The Impact of Release-based Validation on Software Vulnerability Prediction Models

Giulia Sellitto
Department of Computer Science
University of Salerno
Italy
g.sellitto21@studenti.unisa.it

Filomena Ferrucci
Department of Computer Science
University of Salerno
Italy
fferrucci@unisa.it

ABSTRACT

Software vulnerability prediction models represent a promising approach in security defect analysis, since they allow to focus testing and improve software quality. Several studies have proposed Machine Learning approaches which demonstrated satisfying performance when evaluated using K-fold Cross-validation. However, recent research demonstrated that, when applying a release-based validation strategy instead, the performance declined. We want to investigate this issue, by conducting a comparative study on different models. We analyze the impact of using a release-based validation approach on vulnerability prediction models formerly evaluated using cross-validation. We rely on an existing dataset to evaluate two prediction models that exploit code metrics and textual features, respectively. We confirm that the release-based validation approach leads to generally lower performance, highlighting that further research would be needed to make vulnerability prediction models more effective.

CCS CONCEPTS

• **Software and its engineering** → **Software defect analysis**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Software Vulnerabilities, Machine Learning, Prediction Modelling

ACM Reference Format:
Giulia Sellitto and Filomena Ferrucci. 2021. The Impact of Release-based Validation on Software Vulnerability Prediction Models. In *MaLTeSQuE'21: International Workshop on Machine Learning Techniques for Software Quality Evolution, August 23–27, 2021, Athens, Greece.* ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

Software vulnerabilities are a critical threat to software systems, since a single malicious exploit can cause significant damage to people and organizations. It is paramount to release code that has been checked to contain no vulnerabilities, but this requires considerable testing effort. It would be convenient to focus testing on source code portions that are more likely to contain vulnerabilities, but finding such portions is not a trivial task [12] [16]. Several approaches for vulnerability discovery have been proposed: the most promising seem to be those based on Machine Learning prediction models [6]. These approaches are based on the fact that a classifier can learn from vulnerabilities that are known to be present in the software, and then predict whether an unseen portion of code is likely to contain vulnerabilities. Unfortunately, as pointed out by Jimenez et al. [10], most studies investigate the performance of vulnerability prediction models by using K-fold Cross-validation as the evaluation method. This technique, depicted in Figure 1, consists in splitting the dataset in k equal-sized folds and performing k rounds of validation. In each round, k-1 folds compose the training set and one fold serves as the test set. This arrangement ensures that each fold is used for testing once and for training k-1 times. The overall performance of the model is evaluated as the average of the performance obtained in the k rounds of validation. While this validation method can provide initial indications on the accuracy of prediction models, in a real-case scenario vulnerability data become available following time constraints, i.e., we cannot train the models on future data to obtain predictions on current data. We want to investigate how software vulnerability prediction models' performance change when using release-based validation rather than k-fold cross-validation as the evaluation method. We perform our study on the two models proposed by Walden et al. [22], which are based on software metrics and text tokens, respectively. These models have been evaluated by the authors on the Walden et al.'s PHP vulnerability dataset [21] using 3-fold cross-validation. We initially replicate their study, i.e., we evaluate the same models on the same dataset using cross-validation, in order to have a baseline to which subsequently compare the performance obtained by applying a release-based validation method. Also this latter technique consists in several rounds, each focused on one release of the software. At each round, the considered release is used for the testing phase, and the model is trained using the data relative to the previous releases [20] [19] [18]. Figure 2 illustrates this evaluation method. We run 555 experiments to compare how the performance change when employing different models and validation methods for vulnerability prediction. Since the dataset that we work on is highly unbalanced, i.e., the presence of vulnerable files is quite rare with respect to the total number of files, we also investigate the impact of different data balancing techniques, i.e., undersampling and oversampling. We observe that the performance of the models evaluated with the release-based approach is generally lower than the performance obtained when using cross-validation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MaLTeSQuE'21, August 23-27, 2021, Athens, Greece
© 2021 Association for Computing Machinery.

