*Journal of Universal Computer Science, vol. 22, no. 4 (2016), 537-551*
*submitted: 1/10/15, accepted: 30/3/16, appeared: 1/4/16* © *J.UCS*

# An Empirical Investigation of Security Vulnerabilities within Web Applications

**Ibrahim Abunadi**
(Prince Sultan University, Riyadh, Saudi Arabia
iabunadi@psu.edu.sa)

**Mamdouh Alenezi**
(Prince Sultan University, Riyadh, Saudi Arabia
malenezi@psu.edu.sa)

**Abstract:** Building secure software is challenging, time-consuming, and expensive. Software vulnerability prediction models that identify vulnerable software components are usually used to focus security efforts, with the aim of helping to reduce the time and effort needed to secure software. Existing vulnerability prediction models use process or product metrics and machine learning techniques to identify vulnerable software components. Cross-project vulnerability prediction plays a significant role in appraising the most likely vulnerable software components, specifically for new or inactive projects. Little effort has been spent to deliver clear guidelines on how to choose the training data for project vulnerability prediction. In this work, we present an empirical study aiming at clarifying how useful cross-project prediction techniques are in predicting software vulnerabilities. Our study employs the classification provided by different machine learning techniques to improve the detection of vulnerable components. We have elaborately compared the prediction performance of five well-known classifiers. The study is conducted on a publicly available dataset of several PHP open-source web applications in the context of cross-project vulnerability prediction, which represents one of the main challenges in the vulnerability prediction field.

**Key Words:** cross-project vulnerability prediction, software security, software quality, data mining.
**Category:** D.2.0, D.2.8, K.6.5

## 1 Introduction

Software development and engineering is a very complex endeavor that contends with limited resources [Fenton and Bieman, 2014], potentially causing software to behave in an unexpected manner. One essential reason for the insecurity of web applications is the fact that most developers lack appropriate knowledge regarding secure coding [Medeiros et al., 2014]. The most worrisome class of these faults can be exploited by attackers. These faults are considered a security vulnerability [Bishop, 2005] that is recurrent, causing companies to struggle to allocate resources for their management [Nyanchama, 2005].

Software Security Vulnerability Prediction is a research field that utilizes effective methods for predicting the vulnerability in a given software component

[Abunadi and Alenezi, 2015]. These methods help security tester engineers allocate their limited resources to the most vulnerable systems. The process of building secure software systems is expensive, difficult, and time-consuming. Building and distributing vulnerability prediction models can cause quality assurance teams to focus their time and resources on the vulnerable parts of their code base. Researchers, however, usually build vulnerability prediction models that use metrics and vulnerability data. Known as supervised learning approaches in machine learning fields, these models implement different types of learning algorithms.

Finding and solving vulnerabilities in the early stages of software composition is an important step. Software vulnerability prediction is a quality assurance technique that includes inspection and testing within the software quality engineering discipline [Zhang et al., 2011]. However, such quality assurance is best done by engineers who are specially trained in software security [McGraw, 2006]. Methods and techniques that identify components that are more likely to contain vulnerabilities can provide significant aid to the security engineers who focus their attention on higher risk components.

The security of most systems and networks depends on the security of the software running on them. Most of the attacks on these systems exploit vulnerabilities found in these software applications [Walden et al., 2010]. Security failures in software are common and growing [Chowdhury and Zulkernine, 2011]. A vulnerability in software is considered a flaw that can be exploited to cause a security failure. It is very challenging to find vulnerabilities before they manifest themselves as security failures while the software is operating, because security concerns are usually not sufficiently known at early stages of the software development life cycle. Therefore, it is essential to know in advance the characteristics of software files that can indicate vulnerabilities. These indications would help software security testers and managers take proactive action against potential vulnerabilities. Security testing is an important requirement of software security [Damiani et al., 2008], even if it is very resource intensive, security testing activities need to be guided.

Software is a competitive business that evolves rapidly to respond adapting to markets, hardware, and software platforms [Alenezi and Khellah, 2015]. New projects are born, and aged ones are rewritten, creating a challenge for vulnerability prediction models that rely on historical vulnerability data to predict future vulnerability proneness. Because many new projects do not have enough historical data to train prediction models, we can use models estimated from training data available on other projects. PHP web applications are the focus of this paper since PHP has a poor security reputation within the open-source community and in comparison to Java, more than twice as many open-source web applications are written in PHP [Walden et al., 2010].

