. One suggested application of our study is presented in Section 5. Some threats to validity are presented in Section 6. Related work is discussed in Section 7. Conclusions are presented in Section 8.



M1	M2	M3	...	Class Label
25	4.8	15	...	Non-Vulnerable
9.8	14	56	...	Vulnerable
...	...	...	...	...
...	...	...	...	...

M1	M2	M3	...	Class Label
41	9	31.4	...	?
21	12	34	...	?
...	...	...	...	?
...	...	...	...	?

Figure 1. The Proposed Approach

2. Software Metrics

Software metrics [15] are measures utilized to evaluate the process or product quality. These metrics helps project managers to know about the progress of software and assess the quality of the various artifacts produced during development. The software analysts can check whether the requirements are verifiable or not. Software metrics are required to capture various software attributes at different phases of the software development. Software metrics can be utilized to adequately measure various phases of the software development life cycle.

When measuring a software development project, metrics can be divided into two different categories [15]: Process and Product metrics. Process metrics measure the process in which the product is developed. Product metrics measure the final outcome of following a given process or a set of processes. An example of product metrics are Object-oriented (OO) concepts such as coupling, cohesion, inheritance, and polymorphism can be measured using software product metrics. The process metrics can be used to measure the cost and duration of an activity, measure the effectiveness of a process, compare the performance of various processes, or improve the processes and guide the selection of future processes.

3. Approach

We deal with the vulnerability prediction problem as a classification problem in order to predict if a PHP file is vulnerable or not. A key element of prediction is the supervised model, which is a method of combining multiple metrics into a single binary classification prediction. In this work, we evaluate the performance of different classification techniques in predicting vulnerable PHP files. The proposed approach is illustrated in Figure 1. Vulnerability prediction can be used to automatically predict vulnerable files. To build and evaluate the prediction models, the dataset is divided into training and testing sets. The training data has the source code metrics and the class label (vulnerable or not) in order to train the model whereas the testing data has only the source code metrics without a class label. Supervised models require a training set and a validation set. To obtain unbiased evaluation results, we perform a 10-fold cross-validation [12]. Ten-fold cross validation performed by randomly partitioning the data into 10 folds, with each fold being the held-out test fold exactly once. Comparisons between classifiers are based on three measures namely Precision, Recall, F-Measure, and AUC (Area Under Curve). We train the prediction model based on the source code metrics of the PHP files and their class labels (vulnerable or not). After running the model on this training dataset, we test this model on a testing dataset and evaluate the effectiveness of this model in predicting vulnerabilities. Machine learning techniques (classifiers) are used to build prediction models to predict the vulnerabilities software files.

4. Experimental Evaluation

4.1. Data Set

In this study, we used a dataset collected by [24]. The dataset was collected and analyzed in the previous mentioned study. The data contains several software metrics and vulnerability information about their PHP files. The dataset selected is from the open source community. Open source software is usually considered secure since the community leverages large developer communities to detect and fix vulnerabilities in the code [18]. The applications in the dataset are Drupal, Moodle, and PHPMyAdmin. Drupal is a well-known content management system. Moodle is an open source learning management system. PHPMyAdmin is a web based management tool for the MySQL database. Table 1 shows a descriptive statistics about the dataset. Regarding the software metrics in this data, the following metrics were used in this dataset:

1. Lines of code: Number of lines in a PHP source. This includes PHP tokens, excluding lines without PHP tokens.
2. Lines of code (non-HTML): Same as lines of code, except HTML content embedded in PHP files.
3. Number of functions: Number of functions and method definitions in a PHP file.
4. Cyclomatic complexity: The control flow graph size after linear chains of nodes are collapsed into one node. These metrics are computed by adding one to the number of loop and decision statements in the PHP file.
5. Maximum nesting complexity: The maximum depth to which loops and control structures in the file are nested.
6. Halstead's volume: A volume estimate using the number of unique operators and operands and the number of total operators and operands in the file. For the purposes of this metric, operators are method names and PHP language operators, while operands are parameter and variable names.
7. Total external calls: The number of instances where a statement in the file being measured invokes a function or method defined in a different file.
8. Fan-in: The number of files that contain statements that invoke a function or method defined in the file being measured.
9. Fan-out: The number of files containing functions or methods invoked by statements in the file being measured.
10. Internal functions or methods called: The number of functions or methods defined in the file being measured which are called at least once by a statement in the same file.
11. External functions or methods called: The number of functions or methods defined in other files that are called at least once by a statement in the file being measured.
12. External calls to functions or method: The number of files calling a particular function or method defined in the file being measured, summed across all functions and methods in the file being measured.