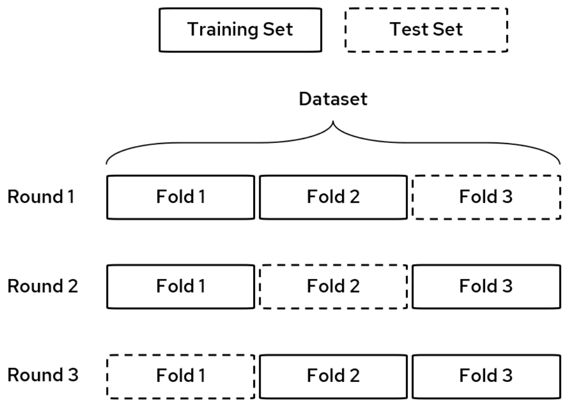

Figure 1: K-fold Cross-validation.

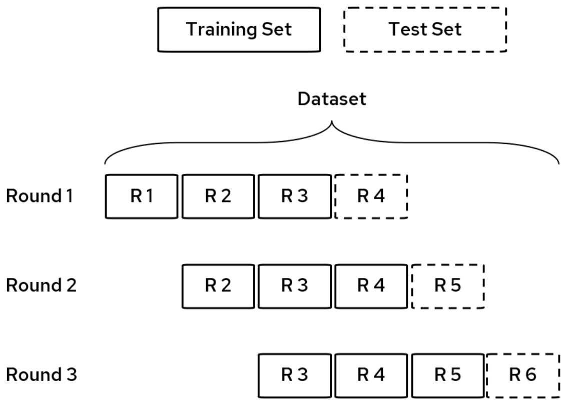

Figure 2: Release-based validation.

2 RELATED WORK

Software vulnerability prediction using Machine Learning has been a *hot topic* for many years [6]. Several prediction models have been proposed by the research community, whose members are interested in analyzing how effectively they would perform in a software developing context. One of the first studies investigating a Machine Learning approach for vulnerability prediction [15] focused on considering file imports, function calls and past information as predictors of vulnerability at component level. Most of the vulnerability prediction models that have been proposed during the following years operate at file-level granularity [6]. [13] and [14] considered developer activity metrics as predictors of the presence of security vulnerabilities in the source code of the Linux kernel, the PHP programming language and the Wireshark network analyzer, and found out that the developer activity metrics can indeed be used as indicators for source code file vulnerability. Other works [20] [19] deeper explored the predictive power of other metrics, such as complexity metrics and code churn metrics. They based their investigation on the idea that complex code is difficult to understand and test, and frequent or large code changes can introduce vulnerabilities. Complexity metrics have been also studied [22] along with other product metrics, such as number of lines of code, indicators of code volume and so on. They proposed two models for vulnerability prediction, one based on software metrics and the other based on text tokens. The latter consists in the *Bag of Words* approach, which has been also applied in previous work [8] [17]. The performance obtained from the two models have been evaluated with 3-fold cross-validation on a dataset [21] containing vulnerabilities found in several releases of three open-source PHP projects, i.e., PHPMyAdmin, Moodle and Drupal. The same dataset has been used in several later studies. [4] carried out a large-scale empirical analysis, considering 48 different prediction models with the aim of determining the best method in terms of output performance. [11] evaluated the impact of choosing different settings when building a model. They compared the performance obtained considering 7 classifiers and 4 data balancing techniques to deal with the imbalance of the dataset. The considered data balancing techniques were chosen among the most popular in the literature, i.e., different variations of the Synthetic Minority Oversampling TEchnique (SMOTE) [3] and its extension ADASYN [7]. [11] employed software metrics, text tokens and a combination of the two as predictors. Also [23] experimented the utilisation of the two predictors jointly, proposing an original approach based on a composite prediction model that operates on two levels. First it builds 6 underlying independent classifiers, and then it constructs a meta-classifier to combine the outputs of the 6 underlying classifiers. Another multi-level solution has been proposed by [1], who deployed a vulnerability prediction web service on Microsoft Azure cloud computing platform. A recent study published in 2019 [10] highlighted a problem that arises from the application of such method to evaluate vulnerability prediction models. It has been argued that researchers work under the so-called *perfect labelling assumption*, i.e., they assume, unconsciously or indirectly, that vulnerability data is always available. This is due to the fact that once a vulnerability is known to affect a file, that file in the dataset is labelled as vulnerable from the time at which the vulnerability was introduced in the source code onwards. In a real context, this is not feasible: vulnerabilities are discovered or reported only after a certain period of time that is subsequent to the moment they have been introduced. Hence, when evaluating a prediction model, one should use *real-world labelling* and consider to train the model at a certain time *t*, using only the data available at *t*, i.e., vulnerabilities that have already been discovered before *t*. In their experiments, [10] compared the performance output by vulnerability prediction models under the *perfect labelling assumption* with the performance obtained when considering *real-world labelling* and a release-based validation approach. They discovered that, when evaluated in a scenario that is more similar to the real operating context, vulnerability prediction models did not perform as well as one would wish. Other studies investigated release-based validation approaches, but still under the *perfect labelling assumption*. [18] focused their work on 20 Android applications and employed the *Bag of Words* method based on text tokens. Another study [9] on the Linux kernel dataset applied a release-based validation method, but without considering the mislabelling noise. Recent work [2] also suggested the use of Deep Learning models to enhance the predictive power of the existing proposals. Additional solutions to the vulnerability prediction problem consist in original approaches, for example the one [5] based on Machine Translation. An automatic translator is trained with vulnerable samples of code and their respective fixes, and is capable of producing a corrected version of new samples which are given as input to it. The prediction consists in feeding the translator unseen source code, and if it generates a translation, i.e., a fix, the code is likely to be vulnerable.