Within-project vulnerability prediction is built from a part of a project and evaluated on the remainder of the project. Cross-project vulnerability prediction is done when new projects lack sufficient vulnerability data to build a prediction model. In this case, we use data from other projects to build a prediction model. Several companies and projects might not yet have attained historical information about vulnerabilities in order to build prediction models. As a result, the ability of several metrics collected from software files is evaluated in this work. These files are used to predict vulnerabilities in a project by using other projects' datasets. Hence, this is a cross-project vulnerability prediction endeavor.

The rest of this paper is organized as follows: Section 2 provides a background about this study while Section 3 describes the proposed approach, and Section 4 is a case study with its experimental evaluation. Section 5 discusses the results obtained in this study; Section 6 discusses some threats to validity. Section 7 discusses related work. Last, Section 8 concludes the paper.

## 2 Background

Necessary background information about this study is provided. Subsection 2.1 discusses software vulnerabilities in general. Subsection 2.2 discusses some background details about software metrics. Subsection 2.3 explains the meaning of classification in data mining. Subsection 2.4 explains the difference between within and cross project prediction.

### 2.1 Software Vulnerabilities

Software vulnerability is a security fault or weakness found in software, which lead to security concerns. According to Common Vulnerabilities and Exposures (CVE), 69417 vulnerabilities were found in web applications within 1999-2015. Among them, 31.4 belong to Remote Code Execution, 21% to Denial of Service Attack (DoS), 13.5% to Cross-Site Scripting (XSS), 9% to SQL Injection, 3.1% to File Inclusion, and 1.7% to CSRF. The data used in this study focuses on Code Injection, Cross-site Request Forgery (CSRF), XSS, Path Disclosure, Authorization issues, and other types. These common vulnerabilities are responsible for more than 75% of the total number of vulnerabilities found. All of these types of vulnerabilities are caused by potential weakness in web applications. According to the Open Web Application Security Project (OWASPs) Top Ten Project [OWASP, 2015], SQL injection, cross site scripting (XSS), remote code execution (RCE), and file inclusion (FI) are the most common and serious web application vulnerabilities threatening the privacy and security of both clients and applications. We describe the most common vulnerabilities types as the followings:

- Code Injection: This type allows attackers to modify arbitrary server-side variables, modify arbitrary HTTP headers, or execute PHP, SQL, or native code on the server.
- CSRF: Cross-site request forgery vulnerabilities allowing for outside, malicious HTML to induce the user to perform unwanted actions
- XSS: Cross-site scripting which allows malicious Javascript to be executed in a users browser
- Path Disclosure: A vulnerability that allows for the installation path of the application to be maliciously obtained. Authorization issues: Confidentiality, integrity, or availability violations not related to another category. These include privilege bypass vulnerabilities; information disclosure vulnerabilities; vulnerabilities related to missing or inadequately implemented encryption.
- Other: Miscellaneous vulnerabilities related to phishing, man-in-the-middle attacks, or unspecified attack vectors.

### 2.2 Software Metrics

Software metric is a measure of a degree to which a software system possesses some property. There are four categories of software metrics. This classification is based on what they measure and what area of software development they focus on. At a very high level, software metrics can be classified as Process Metrics, Project Metrics, Product Metrics, and Personnel Metrics [Fenton and Bieman, 2014]. The data used in this study focuses on Product Metrics. They measure characteristics of the result of a software development process. Product metrics are typically calculated from source code. Product metrics refer to different features of the product such as design features, size, complexity, and performance.

### 2.3 Classification

Classification consists of predicting a certain outcome based on a given input. The objective of classification is to accurately predict the target class for each case in the data. The classification algorithm processes a training set containing a set of attributes and the respective outcome, usually called goal or prediction attribute [Scandariato et al., 2014].

