Journal of Theoretical and Applied Information Technology
31st July 2018. Vol.96. No 14
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DISCOVERING THE RELATIONSHIP BETWEEN SOFTWARE COMPLEXITY AND SOFTWARE VULNERABILITIES

[1]YASIR JAVED, [2]MAMDOUH ALENEZI, [3]MOHAMMED AKOUR, [4]AHMAD ALZYOD

[1,2]College of Computer & Information Sciences Prince Sultan University Riyadh 11586, Saudi Arabia

[3,4]The Faculty of Information Technology and Computer Sciences, Yarmouk University, Irbid, Jordan

ABSTRACT

Software vulnerabilities might be exploited badly which might eventually lead to a loss of confidentiality, integrity, and availability which translated into a loss of time and money. Although several studies indicated that complexity in software is the main cause of vulnerabilities, still the argument is poorly designed and maintained. Moreover, some studies have already related complexity to vulnerabilities and found that this cannot be generalized. In this work, we explored that what are the factors that contribute more to make a software vulnerable. Several feature selection techniques were applied to find the contribution of each feature. Five classifiers are used in this study to predict the vulnerable classes. The dataset is collected from twelve Java applications, where these applications are analyzed and based on complexity, code coverage, and security. The studied applications are varying in its characteristics regarding a number of code lines, used classes; application size, etc. The result indicates that complexity in all its components (size, depth of inheritance, etc.) can be utilized in predicting vulnerabilities.

Keywords: *Software Vulnerabilities, Software Complexity, Fault Prediction, Relation, Code Complexity*

1. INTRODUCTION

Software vulnerabilities are one of the increasing problems with highest societal impact. Large corporations and individuals face these problems since they depend heavily on software systems. These exploited software vulnerabilities lead to a loss of confidentiality, integrity, and availability which translated into a loss of time and money. Several studies indicated that the main cause of vulnerabilities is software complexity [1]. Software practitioners hypothesize that ill-designed and maintained systems tend to be vulnerable. However, some studies have already studied the connection between complexity and vulnerabilities and found that this cannot be generalized [2],[3]. Morrison et al. [3] could not find any noteworthy relationship among complexity and vulnerabilities in their Microsoft study. They recommended using some security-specific metrics in vulnerability prediction models.

Machine learning and data mining methods have been utilized to construct vulnerability prediction models [4], [5]. These models are constructed with the assumption that the more complex the system is; it has a bigger chance to be vulnerable. It will be a matter of which features to be used as predictors which are usually determined by experience and empirical knowledge.

In this work, authors try to develop a prediction model of the relationship between software vulnerabilities and quality characteristics of software products. The ultimate goal is to use this developed model to predict these relationships in newly developed software where these data are not yet available. These models will help in managing software systems security issues, which will eventually reduce and mitigate the risks of implementing new systems.

Vulnerability exploration can be done in different phases throughout the software development lifecycle. In the literature, several models were suggested to approximate the number of vulnerabilities with a different variation of accuracy [6].

In pursuit of studying vulnerabilities, the data usually are drawn from public vulnerability databases: National Vulnerability Database (NVD) [7], or the Open Source Vulnerability Database [8]. This assumes that in order to predict vulnerabilities using any model, they have to be discovered first. This assumption excludes predicting vulnerabilities in newly released systems. Another reason is the

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

fact that the precision of these models depends on the number of known vulnerabilities for a software application [9].

2. RELATED WORK

There is no clear census on what metrics can be used in predicting software vulnerabilities. Most studies tried to establish correlation or predictability, however, the overall findings are not sufficient. In this section, an exploration of the related work will be presented. Shin and Williams [10] provided indications that defect prediction models can be utilized to predict vulnerabilities. Nevertheless, defect prediction models will not work since they include other issues other than vulnerabilities. Alenezi and Yasir [11] found the most common vulnerabilities in open source web-based project system. Their study depicts that most of the vulnerabilities may compromise the whole system especially in terms of security.

