Detecting Overfitting of Machine Learning Techniques for Automatic Vulnerability Detection

Niklas Risse
niklas.risse@mpi-sp.org
Max-Planck-Institute for Security and Privacy
Bochum, Germany

ABSTRACT

Recent results of machine learning for automatic vulnerability detection have been very promising indeed: Given only the source code of a function f, models trained by machine learning techniques can decide if f contains a security flaw with up to 70% accuracy.

But how do we know that these results are general and not specific to the datasets? To study this question, researchers proposed to *amplify* the testing set by injecting semantic preserving changes and found that the model's accuracy significantly drops. In other words, the model uses *some* unrelated features during classification. In order to increase the robustness of the model, researchers proposed to train on amplified training data, and indeed model accuracy increased to previous levels.

In this paper, we replicate and continue this investigation, and provide an actionable model benchmarking methodology to help researchers better evaluate advances in machine learning for vulnerability detection. Specifically, we propose a cross validation algorithm, where a semantic preserving transformation is applied during the amplification of either the training set or the testing set. Using 11 transformations and 3 ML techniques, we find that the improved robustness only applies to the specific transformations used during training data amplification. In other words, the robustified models still rely on unrelated features for predicting the vulnerabilities in the testing data.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Software and its engineering** → **Software testing and debugging**.

KEYWORDS

machine learning, automatic vulnerability detection, semantic preserving transformations, large language models

ACM Reference Format:
Niklas Risse. 2023. Detecting Overfitting of Machine Learning Techniques for Automatic Vulnerability Detection. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3611643.3617845

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3617845

1 INTRODUCTION

Recently a number of different publications have reported high scores on vulnerability detection benchmarks using machine learning (ML) techniques [1, 5–8, 14]. So, does this mean that the problem of detecting security vulnerabilities in software is solved? How do we know that the reported results are general and not specific to the benchmark datasets?

To study these questions, researchers have tried to explore the capabilities and limits of machine learning techniques in ways that go beyond simple evaluations on benchmark testing sets. For example, it is possible to apply small semantic preserving amplifications to the input programs of a state-of-the-art model and then measure, whether the model changes its predictions and whether it still performs well. Examples for such amplifications are identifier renaming [9, 17–20], insertion of unexecuted statements [9, 16, 18, 19] or replacement of code elements with equivalent elements [3, 10]. The impact of applying semantic preserving amplifications to testing data has been explored for many different tasks in software engineering, and the results seems to be clear: Machine learning techniques lack robustness against semantic preserving amplifications [3, 4, 9, 11, 15–20].

A common strategy to address the robustness problem is training data amplification; applying the same or similar amplifications to the training dataset. Many of the works that reported the lack of robustness of ML models when trained on unamplified data also investigated training data amplification using their respective methods [4, 9, 11, 16–20]. They found a restoration or at least improvement towards the initial high performance. But does training data amplification actually improve the ability of these models to detect vulnerabilities, or are they just overfitting to a different set of data?

We contribute to answering this question by proposing a general benchmarking methodology that can be used to evaluate the capabilities of machine learning models for vulnerability detection by using data amplification. The core of the methodology is a cross validation, in which a selected semantic preserving amplification method is applied to the training dataset of a model, and a *different* amplification method is applied to the testing dataset (see Figure 1). When repeated for all possible pairs out of a set of amplification methods, the resulting scores provide a measure of overfitting to the specific semantic preserving amplification methods that were used during training data amplification.