Table 1. Descriptive Statistics about the Dataset

System	Version	Vulnerable Files	Total Files
Drupal	6.0	62	202
Moodle	2.0.0	24	2942
PHPMyAdmin	3.3.0	27	322

4.2. Classifiers

In this section, we briefly present the machine learning techniques used in this work.

1. **Naive Bayes Classifier**: Naive Bayes is a probabilistic classifier, which assumes that all features are independent. It finds the class with maximum probability given a set of features values using the Bayes theorem.
2. **Decision Trees Classifier**: Decision Tree is a classifier in the form of a tree structure. It is a predictive model that decides the dependent value of a new sample based on diverse attribute values of the existing data. Each internal node in the tree represents a single attribute while leaf nodes represent class labels. Decision trees classify each instance by starting at the root of the tree and moving through it until reaching a leaf node.
3. **Random Forests Classifier**: Random Forests is an ensemble learning method that generates several decision trees during training time. Each tree gives a class label. The Random Forests classifier selects the class label that has the mode of the classes output by individual trees.
4. **Ensemble (Voting)**: Voting is a combining strategy of classifiers [14]. Majority Voting and Weighted Majority Voting are the most popular methods of Voting. In Majority Voting, each ensemble member votes for one of the classes. The ensemble predicts the class with the highest number of votes. Weighted Majority Voting makes a weighted sum of the votes of the ensemble members, and weights typically depend on the classifiers confidence in its prediction or error estimates. We used the voting of the three classifiers noted above.

4.3. Evaluation Metrics

We evaluate the classification algorithms based on Precision, Recall, F-measure, and Area Under Curve (AUC) or ROC (Receiver Operating Characteristics) as [7] argues that AUC is the best measure to report the classification accuracy. Precision measures how many of the vulnerable instances returned by a model are actually vulnerable. The higher the precision is, the fewer false positives exist. Recall measures how many of the vulnerable instances are actually returned by a model. The higher the recall is, the fewer false negatives exist. F-Measure is the harmonic mean of Precision and Recall. In this study, we adopt a binary classifier, which makes two possible errors: false positive (FP) and false negative (FN). A correctly classified vulnerable class is a true positive (TP) and a correctly classified non-vulnerable class is a true negative (TN). The prediction performance measures used in our experiments are described as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.4. Results

In this subsection, we present the results obtained in this study. Table 2 shows the classification results of the three projects. In the Drupal project, the Ensemble (voting) achieved the best performance in all studied measures. The Random Forest classifier competed with very close measure to voting. Since the difference is not that big, we see Random Forest as a good fit since it takes less time than voting. In the PHPMyAdmin project, in terms of AUC, Ensemble achieved the best results. Again. the Random Forest classifier competed with very close measure to voting. In terms of Precision, Random Forest was the best classifier. Likewise, in the Moodle project, in terms of AUC, Ensemble achieved the best results with Random Forests in close competition. Naive Bayes achieved the best results in terms of Precision. We conclude that Ensemble is a better technique but it is not worth the additional effort since the values achieved not that higher than Random Forest. Since AUC is the most informative indicator of predictive accuracy in case of having binary classification [2], we conclude that Random Forest would be sufficient since it does not require a lot of run time [1].

We combined the three datasets to reveal whether there is a big difference between these projects and to gain an insight into the main metrics that are able to predict the vulnerable PHP files. Table 3 shows the classification results of the combined dataset. We can see that Random Forests achieved the best results in terms of AUC. In this combined dataset, we increased the number of instances, which gave the model a lot to learn about the characteristics of these PHP files.