Chowdhury and Zulkernine [12] found that software metrics such as complexity, coupling, and cohesion to be strongly correlated with vulnerabilities. Shin and Williams [13] studied if complexity metrics can be used in predicting vulnerabilities. Their results indicated a correlation between complexity metrics and vulnerabilities in the Mozilla JavaScript Engine. Shin et al. [14] studied the relation of complexity, code churn and developer activity metrics with vulnerabilities. Javed and Alenezi [15] also found out the relation of vulnerabilities and their usual solving time and it seemed evident from their research that they prolong over time with even new versions of software's, thus requiring an in time solution rather than leaving it for later fix. Chowdhury and Zulkernine [16] explored complexity, coupling and cohesion metrics as predictors of vulnerabilities. They empirically demonstrated the study using fifty-two releases of Mozilla Firefox. They demonstrated the feasibility of building productive vulnerability prediction model.

Neuhaus et al. [17] studied the possibility of having correlations between C language include statements and vulnerabilities. They applied machine learning methods to predict vulnerabilities in Firefox and Thunderbird. Zimmermann et al. [18] were able to find only a weak correlation between various metrics and vulnerabilities. Their used metrics included code churn, code complexity, dependencies, and organizational measures. They constructed two predictors in Windows Vista, one was on code churn measures, code complexity metrics, dependency measures, code coverage measures and organizational measures and one was on dependencies between binaries.

Nguyen and Tran [19] employed dependency graphs as predictors of vulnerable components. They used two versions of Firefox JavaScript Engine to validate their predictors. Smith and Williams [20] employed SQL hotspots as predictors of vulnerable components. They found that more SQL hotspots in a component usually tend to indicate having vulnerabilities in that component. They used WordPress and WikkaWiki to validate their predictors.

A common criticism of static analysis tools is that they can produce many false positives [19]. However, recent studies have shown that vulnerability warnings from static analysis tools are not so unreliable after all. A collective opinion has emerged about static analysis tools which are the fact that they produce a lot of false positives [21]. However, recent research reports indicated that warnings generated by these tools are not unreliable. Walden and Doyle [22] studied the warnings generated by the Fortify SCA tool are strongly correlated to NVD vulnerabilities. Edwards and Chen [23] also studied the warnings generated by the Fortify SCA tool are strongly correlated to NVD vulnerabilities. Gegick et al. [24] & [25] showed a statistically significant correlation between static analysis tools warnings and vulnerabilities. Zheng et al. [26] indicated that static analysis tools are effective in finding security vulnerabilities after three large-scale industrial systems study.

Decan et al.[27] used npm packages report to find the vulnerabilities where he rated the vulnerabilities according to high, medium and low. It is found in research that how the vulnerabilities are detected and fixed. It is found that around 40% of vulnerabilities move on to next release thus are not fixed. It hasn't established any relationship with classes but has pointed that there are no automated tools available for detection of vulnerabilities. Kula et al.[28] used the most common set of Java libraries for detection of vulnerabilities but they focused on the popularity of model and bug fixes. It

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

also looked into more appropriate navigation and updating system and found CRAN to be the most appropriate one.

Choudhary et al. [29] evaluated the similar study on Eclipse project taken from GitHub. Their research looked into software vulnerabilities prediction where each feature was evaluated using data mining algorithms. They selected the following attributes

- Commits,
- Refactorings,
- Bugfixes,
- Authors,
- Loc-Added,
- Max-Loc-Added,
- Avg-Loc-Added ,
- Loc-Deleted,
- Max-Loc-Added,
- Avg-Loc-Added,
- Codechurn,
- Max-Codechurn,
- Avg-Codechurn,
- Max-Changeset

as existing metrics while

- Avg Commits,
- Time,
- Line of Code

were considered as new change metrics. They used four classes to divide their parameters their study reveals that Random forest performs better in their case while they only analyzed based on precision, recall, and fault.

Moser et al. [30] also conducted a similar kind of study where he did static analysis on Eclipse change logs while the metric taken were

- weighted, Average age,
- Code Churn (Max, Avg, Normal).
- Loc (Max, Avg Deleted)
- ChangeSet (Max, Avg)

and their observation was that decision tree performs better than other algorithms. They also analyzed the model according to precision and recall. They found their model to be performing better around 80%.