In addition to the general methodology, we present the results of an empirical study, in which we apply the proposed methodology to three state-of-the-art ML techniques for vulnerability detection. We implemented 11 different semantic preserving amplification methods and tried to cover types of amplifications commonly used

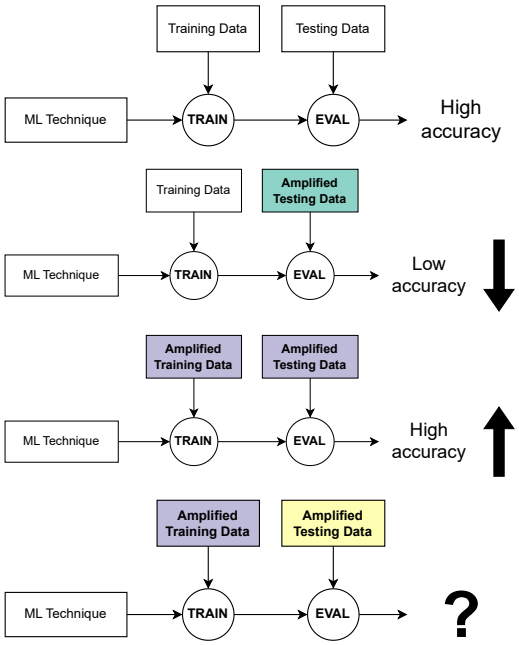

Figure 1: Our proposed methodology to detect overfitting of machine learning techniques for vulnerability detection. Different amplification methods are represented by different colors.

in the literature [9, 10, 13, 16–20]. Table 1 lists all amplification methods, categorizes them by type and provides short descriptions for each of them.

In order to evaluate ML techniques which represent the state-of-the-art of machine learning for vulnerability detection, we chose the Top-3 techniques from the CodeXGLUE leaderboard [12] for which the authors provide open-source implementations. Based on the described criteria, we selected CoTexT [14], VulBERTa [8] and PLBart [2] for our experiments.

As our main data source we use the Devign dataset [21] from the CodeXGLUE benchmark, which contains 26.4k C functions (45.6% contain security vulnerabilities) from the two popular open source repositories Qemu[1] and FFmpeg[2]. Most of the vulnerabilities in the dataset are memory-related, e.g. memory leaks, buffer overflows, memory corruption or crashes.

Figure 2 shows the result of evaluating the three selected techniques using our proposed methodology. As expected, we find a strong benefit of training data amplification (59.8% average restoration of accuracy) when the amplification methods applied to training and testing dataset are the same. However, we find no improvement in performance when the amplification methods applied to training and testing dataset are different. In fact, we even find an additional 35.7% average decrease in accuracy. In other words, state-of-the-art ML techniques severely overfit to the specific label-unrelated features introduced by training data amplification. The improved robustness only applies to the specific type of amplification method used during training.

[1]Qemu: https://github.com/qemu/qemu
[2]FFmpeg: https://github.com/FFmpeg/FFmpeg

Table 1: The semantic preserving amplification methods that we implemented for our experiments.

Identifier	Type	Description
a_1	Identifier Renaming	Rename all function parameters to a random token.
a_2	Statement Reordering	Reorder all function parameters.
a_3	Identifier Renaming	Rename the function.
a_4	Statement Insertion	Insert unexecuted code.
a_5	Statement Insertion	Insert comment.
a_6	Statement Reordering	Move the code of the function into a separate function.
a_7	Statement Insertion	Insert white space.
a_8	Statement Insertion	Define additional void function and call it from the function.
a_9	Statement Removal	Remove all comments.
a_{10}	Statement Insertion	Add code from training set as comment.
a_{11}	All	Random selection of a_1 to a_{10}.

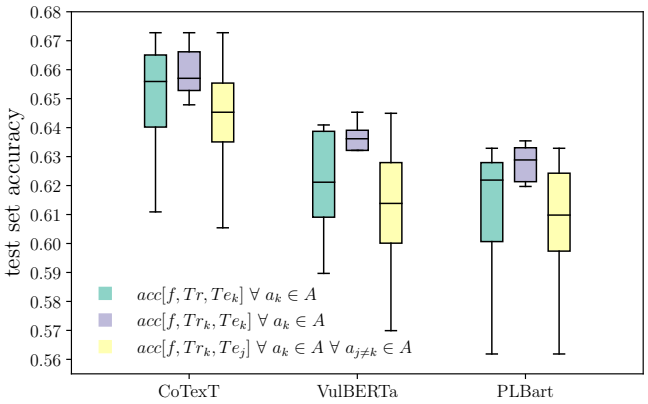

Figure 2: Testing dataset performance of the three ML techniques when only testing data is amplified (green boxplots), when training- and testing data is amplified using the same amplification method (purple boxplots), and when training- and testing data is amplified using *different* amplification methods (yellow boxplots). Each boxplot represents the distribution of the testing dataset accuracies over the 11 amplification methods.

