# Cross-project Vulnerability Prediction based on Software Metrics and Deep Learning

Ilias Kalouptsoglou, Miltiadis Siavvas*, Dimitrios Tsoukalas, and Dionysios Kehagias

Centre for Research and Technology Hellas, Thessaloniki, Greece
{iliaskaloup,siavvasm,tsoukj,diok}@iti.gr

**Abstract.** Vulnerability prediction constitutes a mechanism that enables the identification and mitigation of software vulnerabilities early enough in the development cycle, improving the security of software products, which is an important quality attribute according to ISO/IEC 25010. Although existing vulnerability prediction models have demonstrated sufficient accuracy in predicting the occurrence of vulnerabilities in the software projects with which they have been trained, they have failed to demonstrate sufficient accuracy in cross-project prediction. To this end, in the present paper we investigate whether the adoption of deep learning along with software metrics may lead to more accurate cross-project vulnerability prediction. For this purpose, several machine learning (including deep learning) models are constructed, evaluated, and compared based on a dataset of popular real-world PHP software applications. Feature selection is also applied with the purpose to examine whether it has an impact on cross-project prediction. The results of our analysis indicate that the adoption of software metrics and deep learning may result in vulnerability prediction models with sufficient performance in cross-project vulnerability prediction. Another interesting conclusion is that the performance of the models in cross-project prediction is enhanced when the projects exhibit similar characteristics with respect to their software metrics.

**Keywords:** Software Quality · Security · Vulnerability Prediction

## 1 Introduction

According to the ISO/IEC 25010 [1] International Standard on Software Quality, software security is one of the main quality characteristics of modern software products. In order to enhance the security of the produced software, software industries should ensure that their products are not bundled with critical vulnerabilities. The exploitation of a single vulnerability may lead to far-reaching consequences both for the end user (e.g., information leakage) and for the owning enterprise of the compromised software (e.g., financial losses and reputation damages) [2]. Hence, appropriate mechanisms are required in order to assist the

* Corresponding Author

identification and mitigation of software vulnerabilities as early in the Software Development Lifecycle (SDLC) of a software product as possible [3].

One such mechanism that enables the early identification and mitigation of software vulnerabilities is vulnerability prediction [3]. Vulnerability prediction, which is actually inspired by the idea of fault prediction [4] that is used for reliability assessment [5], is a subfield of software security aiming to predict software components that are likely to contain vulnerabilities (i.e. vulnerable components). Vulnerability prediction models (VPMs) are normally built based on Machine Learning (ML) techniques that use software attributes as input (e.g., software metrics [6–8]), to discriminate between vulnerable and neutral components. These models can be used for prioritizing testing and inspection efforts, by allocating limited test resources to potentially vulnerable parts [9].

Several VPMs have been proposed over the years utilizing various software factors as inputs for predicting the existence of vulnerable components, including software metrics [10], text mining [9, 11], and static analysis alerts [12, 13]. Although these models have demonstrated promising results in predicting the existence of vulnerabilities in the projects on which they have been trained (i.e., within-project vulnerability prediction), they failed to sufficiently predict the existence of vulnerabilities in previously unknown software projects (i.e., cross-project vulnerability prediction) [14, 3]. However, the adoption of advanced ML techniques like deep learning is expected to increase the predictive performance of existing models in cross-project vulnerability prediction. Despite some initial attempts (e.g., [15, 11]), more work is required in order to reach safer conclusions.

To this end, in the present paper we investigate whether the adoption of deep learning along with software metrics may lead to accurate and practical cross-project vulnerability prediction. For this purpose, we initially construct a vulnerability dataset by carefully restructuring the dataset provided by Walden et al. [16]. The resulting dataset contains 21,445 vulnerable and clean files, which were retrieved from three real-world open-source PHP applications. Subsequently, based on this dataset, several machine learning (including deep learning) models are constructed, both for the case of within-project vulnerability prediction and for the case of cross-project vulnerability prediction. In each one of these cases, the produced models are evaluated and compared with respect to their accuracy and practicality based on a set of performance metrics. Finally, feature selection is also applied in order to examine how the selected features affect the performance of the produced models.

The rest of the paper is structured as follows. Section 2 discusses the related work in the field of vulnerability prediction. Section 3 provides information about the adopted methodology, whereas Section 4 presents the results of our analysis. Finally, Section 5 concludes the paper and presents directions for future work.

## 2 Related Work

A large number of VPMs has been proposed in the literature over the past decade [3]. As stated in [14], the main VPMs that can be found in the literature

utilize software metrics [10, 17, 18], text mining [9, 15], and security-related static analysis alerts [12, 13] to predict vulnerabilities in software products. However, as stated before, while these models have demonstrated promising results in predicting the existence of vulnerabilities in the software projects on which they have been built (i.e., within-project vulnerability prediction), they have failed to demonstrate sufficient performance in cross-project vulnerability prediction. This is a major shortcomings of existing VPMs, affecting their practicality [3].