Dan et al. [31] used different techniques and approaches to find the location of vulnerabilities. They used Java-based applications to build an accurate predictor and used the same project as our research. Their model worked around 85% for the prediction of their predictor for vulnerabilities.

We based this research on these above parameters but we will first use the feature selection to most relevant attributes that can help in fixing the big chunk of issues while others can be controlled or have no contribution even.

3. EXPERIMENTS AND RESULT

3.1 Data Collection

In this section, we gathered data from twelve Java applications. We selected various application based on their usage of popularity and their wide usage in research as shown in the literature. Some of the project considered are small while others are large like Calculator have only 16 and Cinema have only 24 files while other like Jfree chart having 1014 and Jgap have 694 files. The diversity is considered in order to see the impact of software vulnerabilities as well building a generalized solution that can work in all cases. Secondly, we choose the project that is already on GitHub as it is the biggest and most used repository and moreover most of the research is being conducted on the GitHub is shown in the literature. Thirdly multiple Categories of feature selection has been applied to collected results. Most of the literature referred uses the following algorithms so we selected the following three algorithms

- ReliefFAttributeEval
- PCA
- CFSSubsetEVal

Fourthly, the following algorithms are applied to evaluate the effectiveness of each algorithm. Our study includes all of the algorithms that are used as a part of most of the referred literature to correctly validated the results. The following are the set the algorithms selected

- Decision table
- Linear Regression
- SMOReg
- IBK and
- Random Forest

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1 shows the list of Java applications that have been selected. It also shows the Github ID extracted from the links and number of Files that are being considered in each project. The number of Lines means Lines of code analyzed. These applications have been analyzed them based on complexity, code coverage, and security. The constructed dataset contains credible applications that vary in its characteristics regarding a number of code lines, used classes; application size etc. the main source of applications is *github.com*

Table 1: showing the list of Java Applications considered

Name of Java Application	Version	Github ID	Files
ApaCLI	1.5	apache/commons-cli	41
Cinema	1.0	Pozo/telnet-ascii-cinema	26
commons-codec	1.10	apache/commons-codec	115
commons-lang	3-3.4	apache/commons-lang	280
jfreechart	1.0.19	jfree/jfreechart	1014
jgap	3.6.3	jgap.cvs.sourceforge	694
joda-time	2.9.7	JodaOrg/joda-time	329
jtopas	1.0	jdc0589/jTranslate	64
marc4j	2.8.3	marc4j/marc4j	122
PureMVC	1.1	PureMVC/puremvc-java-standard-framework	51
calculator	1.0	kranonit/calculator-unit-test-example-java	16
JSON	20171018	stleary/JSON-Java-unit-test	55

Authors used SourceMonitor[1]Version 3.5 to study code complexity, by checking software source code to find out module's complexity and the amount of used code in the source file. We choose this tool due to various reasons, such as: Capability to work with Windows GUI or scripts using XML command files in addition to probability of checking multiple coding language such as C, C++, C#, VB.Net, Java, and more, ability to collect metrics in a quick fashion, offers Modified Complexity metric option, you might represent and print metrics in different formats, that can be exported and used in other tools.

For coverage check, authors chose to use EclEmma[2] Java Code Coverage 2.3.3. which is an Eclipse open source Java code coverage tool. The most important features of EcIEmma are: fast develop/test cycle, rich coverage analysis, and non-invasive. The third used tool is VisualCodeGrepper[3] Version 2.1.0, this tool is an automated code security review to check the security of applications. VCG supports many languages such as C++, C#, VB, PHP, Java and PL/SQL. VCG can be used as a white-box tool that is capable of analyzing software to find security issues. Some of the power points of this tool are: it has a config file for each language to expand the possibility of searching customized functions "written by you", find phrases embedded in comments that can indicate broken code.

3.2 Experimental Results

The experimental study aims to find, what are the factors that contribute to making a software vulnerable. Several feature selection techniques were applied to know how much each factor is contributing. Following is a brief description of these techniques.