3 RESEARCH QUESTIONS

Our research consists in taking some steps to get from the traditional cross-validation approach, which has been broadly used to evaluate vulnerability prediction models in research contexts [6], toward the more realistic one suggested by Jimenez et al. [10]. In a real-case scenario, one would train the model using only the data referred to vulnerabilities already discovered in prior releases of the software and make predictions on the latest code to be released. The *goal* of our study is to compare the performance of vulnerability prediction models evaluated with cross-validation and with a relase-based approach, with the *purpose* of investigating the suitability of such models in a real scenario. The *perspective* is of both practitioners and researchers: the former are interested in understanding whether such approaches based on machine learning can be used in their work context; the latter are interested in evaluating pros and cons of the use of prediction models for vulnerability analysis. We investigate how a release-based validation method would impact the performance of models formerly evaluated using cross-validation.

RQ$_1$: *What is the performance of vulnerability prediction models validated using a release-based approach when compared to models evaluated using cross-validation?*

We focus our research on the two models proposed by Walden et al. [22], i.e., the one based on software metrics and the one based on textual features. Since we expect a difference between the performance obtained with cross-validation and with the release-based approach, we want to analyze to what extent each model's performance is influenced by the selection of a different validation method.

RQ$_2$: *Which modelling approach is more sensitive to the use of a different validation method?*

For our investigation we use the popular Walden et al.'s PHP vulnerability dataset [21]. Since the dataset is highly unbalanced, i.e., vulnerabilities are not much frequent, we want to examine the use of data balancing techniques, such as undersampling and oversampling. Undersampling consists in removing samples from the majority class in order to equalize the number of elements in the minority class. Oversampling consists in augmenting the minority class by adding to the dataset several synthetic vulnerable elements. We want to study the impact of such techniques on the models' performance.