In fact, several empirical studies have shown that that text mining-based models exhibit better predictive performance compared to other state-of-the-art techniques [16, 19, 20], but they perform poorly in cross-project vulnerability prediction [13]. This can be explained by the fact that that these models base their predictions on the frequencies of specific text features (i.e., keywords) extracted directly from the source code (e.g., [9]), which makes them highly project-specific. On the contrary, VPMs based on software metrics have been found to be more promising solutions in cross-project vulnerability prediction [19, 16], as software metrics are able to capture more high-level characteristics of the analyzed code. Therefore, more work is required towards this direction.

Recently, several researchers have started examining whether the adoption of more advanced ML techniques, particularly deep learning, may lead to better VPMs. For instance in [15], the authors employed deep learning, and more precisely a Long-short Term Memory (LSTM) network [21], to construct new features from the vulnerability dataset provided by Scandariato et al. [9]. Subsequently, based on these features, they built both metric-based and text mining-based VPMs using Random Forest algorithm. The results of their experiments indicate that the new models are better than the models of Scandariato et al. [9], both in within-project and in cross-project vulnerability prediction. Similarly, Pang et al. [11] used deep neural networks to built text mining-based VPMs using four software projects retrieved from the dataset provided by Scandariato et al. [9]. Their results showed that deep learning-based VPMs demonstrate sufficient predictive performance in within-project vulnerability prediction.

Although these studies provide useful insights regarding the efficacy of deep learning in vulnerability prediction, they are hindered by several shortcomings. First of all, in [15], deep learning was used only for the construction of new features, while the produced models were built using the Random Forest algorithm. However, better results could have been achieved, if deep learning was adopted for the construction of the VPMs, especially in the case of cross-project vulnerability prediction. Secondly, although Pang et al. [11] built deep learning-based VPMs, they focused only on the case of within-project vulnerability prediction, not providing any insight on the performance of these models on cross-project vulnerability prediction. Finally, an important issue of both studies is that the vulnerability dataset that was utilized does not contain actual vulnerability data retrieved from real-world vulnerability databases. That is, the source code files of the software projects are marked as vulnerable (or clean), based on the outputs of a commercial static code analyzer (i.e., Fortify SCA[1]), and not based

[1] https://www.microfocus.com/en-us/products/static-code-analysis-sast/overview

on actual reported vulnerabilities. As a result, due to the fact that static code analyzers are prone to false positives and false negatives [20, 3], many of the files that are marked in the dataset as vulnerable may, in fact, be clean (and vice versa), obviously affecting the correctness of the produced models.

The main goal of the present study is to examine whether the adoption of deep learning along with software metrics may lead to VPMs with sufficient accuracy and practicality, especially in the case of cross-project prediction. Contrary to the previous studies, in the present work, we put significant emphasis on the predictive performance of the produced models in cross-project vulnerability prediction, since in this case, poor performance may hinder the practicality and acceptance of the produced models, whereas it is an open issue in the related literature that remains unresolved [3]. In addition, as opposed to the previous studies that were based on static analysis to classify the source code files as vulnerable or clean, in the present work, we construct our experiments based on real-world vulnerability data retrieved from actual vulnerability databases and vendor advisories. In particular, our work is based on the vulnerability dataset provided by Walden et al. [16], which contains actual vulnerability data retrieved from three popular open-source PHP web applications.

## 3 Methodology

### 3.1 Dataset

For the purposes of the present study, the vulnerability dataset provided by Walden et al. [16] was utilized. This dataset contains real-world vulnerability information retrieved from public databases, including both vendor advisories and the National Vulnerability Database[2] (NVD) for three popular PHP web applications, namely Drupal, PHPMyAdmin, and Moodle.

More specifically, to construct the dataset, Walden et al. [16] fetched multiple versions of these three open-source applications, and for each source code file they marked its vulnerability status based on whether there were relevant reports for these files on vendor and NVD security advisories. In brief, according to this approach, files that have at least one vulnerability reported on vulnerability databases are marked as vulnerable, whereas files that do not have any reported vulnerabilities are defined as neutral. It should be noted that files with no vulnerabilities are defined as neutral, and not as clean, because they may contain potential vulnerabilities that have not been reported yet.

For each one of the source code files of the dataset, 12 software metrics were computed, using a custom static code analyzer proposed by Walden et al. [16], which is based on the PHP compiler front-end developed by Vries and Gilbert [22]. The reasoning behind the adoption of this tool is twofold. Firstly, no tools are available on the market that are capable of computing a sufficient set of coupling, complexity, and size metrics of PHP files. Secondly, using this tool could enable us more reliably compare the results of the present work to the

[2] https://nvd.nist.gov/

results of the work of Walden et al. [16] on which the present study was based. The source code metrics collected for each version of every PHP file of the three selected software applications are listed below:

- **Lines of code (LoC)**: The total lines of code of the PHP file being measured, excluding lines that do not have PHP tokens.
- **Non-HTML lines of code (Nonecholoc)**: The same with the LoC metric, excluding lines containing HTML code embedded in the file being measured.
- **Number of functions (Nmethods)**: The total number of functions and methods of the PHP file being measured.
- **Cyclomatic Complexity (Ccom)**: The total number of linearly independent paths through a program's source code.
- **Maximum nesting complexity (Nest)**: The maximum nesting depth of loops and control structures of the PHP file being measured.
- **Halstead's volume (Hvol)**: An estimate of the program's size (i.e., volume) using the number of unique operators (i.e., method names and PHP language operators) and operands (i.e., parameter and variable names).
- **Internal functions or methods called (nIncomingCalls)**: The number of functions or methods defined in a PHP file that are invoked (at least once) by another method or function that belongs to the same file.
- **Fan-in (NIncomingCallsUniq)**: The number of files that invoke methods or functions of the PHP file being measured.
- **Fan-out (nOutgoingInternCalls)**: The number of files that contain methods and functions that are invoked by the PHP file being measured.
- **External calls to functions or method (nOutgoingExternCallsUniq)**: The number of files calling a particular function or method defined in the file being measured, summed across all functions and methods.
- **Total external calls (nOutgoingExternFlsCalled)**: The total number of statements that invoke functions or methods that are not defined in the PHP file being measured.
- **External functions or methods called (nOutgoingExternFlsCalledUniq)**: The total number of statements that invoke methods or functions that belong to the PHP file being measured.

The dataset that is used in the present study is based on the original dataset provided by Walden et al. [16]. However, while constructing it we followed a slightly different approach. More specifically, the original dataset consists of the computed metrics and the vulnerability status of each PHP file and version of the selected PHP applications. This results in a broader dataset of 112,947 samples. In the original study [16], the authors focused on a significantly smaller subset of the dataset, as they considered only one version of each PHP file. However, we observed that significant differences may exist in the source code of different versions of the same file. Therefore, in the present study, we decided to treat the different versions of the same file as different samples of the dataset, provided that a significant difference is observed between their computed metrics (as this indicates significant differences in their actual source code).

Hence, from the original dataset we kept only those files that were different in all the computed metrics. This ensures that the remaining samples refer to PHP files that are significantly different with respect to their source code. Even a small change in the source code of a PHP file (e.g., addition or removal of a single line of code in a function, addition or removal of a single square bracket, etc.) will lead to a difference in at least one of the file-level metrics. Requesting a difference to exist in all the studied metrics means that significant modifications need to be performed between commits in order these files to be considered different. In that way, we ensure that not only duplicates are removed, but also highly different files are considered as part of the dataset. This led to a dataset of 21,445 samples. Table 1 shows the descriptive statistics of the final dataset.

**Table 1.** Descriptive statistics of the final dataset

| Application | Vulnerable files | Total files |
|---|---|---|
| Drupal | 62 | 195 |
| PHPMyAdmin | 480 | 4732 |
| Moodle | 362 | 16536 |

### 3.2 Data Pre-processing

**Sampling Techniques and Scaling** As can be seen by Table 1, the produced dataset is highly imbalanced. In particular, the number of neutral files is significantly larger compared to the number of the files that are marked as vulnerable. This observation, which is in-line with the majority of similar research endeavors that are based on real-world vulnerability data (e.g., [14, 19, 13]) can be explained by the fact that only a small number of vulnerabilities are identified and reported on public databases throughout the lifetime of software applications [14].

Classification problems with imbalanced datasets are challenging, as the skewed distribution makes many conventional ML algorithms less effective, especially in predicting minority class examples. To eliminate this risk, similarly to other studies [16, 19], we applied under-sampling to make the dataset perfectly balanced. For this purpose, we employed the *RandomUnderSampler* offered by the sklearn[3] package to reduce the neutral samples to a number equal to the number of vulnerable samples. More specifically, we retained all the samples that belong to the minority class (i.e., vulnerable files) and randomly chose $N$ samples from the majority class (i.e., neutral files), where $N$ is the total number of the samples of the minority class. It is worth mentioning that under-sampling was applied only to the training set, as re-sampling on test data imposes a bias on the findings. This is crucial in order to preserve the correct testing conditions.

[3] https://scikit-learn.org/

In addition, we performed data normalization due to the fact that ML techniques (especially neural networks) produce better results when data are normalized. For this purpose, we used sklearn's *MinMaxScaler*, which transforms features by scaling each sample to a given range between zero and one.