[1]http://www.campwoodsw.com/sourcemonitor.html
[2]http://www.eclemma.org/
[3]https://sourceforge.net/projects/visualcodegrepp/

ReliefFAttributeEval: It is a way to evaluate the attribute using ReliefF. It works on binary classification and is not dependent on heuristics. It can remove noise as well as work on repeated sampling of attribute and thus not affected by redundant values. [32]

PCA: Principal component analysis is a technique that is used to find out the variation in the dataset and then extracting strong patterns between the attributes. These attributes are called as principal components as these values are uncorrelated linearly. This research uses PCA to find out the most effective attributes of the dataset. [33]

CFSSubsetEVal: is a correlation based feature subset finder that evaluates all the attributes and finds the redundancy among these attributes and focusing on the individual predictive ability to find out highly correlated attributes to be used. [34]

This research uses ReliefF, Principal Component analysis, and CFSSubset and then figured out what are the most common attributes that have been selected by these three attribute selection. Table 1 shows the attribute shortlisted by these three algorithms whereas we have selected *Potentially dangerous code* as classification or resultant attribute.

Table 2: Showing The Important Features Selected By Relieff, PCA And Cfssubset

ReliefF	PCA	CFSSubset
Classes	Files	Classes
Files	Branches	MethodClass
Max Depth	Comments Percentage	AvgStaments/ Methods
	Classes	Max Depth
	MethodsClass	Avg Depth
	Avg Statement/ Methods	

It is evident from Table 2 that *Classes* has been identified as a potential attribute while *Files, Max Depth, Method Class and Avg Statement Methods* has been identified more than once. Thus creating a selection of ranking based on our selection of attributes that are selected more than once by the attribute selection attribute as it is assumed that these attributes are much effective. So the attribute selected as follows

Classes, Files, Method Class, Max Depth, Avg Statement/ Methods, then the rest as Branches, Comments Percentage, Avg Depth thus selecting five attributes as main and three as supportive attributes and will be used in along with the classification attribute Potentially Dangerous code.

After evaluation of features contributing, another experiment is to classify these classes as vulnerable or not. The prediction model is trained using five of the classification techniques. We use the following classification algorithms in our study.

Decision Table: Decision table is a precise way of selection of most impacting attributes, complex rules can be modeled and are easier to program in a way like programming if then else or switch cases. [30] This research selects the decision table in order to get a rule that provides a relative classification that can provide high probable classification.

Linear Regression: Linear Regression is a way to create an equation that can fit the provided dataset; the output is linear function thus helping in building an easy classification model where a simple linear equation can handle all kinds of data [35]. This research selects linear regression for the same reason where a single equation can be provided in order to find the potentially dangerous code where it can rate with high probability.

SMOReg: SMO uses heuristics for training model and is based on regression. It also supports vector machines and is more efficient and fast [36]. This research uses in order to build a fast and high probabilistic training model.

IBK: IBK is a classifier based on K nearest neighbor. It has a great advantage for case-based reasoning [37] so this research uses this so that the present data set can be used for classification of new programs.

Table 3: Statistics Of The Results For Each Selected Algorithm

	Decision table	Linear Regression	SMOReg	IBK	Random Forest
Correlation coefficient	0.8535	0.8319	0.8514	0.9276	0.9549
Mean absolute error	578.625	853.4078	877.7278	492.4167	496.0497
Root mean squared error	1033.6739	1272.6324	1020.3108	653.1926	747.5166
Relative absolute error	35.1013 %	51.7706 %	53.2459 %	29.8717 %	30.0921 %
Root relative squared error	55.2223 %	67.9882 %	54.5084 %	34.8957 %	39.9348 %

Table 4: Statistics Of The Results For Each Selected Algorithm For Reduced Attributes

	Decision table	Linear Regression	SMOReg	IBK	Random Forest
Correlation coefficient	0.8535	0.9134	0.9272	0.9372	0.9603
Mean absolute error	578.625	582.7637	436.0651	427.8333	397.8513
Root mean squared error	1033.6739	820.4736	663.9729	606.6038	636.6357
Relative absolute error	35.1013 %	35.3524 %	26.4532 %	25.9538 %	24.135 %
Root relative squared error	55.2223 %	43.8324 %	35.4716 %	32.4068 %	34.0112 %