RQ$_3$: *Which data balancing technique leads to better performance?*

Vulnerability Category	Number of Samples
Code Injection	10
CSRF	1
XSS	45
Path Disclosure	12
Authorization Issues	6
Other	1
Total	75

Table 1: Distribution of vulnerabilities in the dataset

4 METHODOLOGY

4.1 Dataset

The *context* of our research is represented by the PHP vulnerability dataset [21] which contains vulnerability data mined from three popular open-source PHP applications, i.e., PHPMyAdmin, Moodle and Drupal. The dataset was published in 2014 along with Walden et al.'s proposal of two vulnerability prediction models, one based on software metrics and the other based on text tokens [22]. For our investigation we need data related to several releases of the same application, but only release 6.0.0 of Drupal is provided by the dataset. Moreover, due to compilation problems faced with Moodle, we decide to focus our study only on the PHPMyAdmin dataset. However, this represents a preliminary work and further larger-scale analyses are already part of our future research agenda. The dataset includes data relative to 95 releases of PHPMyAdmin, between 2.2.0 and 4.0.9. The authors of the dataset collected source code files of major (e.g. v2.0), minor (v2.1), and subminor (v2.1.5) versions of the application from the public Git repository and mined vulnerability information from the security announcements maintained by the project itself. The dataset contains 75 vulnerabilities, categorized as shown in Table 1.

4.2 Vulnerability Prediction Models

We study Walden et al.'s [22] vulnerability prediction models, i.e., the one based on software metrics and the one based on text tokens. These models perform file-level vulnerability prediction: each file of the software system under analysis is labelled as *vulnerable* if it contains at least one vulnerability, or as *neutral* if it is known to contain no vulnerabilities at the moment. We have to point out that files are never labelled as *non-vulnerable*, since we cannot be sure that they do not contain any vulnerabilities at all. In fact, they could contain vulnerabilities that have not been discovered yet.

The model based on software metrics works as follows. Each file is represented by a set of numeric values, i.e., the software metrics computed by the authors [22] and provided in the dataset:

- Lines of Code. The number of lines in a source file where PHP tokens occur. Lines not containing PHP tokens, such as blank lines and comments, are excluded from the count.
- Lines of Code (non-HTML). This is the same as Lines of code, except HTML content embedded in PHP files, i.e., content outside of PHP start/end tags, is not considered.

- Number of Functions. The number of function and method definitions in a file.
- Cyclomatic Complexity. The size of a control flow graph after linear chains of nodes are collapsed into one. This metric is computed by adding one to the number of loop and decision statements in the file.
- Maximum Nesting Complexity: The maximum depth to which loops and control structures in the file are nested.
- Halstead's Volume. The volume estimated as the file's vocabulary multiplied by the logarithm of the file length. The vocabulary is given by the sum of the number of unique operators and the number of unique operands. Operators are method names and PHP language operators, operands are parameter and variable names. The file lenght is given by the sum of the total number of operators and the total number of operands.
- Total External Calls. The number of times a statement in the file invokes a function or method defined in a different file.
- Fan-in. The number of other files that contain statements that invoke a function or method defined in the file being measured.
- Fan-out. The number of other files that contain functions or methods invoked by statements of the file being measured.
- Internal Functions or Methods Called. The number of functions or methods defined in the file that are called at least once by a statement of the same file.
- External Functions or Methods Called. The number of functions or methods defined in other files which are called at least once by a statement in the file being measured.
- External Calls to Functions or Methods. The number of files that contains statements calling a particular function or method defined in the file being measured, summed across all functions and methods of the file being measured.

The software metrics of each file constitute the independent variable the classifier takes as input. The dependent variable is the label *vulnerable* or *neutral*. The classifier is able to predict the label of an unseen file based on the software metrics values.

The model based on text tokens employs the *Bag of Words* approach. Each source code file is first *tokenized*, that is, converted into a list of tokens. Tokens represent language keywords, punctuation, and other code constructs, in addition to variable names and literals. The original authors used the PHP built-in `token_get_all` function to collect tokens and made them available in the dataset [21]. All the tokens generated from all the files make up the vocabulary of the application, which is used to create the words frequency vector of each file. The frequency vector of a file stores the number of occurrences of each word of the vocabulary in the considered file. The values in the frequency vector constitute the independent variable for the classifier.