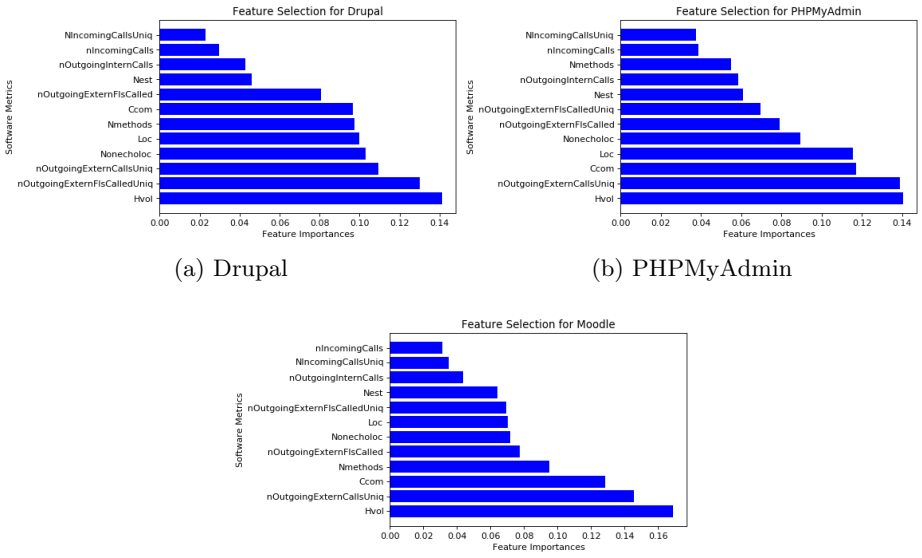
**Feature Selection** The selection of independent input variables (i.e., features) is a critical part in the design of ML algorithms. Each additional feature makes the model more complicated by adding an extra dimension. A large number of input variables may lead to the "curse of dimensionality", a phenomenon in which the model's performance degrades as the inputs' number increases. Feature selection is a strong weapon against the curse of dimensionality, as it reduces both the computational cost of modeling and training time. In many cases, feature selection can even improve the performance of the model, since irrelevant features can negatively affect the model performance.

In order to study the statistical significance of each file-level software metric over the existence of vulnerabilities, we applied an embedded method named Tree-based Elimination (TBE) [21]. TBE uses the built-in feature selection mechanism of the Random Forest algorithm to measure the importance of each feature. Importance provides a score that indicates how valuable each feature is in the construction of the decision trees within the model. We chose Random Forest as it is one of the most popular methods for feature selection [23].

We applied the TBE method independently on the feature set of each application of our dataset and ranked the 12 metrics (features) presented in Section 3.1 by the order that they were selected by the method. The results are illustrated in Figure 1. In particular, in Figure 1a, Figure 1b and Figure 1c, we present the importance of each metric in vulnerability prediction for Drupal, PHPMyAdmin and Moodle respectively, as computed by the TBE method.

By having a look at Figures 1a, 1b and 1c, we can clearly see that the ranking of metrics is highly similar for PHPMyAdmin and Moodle, whereas the top three metrics (i.e., Hvol, nOutgoingExternCallsUniq, and Ccom) are identical for both applications. More specifically, the importance of Hvol in vulnerability prediction was found to be 0.140226 and 0.171136 for PHPMyAdmin and Moodle respectively. The importance of nOutgoingExternalCallsUniq metric equals to 0.139171 for PHPMyAdmin and 0.147026 for Moodle, while the importance of Ccom was found to be 0.118421 and 0.126714 for PHPMyAdmin and Moodle respectively. In addition to the top metrics, the rest of the ranking is also highly correlated, indicating that a model trained to predict vulnerabilities in one application, may as well perform equally good in the other. Regarding Drupal, the Hvol metric is also ranked first with an importance of 0.141249. In the second place, however, we can see the nOutgoingExternFlsCalledUniq metric (with an importance of 0.129682), while the nOutgoingExternCallsUniq metric that is ranked second for PHPMyAdmin and Moodle, in this case is ranked third (with an importance of 0.106244).

We believe that Hvol is considered the most important metric by the TBE as it indicates how many distinct operators and operands are used. Thus, if all




(a) Drupal (b) PHPMyAdmin

(c) Moodle

**Fig. 1.** Feature selection for each one of the software applications of the selected dataset, namely (a) Drupal, (b) PHPMyAdmin, and (c) Moodle.

the operators and operands are used in a project, as opposed to only using the safe ones, the Hvol value will be directly affected. The same holds for the Ccom metric. If we use only the safe linearly independent paths instead of using all of them, it will have an important variation in Ccom's value.

It should be noted, that in the case of within-project vulnerability prediction (see Section 4.1), for the construction of the application-specific models, the top three features of their corresponding application are used. However, in the case of cross-project prediction (see Section 4.2), the top three features of the dominant projects, i.e., PHPMyAdmin and Moodle, are used (i.e., the Hvol, nOutgoingExternCallsUnique, and Ccom metrics) for each one of the three studied cases, in order to investigate whether the application of a vulnerability prediction model, which was built based on a specific application, on previously unknown applications that demonstrate similarities with respect to the importance of their features in the existence of vulnerabilities may affect its prediction performance.

### 3.3 Model Selection and Performance Metrics

**Selection of Classifiers** In the present study different ML algorithms are used in order to build models able to discriminate between vulnerable and neutral source code files. Contrary to the previous work of Walden et al. [16], in which emphasis was given only on the Random Forest algorithm, in the present work

we also investigate additional ML algorithms, including Support Vector Machine (SVM), XGBoost, and an ensemble method that combines various ML algorithms. We also examine the ability of deep learning, specifically the Multi-Layer Perceptron (MLP), to provide reliable vulnerability predictions.