RandomForest: RandomForest is another classification technique that develops multiple decision trees where each tree is extended to the maximum extent, while the correlation between two trees are found and each tree strength is found [38]. This research uses this algorithm as it has high accuracy and can run on large datasets or databases. Table 3 shows the results that are achieved using the above selected five algorithms

It is shown in table 3 that Random forest can be used as best classifier as well as IBK where the correlation coefficient is above 0.9 and relative absolute error is lower than or equal to 30%. It is also seen that all of the selected algorithms have more than 80% correlation coefficient that shows that selected attributes are highly correlative efficient.

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

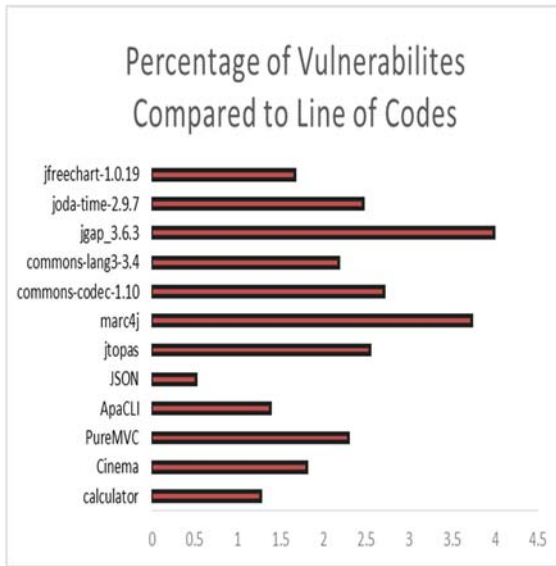

Figure 1: Showing The Number Of Vulnerabilities Compared To Line Of Code

Figure 1 shows the relation of lines of codes with potential vulnerabilities although the relation might not reveal much knowledge in form of percentage value but it gives an indication that it doesn't matter the size of project the vulnerabilities may exist in both cases especially in larger projects even if there is a community there are chances of vulnerabilities.

Compared to the analysis performed by [27] – [31] where the results were having a value of 85% especially by considering more number of variables study by Choudhary et al. [29] where the prediction and Moser et al. [30]. Our results with the reduced number of features perform slightly better showing that if we can detect the major violation reason a great dealt with software vulnerabilities can be avoided. The depth of code is also related to potential violations as shown in Figure 2.

Figure 2 shows that depth and violations are highly related along with the number of files. The small lines represented the relation of files and violations while purple line represents the depth. The higher the depth there are more likely chances of having the vulnerabilities.

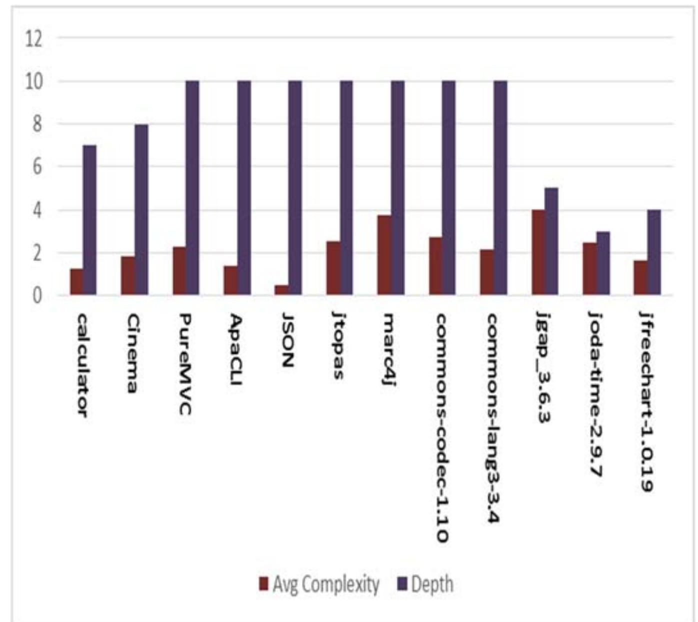

Figure 2: Showing The Number Of Vulnerabilities With Max Depth For Selected Projects.