Table 2. Classification Results of the Projects

Project	Algorithm	Precision	Recall	F-Measure	AUC
Drupal	Naive Bayes	0.739	0.752	0.733	0.762
	Decision Trees	0.754	0.748	0.75	0.723
	Random Forests	0.747	0.757	0.749	0.786
	Ensemble (Voting)	0.774	0.782	0.768	0.798
PHPMyAdmin	Naive Bayes	0.878	0.854	0.865	0.702
	Decision Trees	0.886	0.91	0.894	0.531
	Random Forests	0.905	0.922	0.899	0.675
	Ensemble (Voting)	0.889	0.913	0.896	0.714
Moodle	Naive Bayes	0.986	0.933	0.958	0.787
	Decision Trees	0.984	0.992	0.988	0.45
	Random Forests	0.984	0.991	0.987	0.614
	Ensemble (Voting)	0.984	0.992	0.988	0.742

Table 3. Classification Results of the Combined Dataset

Project	Algorithm	Precision	Recall	F-Measure	AUC
Dataset Combined	Naive Bayes	0.946	0.916	0.93	0.744
	Decision Trees	0.953	0.966	0.957	0.694
	Random Forests	0.965	0.971	0.964	0.853
	Ensemble (Voting)	0.959	0.969	0.961	0.835

It is important to understand, which features are the most influential features in determining whether particular PHP component is vulnerable or not. This will examine how well each metric can individually differentiate files as vulnerable or neutral. The most influential features can be computed using gain ratio [11]. Gain ratio provides a normalized measure of the contribution of each feature to classification. A secondary advantage gain ratio is to provide a comparison between each metric. Table 4 shows the most influential metrics and their abilities to predict vulnerable files. The higher the gain ratio is, the more important the feature is in identifying vulnerable files. The two most influential metrics in predicting vulnerabilities are the number of functions and lines of code. This indicates that large non-modular files [3] are more subject to vulnerabilities. The remaining influential metrics are related to coupling and complexity which complies with existing research and shows that software complexity is the enemy of software security [16].

Table 4. The Influential Metrics using the Gain Ratio Measure

Measure	**Gain Ratio**
Number of Functions	0.0288
Lines of Code (non-HTML)	0.0271
Lines of Code	0.0265
Halstead’s Volume	0.0199
Maximum Nesting Complexity	0.0173
Fan-In	0.0171
Cyclomatic Complexity	0.0171
Total External Calls	0.0161
Fan-Out	0.0148
Internal functions called	0.0126
External calls to functions	0.0113

5. Application

On the Internet, content filtering (also known as information filtering) is the use of a program to screen and exclude from access or availability web pages or e-mail that is deemed objectionable. An application firewall is a type of firewall that controls input, output, and/or access from, to, or by an application or service. It operates by monitoring and potentially blocking the input, output, or system service calls that do not meet the configured policy of the firewall.

Web content filtering or web filtering is usually used to prevent access to undesirable (vulnerable) web pages [9]. Web filtering uses screening of web requests and analysis of the contents of the received Web pages to block undesired Web pages. Current implementations of Web filtering use techniques such as blacklisting or whitelisting, keyword searching and rating systems. Blacklists and whitelists are costly and infeasible to maintain. Keyword searching is subject t to mistakes in spelling and rating systems are not reliable sources of information [8]. What we are proposing is a content-based filtering that is based on the content of PHP files. Screening a PHP request will determine if it passes the firewall or not. Since our web filtering approach is based on the content of Web pages that may extend over a number of IP packets, filtering must be performed after the page is reconstructed in a single file.

One application would be developing a rule-based firewall according the classification rules to filter vulnerable requests. This firewall would be able to distinguish vulnerable requests after evaluating some of the features of the PHP file. A firewall is a system for enforcing access control policy between two networks and is one of the most important measures to protect against network attacks. Adding a content filtering functionality to the firewall based on the rules we got from applying predictive models of PHP files.

6. Threats to Validity

In this section, we discuss the threats to validity of our proposed approach. The dataset was collected by Walden et al. [24]. Since the dataset is publicly available, we believe that our results are credible and can be reproduced. The impact of data preprocessing on prediction performance is also an interesting problem that needs further investigation. Other type of threats considers issues that affect the validity of statistical inferences. We mitigate this threat by using standard techniques for our statistics and modeling, and we used a well-recognized tool for these purposes (Weka). Since we only explored PHP web applications, our results might be specific to them. However, the selected applications are open source and from different domains. Future studies with a broader set of web applications, including both commercial and open source applications, would be needed to generalize the results to the entire class of PHP web applications. In addition, in order to generalize the results to other web applications written in other languages or to other types of software, desktop or mobile applications should be explored.