4.3 Experimental Settings

Our work consists in the following phases: first we replicate Walden et al.'s study [22] in order to have a baseline measure of the performance of the two vulnerability prediction models, and then we analyze the performance variation caused by the use of a different validation method. In the first step, we execute a series of experiments to evaluate the performance of the two models by using 3-fold cross-validation, as done by Walden et al. [22]. For each model, i.e., the one based on software metrics and the one based on text tokens, the PHPMyAdmin dataset is split into 3 equally large folds and 3 rounds of validation are executed. The overall performance of each model is computed as the average performance obtained in the 3 rounds. The choice of 3 for the k value is explained by Walden et a.[22] and is driven by the low absolute number of vulnerable samples in the dataset. In the second step, we focus on the release-based validation approach. We perform several experiments by considering one release per time in the validation set. The training set is made up of a certain number of releases which are prior to the one each experiment is focused on. We follow the methodology suggested by Shin et al. [20] [19], by including 3 prior releases in the training set. Thus, we start by using the first three releases for training and the fourth for testing, and we proceed as depicted in Figure 2, resulting in the execution of about 92 experiments for each model. As done by Walden et al. [22], we employ the Random Forest classifier, which has been broadly used by the research community [6]. Since the dataset is highly unbalanced and RQ_3 pushes us to investigate data balancing techniques, we repeat all the experiments 3 times: once we use the dataset as-is, once we perform undersampling on the training data and once we perform oversampling on the training data. Undersampling consists in removing samples from the majority class, i.e., the neutral files subset, in order to match the number of samples in the minority class, i.e., the vulnerable files. We use the random undersampling strategy implementation provided by the Scikit-learn package for Python. Oversampling consists in augmenting the number of samples in the minority class to match the size of the majority class. To perform oversampling we follow Jimenez et al. [10] and use the Chawla et al.'s Synthetic Minority Over-sampling TEchnique (SMOTE) [3], which adds synthesised samples that have similar features as the real instances of the minority class.

5 ANALYSIS OF THE RESULTS

We execute a total of 555 experiments in order to collect information about the performance of the considered software vulnerability prediction models and provide answers to our research questions. In this paper, we discuss the results as summarized by F_1 *score*, but additional performance indicators are publicly available in the replication package.[1] Figure 3 depicts the F_1 score obtained from the execution of the experiments: the dots indicate the average score obtained in 3-fold cross-validation, and the box plots summarize the measures collected in all the rounds of release-based validation. By analyzing the performance plot and reasoning about the motivations of the observed values, we can discuss the answers to our research questions.

5.1 Answer to RQ_1

When considering the dataset as-is, we note that the F_1 score obtained in the release-based validation method is better than the one output by cross-validation. We think that this is due to the