For the construction of these models, hyperparameter tuning is initially employed in order to determine the optimal parameters for each model. More specifically we use the Grid-search method [24], which is commonly used to find the optimal hyper-parameters of a model, by performing an exhaustive search over specified parameter values for an estimator. Subsequently, we train the models using the dataset presented in Section 3.1. For instance, for the case of MLP, several models were constructed using the hyperparameters presented in Table 2. It should be noted that the same parameters were used for the MLP models that were built both for the case of within-project and cross-project vulnerability prediction. The performance of the produced models is compared using the performance parameters presented in the rest of this section. Based on these performance metrics the best model in each case is determined.

**Table 2.** The selected hyperparameters of the Multi-layer Perceptrons (MLPs) that are constructed in the present study.

| **Hyperparameter Name** | **Value** |
|---|---|
| **Number of Layers** | 4 |
| **Number of Hidden Layers** | 3 |
| **Number of Hidden Units (per Hidden Layer)** | 1000/500/50 |
| **Weight Initialization Technique** | Glorot Xavier |
| **Learning Rate** | 0.01 |
| **Gradient Descent Optimizer** | Adagrad |
| **Batch Size** | 128 |
| **Activation Function** | relu |
| **Output Activation Function** | sigmoid |
| **Loss Function** | Cross-entropy |
| **Over-fitting Prevention** | Dropout = 0.15 (per layer) |

**Evaluation Metrics** Several performance indicators are available in the literature and are commonly used for evaluating the predictive performance of ML models. These performance indicators are normally computed based on the number of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) that are generated by the produced models.

Similarly to previous works in vulnerability prediction [9, 13, 14, 16, 15], we put specific emphasis on the Recall ($R$) of the produced models, as the higher the Recall of the model, the more actual vulnerabilities it predicts. Recall is described as follows:

$$R = \frac{TP}{TP + FN} \tag{1}$$

Apart from the ability of the produced models to accurately detect the vast majority of the vulnerable files that a software product contains, it is important to take into account the volume of the produced FP, (i.e., neutral files that are marked as vulnerable by the models), since they are known to affect the models' practicality. A large number of FP forces the developers to inspect a non-trivial number of non-vulnerable files, in order to detect a file that is actually vulnerable. Hence, the number of FP is associated to the manual effort required by the developers for identifying files that actually contain vulnerabilities.

In order to measure the impact of the produced FP, similarly to [16], we use the *Inspection Ratio* ($I$), which is the percentage of files that one has to consider (i.e., inspect manually) to make it possible to find the TP identified by the model. This performance metric, which actually corresponds to the effort required for identifying a vulnerable file, receives values between 0 and 1, and is given by the following formula:

$$I = \frac{TP + FP}{TP + NP + FP + FN} \tag{2}$$

These two performance metrics are used as the basis for the comparison of the produced VPMs, as well as for the selection of the best model in each one of the studied cases that are presented in Section 4, as their combination provides a complete picture of the model performance in predicting vulnerable files. In fact, *Recall* ($R$) indicates how effectively the produced models detect TP, whereas the *Inspection Ratio* ($I$) indicates how efficient the model is in predicting TP, based on how many FP have to be triaged by the developer until a TP is detected.

### 3.4 Strategy

**Within-project Prediction** The first step of our study is to investigate the performance of the selected ML algorithms described in Section 3.3 in predicting the occurrence of vulnerabilities in the software applications on which they have been trained. To achieve this, we trained the selected models on each application of our dataset, namely Drupal, PHPMyAdmin, and Moodle, and evaluated them based exclusively on data retrieved from the corresponding application.

For the evaluation of the models we decided to employ k-fold cross-validation. In k-fold cross-validation, the dataset is split into k folds from which the k-1 participate in training and the remaining one participates in evaluation. The fold that remains for the evaluation is different every time. In this way, we have a model that is trained and evaluated k times using each time different training and testing data. The model performance is the average performance of these k models. In that way, we reduce the possibility of having biased results.

We chose to perform a 3-fold cross-validation in order to be in line with the work of Walden et al. [16]. More specifically, we used the sklearn's stratified cross-validation in order to construct folds that retain the percentage of samples for each class, thus creating realistic conditions for model training and validation. The 3-fold cross-validation was repeated 10 times for each model, after shuffling the dataset randomly before each iteration. Hence, the computed performance

metrics (i.e., *Recall* and *Inspection Ratio*) of each model are the average values of 30 models trained and evaluated on the same dataset. This enhances the reliability of the produced results, as the potential bias introduced by the selection of non-representative subsets of the broader dataset is avoided.

**Cross-project Prediction** During the second part of this study, we examined the predictive performance of the software metrics-based VPMs, constructed using the ML algorithms presented in Section 3.3, in cross-project vulnerability prediction. In cross-project vulnerability prediction, emphasis is given on the ability of a given vulnerability prediction model to accurately predict vulnerabilities in previously unknown software projects (i.e., applications).