The decision table algorithm returned classes as the most relevant in decision making.

While linear regression gave the following equation

$$Potenitally\ Dangerous\ Code = 4.9514 * Files + 136.51.02 * MethodsClas - 744.6754$$

That shows that it is dependent on files and max depth. Moreover, the selection of last three attributes may not be making much effect so let's remove and run the algorithm again with only Classes, Files, Method Class, Max Depth and Avg Statement/ Methods and the Table 4 is achieved.

Table 3 shows relevant attributes it shows that results are still high coefficient. The results are improved due to removal or less relevant attributes and the value have gone above 90% while the relative absolute error has been reduced especially for linear regression and SMOreg from above 50% to less than 36%. The result also shows that selection of relative attributes.

Attribute selected by decision tree is still *Classes* while the linear equation returned is given as below

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$Potenitally\ Dangerous\ Code = 5.1905 * Files + 83.02 * MethodsClas - 422.61$$

Here looking at above equation it is shown that *methodClass* is selected instead of max depth from the previous equation and it will result in the better predictor.

This research work focused on finding the most optimal predictor for finding vulnerabilities. Decision Table, Linear Regression, SMOReg, IBK, and RandomForest were used in finding out the most relevant factors contributing to making the software's vulnerable. It is also evident from the literature that software vulnerabilities are mostly happening due to the complexity and bad design. Thus it is required to write a clean, modular and highly coupled software's to decrease the chance of vulnerabilities.

4. CONCLUSION

Software vulnerabilities if exploited can result in software security breaches resulting in loss or compromise of data, confidentiality that results in higher cost so major losses. There is a number of studies that have been done in the area of finding software vulnerabilities or finding its complexity but lacks the generalized concept of finding the contributing factor for making the software vulnerable. We selected twelve projects based on the varying size and popularity. We selected five classifiers after the using the feature selection algorithm. After studying java applications with the aim of exploring the relationship between complexity and software vulnerabilities, we can conclude that there is a strong relationship. We explored the relationship by using both multiple classifiers and multiple feature selection algorithms. Our finding indicates that a number of files and the percentage of methods to classes are main contributors to making a software vulnerable. This strengthens the software engineering theory that mentions modularity as the main goal in software design. Modularity means small components with specific goals. Reducing size, complexity, and coupling would make the software more modular and more secure.

REFERENCES

[1] Lagerström, Robert, et al. "Exploring the Relationship between Architecture Coupling and Software Vulnerabilities: A Google Chrome Case." (2017).

[2] Y. Shin and L. Williams, “Is complexity really the enemy of software security?,” in Proceedings of the 4th ACM workshop on Quality of protection, pp. 47-50, 2008.

[3] P. Morrison, K. Herzig, B. Murphy, and L. Williams, “Challenges with applying vulnerability prediction models,” in Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, p. 4, 2015.

[4] Y. 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, vol. 37, no. 6, pp. 772–787, 2011.

[5] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, “Predicting vulnerable software components,” in ACM Conference on Computer and Communications Security (CCS), 2007

[6] O. Alhazmi and Y. Malaiya, “Prediction capabilities of vulnerability discovery models,” in Proc. RAMS ’06. Annu. Rel. Maintainability Symp., 2006, pp. 86–91.

[7] National Vulnerability Database 2011 [Online]. Available: http://nvd. nist.gov/

[8] The Open Source Vulnerability Database 2011 [Online]. Available: http://osvdb.org/

[9] O. Alhazmi and Y. Malaiya, “Application of vulnerability discovery models to major operating systems,” IEEE Trans. Rel., vol. 57, no. 1, pp. 14–22, Mar. 2008.

[10] Y. Shin and L. Williams,, “Is complexity really the enemy of software security?” in ACM Workshop on Quality of Protection (QoP), 2008.

[11] Alenezi, Mamdouh, and Yasir Javed. "Open source web application security: A static analysis approach." Engineering & MIS (ICEMIS), International Conference on. IEEE, 2016.