[1]https://github.com/giuliasellitto7/maltesque2021

high imbalance of the dataset and the different sizes of the considered training and test sets. Our highly unbalanced dataset contains limited samples of vulnerable elements and plenty of samples of neutral elements. Hence, it is easy for the model to recognize neutral elements: this leads to a high number of True Negatives and a low number of False Negatives. The scarce presence of vulnerable samples in the dataset causes low amounts of Positives predictions in general. This impacts on the computation of the performance indicator and the overall performance seems quite promising, but, in our opinion, it is biased by the dataset conformation. It is needed to also reason about the size of the training and test sets used to perform cross-validation and release-based validation, respectively. When using cross-validation, the whole dataset is split into 3 folds, of which 2 are used for training and the other one for testing. So the training-test data ratio is 2:1 and the size of the test set is similar to about 30 releases of the application data. When applying the release-based validation approach, 3 releases are used for training and one is used for testing, thus the training-test data ratio is 3:1 and the size of the test set is much lower. We conjecture that there are fewer chances for the model to make mistakes in the release-based validation setting, so the performance resulting from this evaluation method is better. However, the high imbalance of the dataset leads us to come up with poorly meaningful arguments. Thus, we deeperly focus on analyzing the results obtained when using a balanced dataset. When considering the undersampled dataset, we observe a significant change in the F_1 score. We note that the release-based evaluation method outputs very different performance and we think that the cause has to be researched in the size of the training set. In each round of the release-based validation, data relative to three releases of the application are used for training the model, and we remember that the presence of vulnerable samples is rare. When applying undersampling, the number of neutral samples is reduced to match the number of vulnerable ones. Hence, the model ends up with very little data to train and cannot learn how to discriminate among the classes. This causes a severe decline of the performance. The situation seems to get better when considering the oversampled dataset. The SMOTE oversampling technique is used by Jimenez et al. [10] and allows us to augment the number of vulnerable samples by building synthetic data which is similar to the real one. The model has more opportunity to learn in the training phase, since the dataset is larger *and* balanced: this results in amazing performance. The cross-validation outputs better indicators, since the whole dataset is used: we remind that in the release-based validation only the data relative to three releases of the application is included in the training set. As a summary, we report that the performance output by the release-based validation is generally lower than the one obtained with the cross-validation approach. The only observed case in which the release-based validation leads to higher scores than the cross-fold one is the experimentation on the unbalanced dataset, which is not suitable to be applied in a realistic scenario.

5.2 Answer to RQ_2

By looking at the plot in Figure 3, we can observe the cases in which the boxes relative to the release-based validation method are more distant from the dots representing the performance obtained with

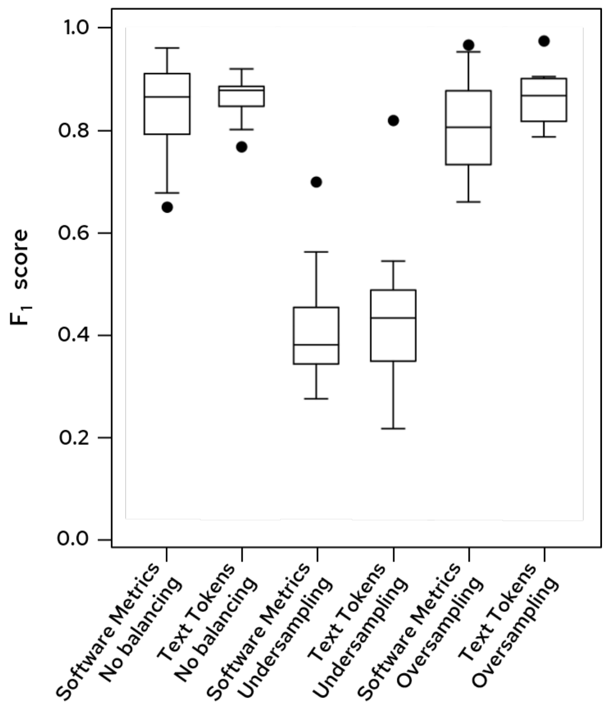

Figure 3: F_1 score of the models.

cross-validation: we can intuitively say that such cases indicate the settings that cause the most difference in precision. We compute the percentage of difference observed in the two approaches: the Software Metrics based model shows an average variation of 23%, and the Text Tokens based one of 19%. We assert that the modelling approach based on Software Metrics is more sensitive to the use of a different validation method.

5.3 Answer to RQ_3

We partially answered to this research question in our previous discussion about RQ_1. When performing no data balancing at all, the performance seems to be good, but it is biased by the high imbalance of the dataset. The undersampling technique drastically reduces the size of the dataset that remains available for training, thus resulting in poor performance. The oversampling method seems to be the one leading to the best results.