For this purpose, we trained several ML models based on the data of two out of the three PHP applications of the dataset (i.e., by merging their vulnerability data into a single dataset), and tested them on the vulnerability data of the remaining application. We repeated this experiment three times, covering the cases described in Table 3 (i.e., combinations of training and testing data).

**Table 3.** The studied cases of cross-project vulnerability prediction.

| Case | Training Set | Testing Set |
|---|---|---|
| Case 1 | Drupal+PHPMyAdmin | Moodle |
| Case 2 | Drupal+Moodle | PHPMyAdmin |
| Case 3 | Moodle+PHPMyAdmin | Drupal |

The reasoning behind the decision of merging vulnerability data of two applications for constructing the training set in each one of the studied cases is twofold. Firstly, the combination of data retrieved from various software projects for training enhances the generalizability of the produced models, as the models will be able to capture patterns from several applications (instead of one), which is expected to lead to better cross-project vulnerability prediction. Secondly, by merging vulnerability data from various projects the size of the training set is increased, which is critical for the performance of deep learning models. It should be noted that in the case of cross-project vulnerability prediction, 3-fold cross-validation was not required, as a completely independent test set (i.e., the vulnerability data of the remaining application) was used for the evaluation.

## 4 Results and Discussion

### 4.1 Within-project Prediction

In this section, we present the results of within-project vulnerability prediction. Table 4 reports the evaluation results of the classifiers that were build based exclusively on the vulnerability data of each one of the three PHP applications. These results were obtained after applying the cross-validation approach

described in Section 3.4. In brief, the values on the Table 4 depict the average performance of the models over 10 executions of the 3-fold cross-validation experiment. As mentioned in Section 3.4, we considered *Recall* and *Inspection Ratio* as the main performance indicators, as they provide a complete picture of the accuracy and practicality of the produced models (see Section 3.3).

**Table 4.** Classification results of the projects.

| Application | Indicator | SVM | RF | MLP | XGBoost | Ensemble |
|---|---|---|---|---|---|---|
| Drupal | Recall | 71.17 | **77.11** | 75.32 | 76.27 | 75.76 |
| | Inspection Rate | 41.44 | **47.64** | 44.92 | 44.58 | 46.35 |
| PHPMyAdmin | Recall | 82.38 | **90.25** | 87.57 | 85.26 | 90.62 |
| | Inspection Rate | 17.59 | **19.13** | 21.47 | 22.84 | 19.77 |
| Moodle | Recall | 81.77 | **92.13** | 84.63 | 86.81 | 91.16 |
| | Inspection Rate | 7.78 | **16.65** | 17.97 | 19.41 | 15.74 |
| Combined | Recall | 78.34 | **90.32** | 85.25 | 83.54 | 90.45 |
| | Inspection Rate | 18.74 | **17.30** | 17.81 | 23.95 | 18.88 |

As can be seen by Table 4, in the case of Drupal, all classifiers, except for SVM have similar performance metrics, with RF being slightly better with respect to Recall. In the case of PHPMyAdmin, we notice that the RF classifier shows the best performance in terms of Recall (90.25%) and Inspection Ratio (19.13%). In the case of Moodle, RF demonstrates again the best performance, with a Recall of 92.13% and an Inspection Ratio of 16.65%, but competes closely with the stacking ensemble method (which has a Recall of 91.16% and an Inspection Ratio of 15.74%). These results are in bold font.

As can be seen by Table 4, RF is the best classifier in all the studied cases. This indicates that metric-based VPMs that are constructed using the RF algorithm exhibit sufficient predictive performance and practicality in within-project vulnerability prediction. This observation is in line with the the results of other similar studies (e.g., [19, 13, 9, 15]), including the work of Walden et al. [16]. In all these research endeavors, RF was found to be the most promising model for the construction of project-specific VPMs. It should be noted that a direct comparison of our evaluation results with those of Walden et al. [16] would not be correct, since the original dataset was restructured (see Section 3.1), whereas a different pre-processing process was employed (see Section 3.2). Finally, another interesting observation is that the MLP-based VPMs also demonstrate sufficient predictive performance, which suggests that they may also constitute a promising solution for within-project vulnerability prediction.

### 4.2 Cross-project Prediction

In this section we present the results of cross-project vulnerability prediction. As described in detail in Section 3.4, we investigate three individual cases (see Table 3). In each case, we merge the vulnerability data of two PHP applications

and we test the produced models on the vulnerability data of the remaining one. Feature selection is also applied in order to improve the predictive performance of the produced models. It should be noted that the utilization of all the 12 features, did not lead to any good model in cross-project prediction. However, by applying feature selection (see Section 3.2) we achieved much better results.