A classification task begins with a data set in which the class assignments are known. The simplest type of classification problem is binary classification. In binary classification, the target attribute has only two possible values: for example, vulnerable or not. In building the prediction model, a classification

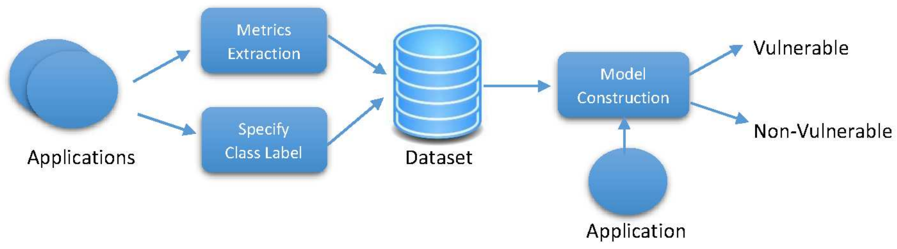


**Figure 1:** The Proposed Approach.

algorithm finds relationships between the predictors and the target. Different classification algorithms use different techniques for finding relationships. The model is built on these relationships. These models are tested by comparing the predicted values to known target values in a set of test data. The historical data for a classification project is typically divided into two data sets: training data and testing data [Scandariato et al., 2014, Alpaydin, 2014].

### 2.4 Within vs Cross Project Prediction

Within-project vulnerability prediction is built from a part of a project and evaluated on the remainder of the project [Shar et al., 2014]. Cross-project vulnerability prediction is done when new projects do not have enough vulnerability data to build a prediction model. Data from other projects can be used to build a prediction model. Several companies and projects might not yet have attained historical information about vulnerabilities in order to build prediction models. Thus, this study introduces the use of cross project vulnerability to overcome this practical and theoretical gap. To the best of our knowledge this methodology has not been used in software vulnerability prediction research before.

## 3 The Proposed Approach

We formulate vulnerability prediction as a classification problem by predicting if a PHP file is vulnerable. In this study, we present a framework for how cross-project vulnerability prediction can be used to automatically predict vulnerable files in new projects. The proposed approach is illustrated in Figure 1. Based on the source code metrics of the PHP files of two projects and their class labels (vulnerable or not), we train the prediction model founded on the data of two different PHP projects. After running the model on this training dataset, we test this model on a third project and evaluate the effectiveness of this model in predicting the vulnerabilities of another project (cross-project prediction). Cross-project vulnerability prediction models are trained on data from one or more projects for which predictors (e.g., product metrics) and actual vulnerabilities are available. Then, machine learning techniques (classifiers) are used to build prediction models to foresee the vulnerabilities in the software files of a new project.

## 4 Case Study

### 4.1 Dataset

In this study, we used a dataset collected and analyzed by Walden, Stuckman, and Scandariato [Walden et al., 2014]. The data contain several software metrics and vulnerability information about their PHP files. 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. Regarding the software metrics in these datasets, the following metrics were collected: lines of code, lines of code (non-HTML), number of functions, cyclomatic complexity, maximum nesting complexity, Halsteads volume, total external calls, fan-in, fan-out, internal functions or methods called, external functions or methods called, and external calls to functions or method. For more information about these metrics and how they were collected, please consult Walden et al. [Walden et al., 2014]. Table 1 shows descriptive statistics about the dataset.

**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 Experimental Design

We applied five well-known and most-used classifiers for building the vulnerability prediction models of the available dataset in terms of evaluation measures. We further explored which of these systems would be a good candidate for the training dataset.

### 4.3 Classifiers

One of the methodologies used to automatically predict vulnerabilities is performing data mining techniques on software metrics. The classification process consists of two phases: the learning phase and the classification (prediction) phase [Han et al., 2011]. The model learns through training data, and the classification is evaluated based on its ability to predict class labels for given data.

In this study, to build vulnerability prediction models, we used five popular data mining and statistical techniques (classifiers), namely, Nave Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM), J48, and Random Forest (RF). These classifiers classify software files as vulnerable or non-vulnerable. All these classification algorithms are implemented in Weka. The Weka default settings of these algorithms were used [Hall et al., 2009].

Naive Bayes (NB) is a probabilistic classifier that assumes that all features are independent. It finds the class with maximum probability given a set of feature values using the Bayes theorem [Lewis, 1998].