6 THREATS TO VALIDITY

A potential threat to the validity of our study is represented by the utilized dataset. We cannot absolutely exclude inaccuracies in the data, i.e., potential imprecisions in the collection on vulnerabilities and their mapping to the affected files. Nevertheless, we are confident about its correctness, since it has been broadly used and validated, in over 90 researches [6] [4] [11] [23] [1]. Another potential problem may be the multi-collinearity of features that we used as predictors. The goal of our research was to investigate the difference that a release-based validation approach induces to the performance of vulnerability prediction models, so we did not focus on improving the feature selection of the existing models. We used the default implementation of the Random Forest classifier provided in the Scikit-learn package for Python, and we did not tune any parameters, because that lay out of our scope. The only factors that we influenced were the validation method and the dataset composition, which represented the main focus of our work, so we can assert that the different results that we obtained are due to these variations. However, we cannot claim that the results that we discussed are generalizable and statistically significant, since we did not perform a sufficiently large number of experiments and we only included a single application in our dataset. This preliminary study dictates the agenda for our next work: in the future, we plan to execute a higher amount of tests on a larger dataset to analyze the statistic relevance of our findings.

7 CONCLUSION

We presented our empirical study, in which we analyzed the impact of a release-based validation approach on the performance of two vulnerability prediction models, formerly evaluated with cross-validation. We conducted our research using the dataset provided in [21] and the models proposed by Walden et al. [22]. We ran a total number of 555 experiments and we examined the performance observed in all the executions. We noted that the release-based validation approach leads to lower performance in general, so there is the need to further work for providing the practitioners machine-learning solutions which are suitable to be employed in a real context. In the immediate future we plan to continue researching in the same direction, by following Jimenez et al.'s study [10] about the importance of accounting for *real-world labelling* of the training data when applying release-based validation methods. The concept of *real-world labelling* is the following: if we consider to train our vulnerability prediction model at time *t*, we must include in the dataset only those vulnerabilities that have already been discovered before *t*. This makes the empirical analysis context almost identical to the real-world scenarios in which the model is supposed to be used. We want to rearrange the dataset that we used in this study, in order to adapt it to this kind of further investigation. A way to do this would consist in tracing back each vulnerability in the dataset to the time it was discovered and reported. After retrieving this information, we could perform a deeper and more realistic analysis on how vulnerability prediction models would perform in a practical context.

REFERENCES

[1] Cagatay Catal, Akhan Akbulut, Ecem Ekenoglu, and Meltem Alemdaroglu. 2017. Development of a Software Vulnerability Prediction Web Service Based on Artificial Neural Networks. 59–67. https://doi.org/10.1007/978-3-319-67274-8_6

[2] Cagatay Catal, Akhan Akbulut, Sašo Karakatič, Miha Pavlinek, and Vili Podgorelec. 2016. Can we predict software vulnerability with deep neural network?

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (Jun 2002), 321–357. https://doi.org/10.1613/jair.953

[4] Xiang Chen, Yingquan Zhao, Zhanqi Cui, Guozhu Meng, Yang Liu, and Zan Wang. 2020. Large-Scale Empirical Studies on Effort-Aware Security Vulnerability Prediction Methods. *IEEE Transactions on Reliability* 69, 1 (2020), 70–87. https://doi.org/10.1109/TR.2019.2924932

[5] Aayush Garg, Renzo Degiovanni, Matthieu Jimenez, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2020. Learning To Predict Vulnerabilities From Vulnerability-Fixes: A Machine Translation Approach. arXiv:2012.11701 [cs.CR]

[6] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Comput. Surv.* 50, 4, Article 56 (Aug. 2017), 36 pages. https://doi.org/10.1145/3092566

[7] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 1322–1328.