More specifically, for each one of the cases presented in Table 3, we built several ML models. Emphasis was given on the RF algorithm that was found to be the best algorithm in the case of within-project vulnerability prediction, as well as on the MLP, as the specific interest of the present work is to examine whether deep learning can lead to better cross-project vulnerability prediction. As already mentioned in Section 3.2, for each one of the three studied cases, we took into account the top three features of the dominant projects in the training set, i.e., PHPMyAdmin or Moodle. It should be noted that in the case of PHPMyAdmin and Moodle the top three features are exactly the same. We built the models using two projects and evaluated them using the complete vulnerability data of remaining project. In what follows, we report the performance metrics of the two best-performing models, namely RF and MLP. In particular, in Table 5 we present the results of RF and MLP for cross-project prediction after feature selection for each one of the three studied cases (see Table 3).

**Table 5.** The evaluation results of the classifiers that were built using the Random Forest (RF) and the Multi-layer Perceptron (MLP) algorithms in the case of cross-project vulnerability prediction.

| **Project** | **Indicator** | **RF** | **MLP** |
|---|---|---|---|
| Drupal & PHPMyAdmin (train) – Moodle (test) | Recall | 56 | 80 |
| | Inspection Ratio | 33.99 | 38.16 |
| Drupal&Moodle (train) – PHPMyAdmin (test) | Recall | 53 | 72 |
| | Inspection Ratio | 39.77 | 42.29 |
| PHPMyAdmin & Moodle (train) – Drupal (test) | Recall | 47 | 52 |
| | Inspection Ratio | 35.90 | 28.21 |

An interesting observation obtained by Table 5 is that RF, although it was found to be the best model in within-project vulnerability prediction, does not demonstrate sufficient predictive performance in cross-project vulnerability prediction (i.e., the Recall of the produced models does not exceed the value of 56% in all the studied cases). This is in line with the findings of Walden et al. [16] who also observed that the predictive performance of RF drops when it comes to cross-project vulnerability prediction. On the contrary, MLP demonstrates higher Recall than RF in all the three cases. In addition, in the case where Drupal and PHPMyAdmin were used for training and Moodle for testing, as well as in the case where Drupal and Moodle were used for training and PHPMyAdmin for testing, the observed Recall was found to be above 70%, which is considered sufficient for cross-project vulnerability prediction. This indicates that deep learning may provide better VPMs than RF.

Focusing on the MLP models themselves, from Table 5 we can see that in the first two cases the observed Recall is higher than 70%, whereas in the third case the Recall was found to be only 52%. This can be explained by the fact that in the first two cases the models were built and subsequently tested on software projects that were similar with respect to the relevance (i.e., importance) of their features to the existence of vulnerabilities. More specifically, as was discussed in Section 3.2, PHPMyAdmin and Moodle had a very similar feature ranking, whereas their top three features were identical (Hvol, nOutgoingExternCallsUniq, and Ccom). Hence, in these two cases, both in the training set and in the testing set there were projects with similarities with respect to their feature ranking. On the contrary, in the third case, the projects in the training set (i.e., PHPMyAdmin and Moodle) and the project in the test set (i.e., Drupal), are not very similar with respect to their feature ranking, whereas their top three features (which were used as the models' inputs) are not identical. This indicates that the relative importance of the selected features (i.e., metrics) on the existence of vulnerabilities may affect the predictive performance of the produced models in cross-project vulnerability prediction.

In simple words, satisfactory cross-project vulnerability prediction can be achieved, if a model is built on a specific set of projects and applied on a previously unknown software project that exhibits similar characteristics regarding the relative importance of the selected metrics to the existence of vulnerabilities. Hence, several VPMs can be built based on the available data and then, the most suitable model can be applied to a new software project, based on whether the inputs (i.e., features) of the model are within the most important features of the new project, with respect to vulnerability identification.

Contrary to previous research endeavors in the field of vulnerability prediction [14, 3] that demonstrated bad predictive performance in cross-project prediction, the results of the present work indicate that the adoption of software metrics with deep learning may lead to better results, especially when feature selection is employed. This, in fact, constitutes the main contribution of the present work.

## 5 Conclusion and Future Work

In this paper, we investigated the ability of software metrics to be used as the basis for the construction of VPMs, with sufficient accuracy in cross-project vulnerability prediction. For this purpose, we constructed a dataset by properly restructuring the vulnerability dataset provided by Walden et al. [16], which contains real-world vulnerability data (i.e., 21,445 files) retrieved from three popular PHP web applications, namely Drupal, PHPMyAdmin, and Moodle. Several ML, including deep learning, models were built covering both the cases of within-project and cross-project vulnerability prediction. Feature selection was also employed using the popular TBE [21] technique, in order to detect important features and to examine whether feature selection affects the performance of the produced models in the case of cross-project vulnerability prediction.

The results of our experiments showed that RF is the best model in the case of within-project vulnerability prediction, which is in line with the findings of Walden et al. [16]. On the contrary, Deep Learning was found to be the best solution in the case of cross-project vulnerability prediction, as MLP-based models demonstrated sufficient levels of recall and inspection ratio, especially compared to the RF models. This indicates that deep learning and software metrics constitute a promising solution for cross-project vulnerability prediction. Another interesting observation is that the performance of the models in cross-project vulnerability prediction tends to be affected by feature selection. In fact, models that were built based on a specific set of software projects seem to provide better results when applied to new software projects that exhibit similarities with respect to the relevance of their features to the existence of vulnerabilities.