Logistic Regression (LR) is a probability model used to predict a response based on one or more features. It is used as a function of the predictors, using a logistic function to estimate the class label [Hilbe, 2009].

Support Vector Machine (SVM) is a classifier that finds the optimal hyperplane, which maximally separates samples in two different classes. SVM represents the examples as points in the space to divide them by a clear gap so the new examples can be mapped into the same predicated category [Hearst et al., 1998].

J48 is an implementation of the decision tree algorithm in Weka. This algorithm uses a divide-and-conquer approach to growing decision trees. It forms a tree structure and decides the dependent value of a new sample based on diverse attribute values of existing data [Quinlan, 2014].

Random Forests (RF) is an ensemble learning method that generates several decision trees at 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 [Chan and Paelinckx, 2008].

### 4.4 Evaluation Measures

We evaluate the classification algorithms based on Precision, Recall, and F-measure. 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 [Neuhaus et al., 2007]. F-Measure is the harmonic mean of Precision and Recall [Powers, 2011]. 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). For the two-class problem (e.g., vulnerable or non-vulnerable), these performance measures are explained using a confusion matrix, shown in Table 2. There, the confusion matrix shows the actual vs. the predicted results.

**Table 2:** Confusion Matrix

| Actual | Predicted As | |
|---|---|---|
| | Non-vulnerable | Vulnerable |
| Non-vulnerable | TN=True Negatives | FP = False Positives |
| Vulnerable | FN = False Negatives | TP = True Positives |

The prediction performance measures used in our experiments are described as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

### 4.5 Results

Table 3 shows some interesting results where each row shows a classifier performance, and each column depicts an evaluation metric. The J48 and RF classifiers outperform the other classifiers as a whole, indicated by Precision, Recall, and F-measure. Using the F-measure, RF achieves better results compared to the other classifiers. However, J48 and RF classifiers achieve very similar results with very little variations. For example, the F-measure value of the Drupal dataset in case of J48 is 0.751; RF is 0.752, while LR is only 0.727. The F-measure value of the Moodle dataset in case of J48 is 0.990, and RF is 0.991. NB is only 0.959. The F-measure value of the PHPMyAdmin dataset in case of J48 is 0.898, and RF is 0.913, while NB is only 0.866. We found that, in most cases, only J48 and RF classifiers are the best-performing techniques based on Precision, Recall, and F-measure, while the NB and LR classifiers are the worst.

To investigate whether the difference between J48 and RF in building predictive models is significant, the Kruskal-Wallis test is executed. The Kruskal-Wallis

**Table 3:** Classification results.

| System | Classifier | Precision | Recall | F-Measure |
|---|---|---|---|---|
| Drupal | NB | 0.739 | 0.752 | 0.745 |
| | LR | 0.721 | 0.733 | 0.727 |
| | SVM | 0.746 | 0.748 | 0.747 |
| | J48 | **0.754** | 0.748 | 0.751 |
| | RF | 0.747 | **0.757** | **0.752** |
| Moodle | NB | 0.986 | 0.933 | 0.959 |
| | LR | 0.986 | 0.991 | 0.988 |
| | SVM | 0.984 | 0.992 | 0.988 |
| | J48 | **0.987** | 0.994 | 0.990 |
| | RF | **0.987** | **0.995** | **0.991** |
| PHPMyAdmin | NB | 0.878 | 0.854 | 0.866 |
| | LR | 0.888 | 0.916 | 0.902 |
| | SVM | 0.839 | 0.916 | 0.876 |
| | J48 | 0.886 | 0.91 | 0.898 |
| | RF | **0.905** | **0.922** | **0.913** |

test [Sprent and Smeeton, 2007] is a nonparametric alternative to the one-way analysis of variance (ANOVA). The Kruskal-Wallis test was executed individually for F-measure values (See Table 4). The p-value is larger than 0.05, indicating that the difference between the F-measure values is statistically insignificant.