7. Related Work

This section discusses related studies. The section is divided into two subsections: studies on fault prediction and studies on vulnerability prediction using metrics.

7.1. Fault Prediction

Several studies have used software metrics to predict fault-prone components. Basili et al. [4] evaluated the usefulness of Chidamber & Kemerer's OO design metrics for predicting fault-prone classes. They collected the data of eight medium-sized management information systems. They used logistic regression techniques and found that these metrics were able to predict 88% of faulty classes with 60% precision. Briand et al. [5] evaluated the usefulness of Chidamber & Kemerer's OO design metrics to predict faulty classes. They conducted their experiment eight medium-sized management information systems that were developed by students. They used logistic regression model and found that that these metrics were able to predict more than 90% of faulty classes with about 80% precision. Menzies et al. [19] used software metrics to predict defects. They performed their experiment on a NASA repository, MDP. They built prediction models using OneR, J48 and Naive Bayesian classifiers and their Naive Bayesian model detected 71% of defects with a 25% false positive rate. Alenezi et al. [2] used the dataset of 8 open source systems to predict faulty classes. They built prediction models using Naive Bayes, Bayesian Networks, J48, and Random Forests. Their Random Forest model detected 92% of defects.

7.2. Vulnerability prediction

A small number of studies have explored the usefulness of using software metrics to predict vulnerabilities. Gegick et al. [10] used a static analysis tool and software metrics to predict vulnerable components. They conducted their experiment on a large commercial telecommunications software and predicted vulnerable components with an 8% false positive rate and a 0% false negative rate. Shin and Williams [22] computed the correlation between software metrics and software vulnerabilities. Their experiment was conducted on the Mozilla JavaScript Engine (JSE). The correlation tests showed only a weak correlation between software metrics and software vulnerabilities. Shin et al. [21] evaluated the usefulness of software metrics and developer activity metrics in predicting vulnerable files. Their experiment was conducted on two open source projects Mozilla Firefox and the Linux Kernel. They built three prediction models, using each group of metrics separately, and built a model using combined metrics. Discriminant analysis and Bayesian network were used as modelling techniques. In the best cases they were able to predict about 70% of vulnerable files with precision lower than 5%. Chowdhury and Zulkernine [6] used software metrics to predict vulnerabilities. Their experiment was conducted on 52 releases of Mozilla Firefox. They built prediction models based on C4.5 Decision Tree, Random Forests, Logistic Regression and Naive Bayes. The metrics were able to predict almost 75% of the vulnerable files, with a false positive rate of below 30% and an overall prediction accuracy of about 74%. Moshtari et al. [20] used software metrics to predict vulnerabilities. Their experiment was conducted on Mozilla Firefox. Their results showed that about 92% of vulnerable files were detected, with a False Positive rate of 0.12%.

7.3. Equations

Including symbols and equations in the text, the variable name and style must be consistent with those in the equations. Equations should be indented at the left margin and numbered at the right margin, equation number is enclosed with open and close parenthesis () Time New Roman in style and 11pt font size. Define all symbols the first time they are used. All equation symbols must be defined in a clear and understandable way.

8. Conclusion

In this paper, we presented an approach to predict the vulnerabilities of open source web applications. We compared the performance of different classification techniques in predicting vulnerable PHP files. We investigated whether software metrics are discriminative and predictive of vulnerable code, and can guide actions for improvement of code and development team and can prioritize validation and verification efforts. The results indicate that the metrics are discriminative and predictive of vulnerabilities. In addition, we proposed an application of these classification rules in developing a content-based firewall filter. Our future direction is to conduct a similar study with a broader set of web applications, including both commercial and open source applications.

References

[1] M. Alenezi and S. Banitaan, "Bug reports prioritization: Which features and classifier to use? In 12th IEEE International Conference on Machine Learning and Applications (ICMLA)", vol. 2, pp. 112–116. IEEE, (**2013**).