Several directions for future work can be identified. First of all, the present study was based solely on PHP web applications. In order to investigate the generalizability of the produced results, we are planning to replicate our study using software products that are written in other programming languages (e.g., Java, C/C++, etc.) and that belong to different domains (e.g., healthcare). Secondly, although the present analysis was based on a relatively large vulnerability dataset comprising 21,445 files, these files were actually retrieved from three real-world PHP applications. Hence, in the future, we are planning to extend our analysis by enriching this dataset with vulnerability information retrieved from additional PHP applications, in order to ensure that the results are not biased towards specific applications. Finally, we are also planning to replicate our study by using metrics calculated at lower levels of granularity, such as class- or function- level, in order to further investigate the generalizability of our results.

## References

1. ISO/IEC: ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. ISO/IEC (2011)
2. Gelenbe, E., Görbil, G., Tzovaras, D., Liebergeld, S., Garcia, D., Baltatu, M., Lyberopoulos, G.: Nemesys: Enhanced network security for seamless service provisioning in the smart mobile ecosystem. In: Information Sciences and Systems 2013. Springer (2013) 369–378
3. Siavvas, M., Gelenbe, E., Kehagias, D.: Static analysis-based approaches for secure software development. Security in Computer and Information Sciences (2018) 142
4. Kumar, L., Misra, S., Rath, S.K.: An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Computer Standards & Interfaces **53** (2017) 1–32
5. Shukla, S., Behera, R.K., Misra, S., Rath, S.K.: Software Reliability Assessment Using Deep Learning Technique. Towards Extensible and Adaptable Methods in Computing (2018)
6. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on software engineering **20**(6) (1994) 476–493
7. Misra, S., Akman, I., Colomo-Palacios, R.: Framework for evaluation and validation of software complexity measures. IET software **6**(4) (2012) 323–334

8. Misra, S., Adewumi, A., Fernandez-Sanz, L., Damasevicius, R.: A suite of object oriented cognitive complexity metrics. IEEE Access **6** (2018) 8782–8796
9. Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W.: Predicting vulnerable software components via text mining. IEEE Transactions on Software Eng. (2014)
10. Zhang, M., de Carnavalet, X.d.C., Wang, L., Ragab, A.: Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security. IEEE Transactions on Information Forensics and Security (2019)
11. Pang, Y., Xue, X., Wang, H.: Predicting Vulnerable Software Components through Deep Neural Network. Proceedings of the 2017 International Conference on Deep Learning Technologies - ICDLT '17 (2017) 6–10
12. Tang, Y., Zhao, F., Yang, Y., Lu, H., Zhou, Y., Xu, B.: Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-Aware Perspective. IEEE International Conference on Software Quality, Reliability and Security (2015)
13. Yang, J., Ryu, D., Baik, J.: Improving vulnerability prediction accuracy with Secure Coding Standard violation measures. 2016 International Conference on Big Data and Smart Computing, BigComp 2016 (2016) 115–122
14. Jimenez, M., Papadakis, M., Traon, Y.L.: Vulnerability Prediction Models : A case study on the Linux Kernel. In: 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM). (2016) 1–10
15. Dam, H.K., Tran, T., Pham, T.T.M., Ng, S.W., Grundy, J., Ghose, A.: Automatic feature learning for predicting vulnerable software components. IEEE Transactions on Software Engineering (2018) 1–1
16. Walden, J., Stuckman, J., Scandariato, R.: Predicting vulnerable components: Software metrics vs text mining. In: 2014 IEEE 25th International Symposium on Software Reliability Engineering. (Nov 2014) 23–33
17. Ferenc, R., Hegedűs, P., Gyimesi, P., Antal, G., Bán, D., Gyimóthy, T.: Challenging machine learning algorithms in predicting vulnerable javascript functions. In: Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. (2019)
18. Siavvas, M., Kehagias, D., Tzovaras, D.: A preliminary study on the relationship among software metrics and specific vulnerability types. In: 2017 International Conference on Computational Science and Computational Intelligence. (2017)
19. Tang, Y., Zhao, F., Yang, Y., Lu, H., Zhou, Y., Xu, B.: Predicting Vulnerable Components via Text Mining or Software Metrics? An Effort-Aware Perspective. Proceedings - 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015 (2015) 27–36
20. Jimenez, M., Rwemalika, R., Papadakis, M., Sarro, F., Le Traon, Y., Harman, M.: The importance of accounting for real-world labelling when predicting software vulnerabilities. In: 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. (2019)
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comp. (1997)
22. de Vries, E., Gilbert, J.: Design and implementation of a php compiler front-end. dept. Technical report, TR-2007-47, Trinity College Dublin (2007)
23. Genuer, R., Poggi, J.M., Tuleau-Malot, C.: Variable selection using random forests. Pattern recognition letters **31**(14) (2010) 2225–2236
24. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS). (2015)