**Table 4:** The results of two Kruskal-Wallis tests for J48 and RF

| | **F-measure** |
|---|---|
| chi-squared | 2 |
| degree of freedom | 2 |
| p-value | 0.3679 |

We trained the model on two of the datasets, Drupal and PHPMyAdmin, since their vulnerability distribution was more than the Moodle datasets. The results of the cross-project prediction are shown in Table 5. The results were very reliable, considering that the prediction models have predicated the vulnerability of the files successfully even though it is a new project for the prediction model. The difference in performance between within-project prediction and cross-project prediction is minimal. For example, in case of the J48 classifier, the difference between the F-measure in the Moodle dataset within-project prediction and cross-project prediction is 0.014. Figure 2 and Figure 3 show the

**Table 5:** Cross Project Prediction Results

| Classifier | Precision | Recall | F-Measure |
|---|---|---|---|
| J48 | 0.985 | 0.967 | 0.976 |
| RF | 0.984 | 0.959 | 0.971 |

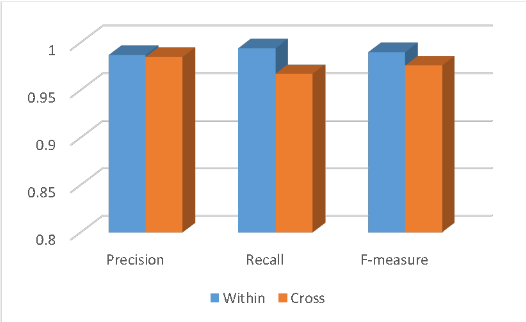


**Figure 2:** Within-project vs. Cross-project Model Performance (J48)

difference between within-project prediction and cross-project prediction using two classifiers (J48 and RF).

## 5 Discussion

In this section, we discuss the results achieved in this study. One question that arises is: are different classifiers equivalent to each other when applied to cross-project defect prediction? We first analyzed the performance of the different

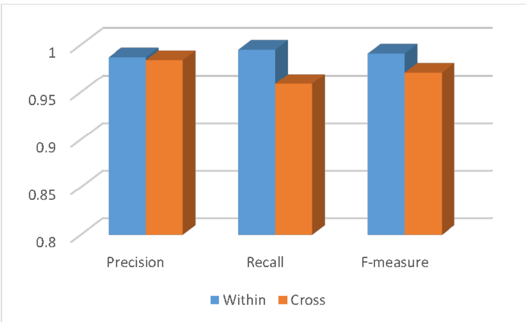


**Figure 3:** Within-project vs. Cross-project Model Performance (RF)

machine-learning techniques to test whether one technique provides better prediction accuracy than the others. We first computed precision, recall, and F-measure metrics obtained by the different techniques. The five experimented machine-learning techniques are not equivalent. RF in case of the F-measure in all datasets obtained the best prediction performance. After a thorough analysis and statistical tests, we found that the best classifiers in within-project prediction are J48 and RF, which is completely consistent with the conclusions drawn in the literature [Fernández-Delgado et al., 2014]. We tested the ability of these two classifiers in cross-project prediction and found that J48 achieved better than RF, but at a minimal margin.

Our experimental results show that, in the training data, it is better to have data that have a reasonable vulnerability distribution, which will enable the classifier to distinguish between vulnerable and non-vulnerable software components.

As shown in figures 2 and 3, this paper focuses on how to use different project vulnerability data in order to predict the vulnerabilities of a new project. The marginal difference between within-project prediction and cross-project prediction shows us the feasibility of cross-project vulnerability prediction.

## 6 Threats to Validity

In this section, we discuss the threats to construct, conclusion, and external validity that affect the validity of our proposed approach.

Construct validity. This type of threat is primarily related to the dataset explored in this study. The dataset were collected by [Walden et al., 2014]. 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.

Conclusion validity. This type of threat 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).

External validity. Since we only explored PHP web application, our results might be specific to them. Even though, 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. Moreover, results could be generalized to other web applications written in other languages or to other types of software, such as desktop or mobile applications should be explored.

## 7 Related Work

This paper considers cross-project vulnerability prediction in web applications using machine learning. Several researchers have explored vulnerability prediction, but their studies suffered from several limitations, namely: they only reported vulnerabilities to label vulnerable components, they used limited classification techniques, and all of them investigated within-project vulnerability prediction.