[2] M. Alenezi, S. Banitaan, and Q. Obeidat, "Fault-proneness of open source systems: An empirical analysis", In International Arab Conference on Information Technology (ACIT 2014), (**2014**), pp. 164–169.

[3] M. Alenezi and M. Zarour, "Modularity measurement and evolution in object-oriented open-source projects", In International Conference on Engineering & MIS 2015 (ICEMIS15), ACM, (**2015**).

[4] V. R. Basili, L. C. Briand and W. L. Melo, "A validation of object-oriented design metrics as quality indicators", IEEE Transactions on Software Engineering, vol. 22, no. 10, (**1996**), pp. 751–761.

[5] L. C. Briand, J. Wust, J. W. Daly and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems", Journal of systems and software, vol. 51, no. 3, (**2000**), pp. 245–273.

[6] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities", Journal of Systems Architecture, vol. 57, no. 3, (**2011**), pp. 294–313.

[7] G. Czibula, Z. Marian and I. G. Czibula, "Software defect prediction using relational association rule mining", Information Sciences, vol. 264, (**2014**), pp. 260–278.
[8] R. Du, R. Safavi-Naini, and W. Susilo, "Web filtering using text classification", In The 11th IEEE International Conference on Networks, IEEE, (**2003**), pp. 325–330.
[9] J. Duan and J. Zeng, "Web objectionable text content detection using topic modeling technique", Expert Systems with Applications, vol. 40, no. 15, (**2013**), pp. 6094–6104.
[10] M. Gegick, L. Williams, J. Osborne and M. Vouk, "Prioritizing software security fortification throughcode-level metrics", In Proceedings of the 4th ACM workshop on Quality of protection, pages 31–38. ACM, (**2008)**.
[11] M. A. Hall and L. A Smith, "Practical feature subset selection for machine learning", In 21st Australasian Computer Science Conference. Springer, (**1998**).
[12] R. Hornung, C. Bernau, C. Truntzer, T. Stadler, and A.-L. Boulesteix, "Full versus incomplete cross-validation: measuring the impact of imperfect separation between training and test sets in prediction error estimation", Technical Report 159, Department of Statistics, University of Munich, (**2014**).
[13] K.-S. Joo and J.-W. Woo, "Development of object-oriented analysis and design methodology for secure web applications", International Journal of Security and Its Applications, vol. 8, no. 1, (**2014**), pp. 71–80.
[14] L. I. Kuncheva, "Combining pattern classifiers: methods and algorithms", John Wiley & Sons, (**2004**).
[15] R. Malhotra, "Empirical Research in Software Engineering: Concepts, Analysis, and Applications", CRC Press, (**2015)**.
[16] G. McGraw, "Software security: building security in, volume 1", Addison-Wesley Professional, (**2006**).
[17] I. Medeiros, N. F. Neves and M. Correia, "Automatic detection and correction of web application vulnerabilities using data mining to predict false positives", In Proceedings of the 23rd international conference on World wide web, ACM, (**2014**), pp. 63–74.
[18] A. Meneely and L. Williams, "Strengthening the empirical analysis of the relationship between linus' law and software security", In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, (**2010**), p. 9.
[19] T. Menzies, J. Greenwald and A. Frank, "Data mining static code attributes to learn defect predictors", IEEE Transactions on Software Engineering, vol. 33, no. 1, (**2007**), pp. 2–13.
[20] S. Moshtari, A. Sami and M. Azimi, "Using complexity metrics to improve software security", Computer Fraud & Security, vol. 5, (**2013**), pp. 8–17.
[21] Y. Shin, A. Meneely, L. Williams and J. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities", IEEE Transactions on Software Engineering, vol. 37, no. 6, (**2011**), pp. 772–787.
[22] Y. Shin and L. Williams, "Is complexity really the enemy of software security?", In Proceedings of the 4th ACM workshop on Quality of protection, ACM, (**2008**), pp. 47–50.
[23] J. Walden, M. Doyle, R. Lenhof and J. Murray, "Idea: java vs. php: security implications of language choice for web applications", In Engineering Secure Software and Systems, Springer, (**2010**), pp. 61–69.
[24] J. Walden, J. Stuckman and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining", In IEEE 25th International Symposium on Software Reliability Engineering (ISSRE), IEEE, (**2014**), pp. 23–33.