[8] Aram Hovsepyan, Riccardo Scandariato, Wouter Joosen, and James Walden. 2012. Software Vulnerability Prediction Using Text Analysis Techniques. In *Proceedings of the 4th International Workshop on Security Measurements and Metrics* (Lund, Sweden) *(MetriSec '12)*. Association for Computing Machinery, New York, NY, USA, 7–10. https://doi.org/10.1145/2372225.2372230

[9] Matthieu Jimenez, Mike Papadakis, and Yves Le Traon. 2016. Vulnerability Prediction Models: A Case Study on the Linux Kernel. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 1–10. https://doi.org/10.1109/SCAM.2016.15

[10] Matthieu Jimenez, Renaud Rwemalika, Mike Papadakis, Federica Sarro, Yves Le Traon, and Mark Harman. 2019. The Importance of Accounting for Real-World Labelling When Predicting Software Vulnerabilities. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 695–705. https://doi.org/10.1145/3338906.3338941

[11] Aydin Kaya, Ali Seydi Keceli, Cagatay Catal, and Bedir Tekinerdogan. 2019. The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models. *Journal of Software: Evolution and Process* 31, 9 (2019), e2164. https://doi.org/10.1002/smr.2164 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2164 e2164 smr.2164.

[12] William Landi. 1992. Undecidability of Static Analysis. *ACM Lett. Program. Lang. Syst.* 1, 4 (Dec. 1992), 323–337. https://doi.org/10.1145/161494.161501

[13] Andrew Meneely and Laurie Williams. 2009. Secure Open Source Collaboration: An Empirical Study of Linus' Law. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '09)*. Association for Computing Machinery, New York, NY, USA, 453–462. https://doi.org/10.1145/1653662.1653717

[14] Andrew Meneely and Laurie Williams. 2010. Strengthening the Empirical Analysis of the Relationship between Linus' Law and Software Security. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (Bolzano-Bozen, Italy) *(ESEM '10)*. Association for Computing Machinery, New York, NY, USA, Article 9, 10 pages. https://doi.org/10.1145/1852786.1852798

[15] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting Vulnerable Software Components. *Proceedings of the ACM Conference on Computer and Communications Security*, 529–540. https://doi.org/10.1145/1315245.1315311

[16] Thomas Reps. 2000. Undecidability of Context-Sensitive Data-Dependence Analysis. *ACM Trans. Program. Lang. Syst.* 22, 1 (Jan. 2000), 162–186. https://doi.org/10.1145/345099.345137

[17] Riccardo Scandariato and James Walden. 2012. Predicting Vulnerable Classes in an Android Application. In *Proceedings of the 4th International Workshop on Security Measurements and Metrics* (Lund, Sweden) *(MetriSec '12)*. Association for Computing Machinery, New York, NY, USA, 11–16. https://doi.org/10.1145/2372225.2372231

[18] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. 2014. Predicting Vulnerable Software Components via Text Mining. *IEEE Transactions on Software Engineering* 40, 10 (2014), 993–1006. https://doi.org/10.1109/TSE.2014.2340398

[19] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A. Osborne. 2011. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Trans. Softw. Eng.* 37, 6 (Nov. 2011), 772–787. https://doi.org/10.1109/TSE.2010.81

[20] Yonghee Shin and Laurie Williams. 2011. Can traditional fault prediction models be used for vulnerability prediction? *Empirical Software Engineering* 18 (02 2011). https://doi.org/10.1007/s10664-011-9190-8

[21] James Walden, Jeff Stuckman, and Riccardo Scandariato. 2014. *PHP Security vulnerability dataset.* https://seam.cs.umd.edu/webvuldata/

[22] James Walden, Jeff Stuckman, and Riccardo Scandariato. 2014. Predicting Vulnerable Components: Software Metrics vs Text Mining. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. 23–33. https://doi.org/10.1109/ISSRE.2014.32

[23] Yun Zhang, David Lo, Xin Xia, Bowen Xu, Jianling Sun, and Shanping Li. 2015. Combining Software Metrics and Text Features for Vulnerable File Prediction. In *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*. 40–49. https://doi.org/10.1109/ICECCS.2015.15