An earlier work regarding vulnerability prediction for web applications was done by Neuhaus et al. [Neuhaus et al., 2007]. Their work predicted vulnerability in the Mozilla code base using only SVMs as a classifier. Recorded precision was only 70%, while recall was only 45%. Yamaguchi, Lottmann, and Rieck [Yamaguchi et al., 2012] determined structural patterns in abstract syntax trees using natural processing techniques. Their results were validated using popular open-source projects such as Pidgin and FFmpeg. The approach used only focused on program functions without an emphasis on a specific application.

Shin and Williams [Shin and Williams, 2008] studied the correlation between complexity metrics and vulnerabilities. Their experimental analysis was based on the Mozilla JavaScript Engine. Their results showed a weak correlation between complexity metrics and security problems. They advised that their results are weak because they designated as vulnerable any function that was changed to fix vulnerability.

Nguyen and Tran [Nguyen and Tran, 2010] studied the dependency graphs as predictors of software vulnerabilities. Their study was also based on the Mozilla JavaScript Engine. The average precision of their study was 68%, which is lower than the one found in this study.

Shin, Meneely, Williams, and Osborne [Shin et al., 2011] studied how useful complexity, code churn, and developer activity metrics were in finding vulnerable files. Their experimental analysis on Firefox and the Linux Kernel revealed that, in the best cases, they were able to predict about 70% of vulnerable files with a precision lower than 5%.

Chowdhury and Zulkernine [Chowdhury and Zulkernine, 2011] used several source code metrics, such as complexity, coupling, and cohesion, to predict vulnerabilities. They conducted a study on 52 releases of Mozilla Firefox and built vulnerability predictive models using C4.5 Decision Tree, Random Forests, Logistic Regression, and Nave Bayes. Their models 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%. They concluded that complexity, coupling, and cohesion metrics are useful in vulnerability prediction.

Walden et al. [Walden et al., 2014] provided a public dataset that contains 223 vulnerabilities found in three PHP web applications and compared text mining vulnerability prediction models with source code metrics prediction models.

Shar and Tan [Shar and Tan, 2012] characterized input sanitization and validation code patterns using static code vulnerability predictors. Although they used multiple classifiers (Nave Bayes, C4.5, and MLP) and their prediction achieved notable accuracies, they only targeted within-project vulnerability prediction for three different cases.

Alenezi et al. [Alenezi and Abunadi, 2015] compared the performance of different classification techniques in predicting vulnerable PHP files and proposed an application of these classification rules. They performed empirical studies on three large open source web-projects in which they investigated which 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.

An alternative method [Medeiros et al., 2014] to that previously discussed, which combines vulnerability detection and correction. They first considered detect vulnerabilities in PHP web applications using four different classifiers (J48, Nave Bayes, Logistic Regression, and MLP), then corrected these vulnerabilities with different procedures that are outside the scope of this paper. Yet, they mentioned that this approach was not optimal since it did not provide correct results.

## 8 Conclusions and Future Work

Software vulnerability prediction is considered an important phase in enhancing the software quality. These predictions help security engineers to forecast the future, i.e. to identify the software components, which are likely to have flaws. Data mining techniques were used to identify vulnerabilities in complete and new projects with no enough data using machine learning. Detection techniques using reliable classifiers were conducted for complete projects. As an outcome of this phase vulnerability models were created and tested on incomplete projects leading to accurately predicting vulnerability flaws in these inactive projects. Overall, this provides a great help to the software project management team to deal with those areas in the project on a timely basis and with sufficient effort. This paper has shown and analyzed how cross project vulnerability predication can be done.

Our future work will focus mainly on two aspects: collecting more open-source projects vulnerabilities to validate the generality of the proposed approach and considering these predictions in developing a rule-based firewall according the classification rules to filter vulnerable requests. This firewall will be able to distinguish vulnerable requests after evaluating some of the features of the PHP file.

## References

[Abunadi and Alenezi, 2015] Abunadi, I. and Alenezi, M. (2015). Towards cross project vulnerability prediction in open source web applications. In *Proceedings of the The International Conference on Engineering & MIS 2015*, page 42. ACM.