[12] Chowdhury and M. Zulkernine, “Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?,” in Proc. 2010 ACM Symp. Appl. Comput., New York, NY, USA, 2010, pp. 1963–1969 [Online]. Available:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

http://doi.acm.org/10.1145/1774088.1774504, ser. SAC '10, ACM

[13] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," in ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2008

[14] Y. 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, vol. 37(6), pp. 772–787, 2011.

[15] Javed, Yasir, and Mamdouh Alenezi. "Defectiveness Evolution in Open Source Software Systems." Procedia Computer Science 82 (2016): 107-114.

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

[17] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in, ACM Conference on Computer and Communications Security (CCS), pp. 529-540, 2007.

[18] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in International Conference on Software Testing, Verification and Validation (ICST), 2010.

[19] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," in International Workshop on Security Measurements and Metrics (MetriSec), 2010.

[20] B. Smith and L. Williams, "Using SQL hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities," in IEEE International Conference on Software Testing, Verification and Validation (ICST), 2011.

[21] Austin and L. Williams, "One technique is not enough: A comparison of vulnerability discovery techniques," in International Symposium on Empirical Software Engineering and Measurement (ESEM), 2011.

[22] J. Walden and M. Doyle, "SAVI: Static-analysis vulnerability indicator," IEEE Security & Privacy, vol. 10, no. 3, pp. 32–39, 2012.

[23] N. Edwards and L. Chen, "An historical examination of open source releases and their vulnerabilities," in ACM conference on Computer and Communications security (CCS), 2012.

[24] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing software security fortification throughcode-level metrics," in ACM Workshop on Quality of Protection (QoP), 2008.

[25] M. Gegick, P. Rotella, and L. Williams, "Predicting attack-prone components," in International Conference on Software Testing Verifi- cation and Validation (ICST), 2009.

[26] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the value of static analysis for fault detection in software," IEEE Transactions on Software Engineering, vol. 32, no. 4, pp. 240–253, 2006.

[27] Decan, A., Mens, T., & Constantinou, E. (2018). On the impact of security vulnerabilities in the npm package dependency network.

[28] Kula, R. G., De Roover, C., German, D. M., Ishio, T., & Inoue, K. (2018, March). A generalized model for visualizing library popularity, adoption, and diffusion within a software ecosystem. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)(pp. 288-299). IEEE.

[29] Choudhary, G. R., Kumar, S., Kumar, K., Mishra, A., & Catal, C. (2018). Empirical analysis of change metrics for software fault prediction. Computers & Electrical Engineering, 67, 15-24.

[30] Moser, R., Pedrycz, W., & Succi, G. (2008, May). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Proceedings of the 30th international conference on Software engineering (pp. 181-190). ACM.

[31] Dam, H. K., Tran, T., Pham, T., Ng, S. W., Grundy, J., & Ghose, A. (2017). Automatic feature learning for vulnerability prediction. arXiv preprint arXiv:1708.02368.

[32] Kononenko, I. (1994, April). Estimating attributes: analysis and extensions of RELIEF. In European conference on machine learning (pp. 171-182). Springer Berlin Heidelberg.

[33] Jolliffe, I. (2002). Principal component analysis. John Wiley & Sons, Ltd.

[34] Hall, M. A., & Smith, L. A. (1997). Feature subset selection: a correlation based filter approach.

[35] M. Hall and E. Frank. Combining naive Bayes and decision tables. In Proc 21st Florida Artificial Intelligence Research Society Conference, Miami, Florida. AAAI Press, 2008.
[36] Seber, G. A., & Lee, A. J. (2012). Linear regression analysis (Vol. 936). John Wiley & Sons.
[37] Li, Chaoqun, and Liangxiao Jiang. "Using locally weighted learning to improve SMOreg for regression." PRICAI 2006: Trends in Artificial Intelligence (2006): 375-384.
[38] I.H. Witten, E. Frank “Data Mining: Practical Machine Learning Tool and Technique with Java Implementation” Morgan Kaufmann, San Francisco (2000)