[Alenezi and Abunadi, 2015] Alenezi, M. and Abunadi, I. (2015). Evaluating software metrics as predictors of software vulnerabilities. *International Journal of Security and Its Applications*, 9(10):231–240.

[Alenezi and Khellah, 2015] Alenezi, M. and Khellah, F. (2015). Evolution impact on architecture stability in open-source projects. *International Journal of Cloud Applications and Computing (IJCAC)*, 5(4):24–35.

[Alpaydin, 2014] Alpaydin, E. (2014). *Introduction to machine learning*. MIT press.

[Bishop, 2005] Bishop, M. (2005). *Introduction to computer security*. Addison-Wesley Boston, MA.

[Chan and Paelinckx, 2008] Chan, J. C.-W. and Paelinckx, D. (2008). Evaluation of random forest and adaboost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery. *Remote Sensing of Environment*, 112(6):2999–3011.

[Chowdhury and Zulkernine, 2011] Chowdhury, I. and Zulkernine, M. (2011). Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57(3):294–313.

[Damiani et al., 2008] Damiani, E., Ardagna, C. A., and El Ioini, N. (2008). *Open source systems security certification*. Springer Science & Business Media.

[Fenton and Bieman, 2014] Fenton, N. and Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. CRC Press.

[Fernández-Delgado et al., 2014] Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

[Han et al., 2011] Han, J., Kamber, M., and Pei, J. (2011). *Data mining: concepts and techniques: concepts and techniques*. Elsevier.

[Hearst et al., 1998] Hearst, M. A., Dumais, S. T., Osman, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28.

[Hilbe, 2009] Hilbe, J. M. (2009). *Logistic regression models*. CRC Press.

[Lewis, 1998] Lewis, D. D. (1998). Naive (bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, pages 4–15. Springer.

[McGraw, 2006] McGraw, G. (2006). *Software security: building security in*, volume 1. Addison-Wesley Professional.

[Medeiros et al., 2014] Medeiros, I., Neves, N. F., and Correia, M. (2014). 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*, pages 63–74. ACM.

[Neuhaus et al., 2007] Neuhaus, S., Zimmermann, T., Holler, C., and Zeller, A. (2007). Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 529–540. ACM.

[Nguyen and Tran, 2010] Nguyen, V. H. and Tran, L. M. S. (2010). Predicting vulnerable software components with dependency graphs. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, page 3. ACM.

[Nyanchama, 2005] Nyanchama, M. (2005). Enterprise vulnerability management and its role in information security management. *Information Systems Security*, 14(3):29–56.

[OWASP, 2015] OWASP (2015). Top ten most critical web application security risks. *http://www.owasp.org*.

[Powers, 2011] Powers, D. M. (2011). Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation.

[Quinlan, 2014] Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

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

[Shar et al., 2014] Shar, L. K., Briand, L., and Tan, H. B. K. (2014). Web application vulnerability prediction using hybrid program analysis and machine learning. *IEEE Transactions on Dependable and Secure Computing*, (99):1.

[Shar and Tan, 2012] Shar, L. K. and Tan, H. B. K. (2012). Predicting common web application vulnerabilities from input validation and sanitization code patterns. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 310–313. IEEE.

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

[Shin and Williams, 2008] Shin, Y. and Williams, L. (2008). An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 315–317. ACM.

[Sprent and Smeeton, 2007] Sprent, P. and Smeeton, N. C. (2007). *Applied nonparametric statistical methods*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, Boca Raton, FL, 4th edition.

[Walden et al., 2010] Walden, J., Doyle, M., Lenhof, R., and Murray, J. (2010). Idea: java vs. php: security implications of language choice for web applications. In *Engineering Secure Software and Systems*, pages 61–69. Springer.

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

[Yamaguchi et al., 2012] Yamaguchi, F., Lottmann, M., and Rieck, K. (2012). Generalized vulnerability extrapolation using abstract syntax trees. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 359–368. ACM.

[Zhang et al., 2011] Zhang, S., Caragea, D., and Ou, X. (2011). An empirical study on using the national vulnerability database to predict software vulnerabilities. In *Database and Expert Systems Applications*, pages 217–231. Springer.