In summary, this paper makes the following contributions:

- ⋆ We present a general methodology that can be used to evaluate ML models for vulnerability detection using data amplification.
- ⋆ We show empirically, that the robustness gained by data amplification only applies to the specific amplification methods used during training, and that robustified models overfit to the unrelated features introduced by semantic preserving amplification methods.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972.

REFERENCES

[1] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 2655–2668. https://www.aclweb.org/anthology/2021.naacl-main.211

[2] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 2655–2668. https://www.aclweb.org/anthology/2021.naacl-main.211

[3] Leonhard Applis, Annibale Panichella, and Arie van Deursen. 2021. Assessing Robustness of ML-Based Program Analysis Tools using Metamorphic Program Transformations. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1377–1381. https://doi.org/10.1109/ASE51524.2021.9678706

[4] Pavol Bielik and Martin Vechev. 2020. Adversarial Robustness for Code. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 84, 12 pages.

[5] Luca Buratti, Saurabh Pujar, Mihaela Bornea, Scott McCarley, Yunhui Zheng, Gaetano Rossiello, Alessandro Morari, Jim Laredo, Veronika Thost, Yufan Zhuang, et al. 2020. Exploring software naturalness through neural language models. *arXiv preprint arXiv:2006.12641* (2020).

[6] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[7] Michael Fu and Chakkrit Tantithamthavorn. 2022. LineVul: A Transformer-based Line-Level Vulnerability Prediction. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. IEEE.

[8] Hazim Hanif and Sergio Maffeis. 2022. VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection. In *2022 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN55064.2022.9892280

[9] Jordan Henkel, Goutham Ramakrishnan, Zi Wang, Aws Albarghouthi, Somesh Jha, and Thomas Reps. 2022. Semantic Robustness of Models of Source Code. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. https://doi.org/10.1109/saner53432.2022.00070

[10] Yaoxian Li, Shiyi Qi, Cuiyun Gao, Yun Peng, David Lo, Zenglin Xu, and Michael R. Lyu. 2022. A Closer Look into Transformer-Based Code Intelligence Through Code Transformation: Challenges and Opportunities. https://doi.org/10.48550/ARXIV.2207.04285

[11] Yiyang Li, Hongqiu Wu, and Hai Zhao. 2022. Semantic-Preserving Adversarial Code Comprehension. In *Proceedings of the 29th International Conference on Computational Linguistics*. International Committee on Computational Linguistics, Gyeongju, Republic of Korea, 3017–3028. https://aclanthology.org/2022.coling-1.267

[12] Microsoft 2021. *CodeXGLUE leaderboards*. Retrieved March 8, 2023 from https://microsoft.github.io/CodeXGLUE/#LB-DefectDetection

[13] Pedro Orvalho, Mikoláš Janota, and Vasco Manquinho. 2022. MultIPAs: applying program transformations to introductory programming assignments for data augmentation. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1657–1661.

[14] Long Phan, Hieu Tran, Daniel Le, Hieu Nguyen, James Annibal, Alec Peltekian, and Yanfang Ye. 2021. CoTexT: Multi-task Learning with Code-Text Transformer. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*. Association for Computational Linguistics, Online, 40–47. https://doi.org/10.18653/v1/2021.nlp4prog-1.5

[15] Md Rafiqul Islam Rabin, Nghi D.Q. Bui, Ke Wang, Yijun Yu, Lingxiao Jiang, and Mohammad Amin Alipour. 2021. On the generalizability of Neural Program Models with respect to semantic-preserving program transformations. *Information and Software Technology* 135 (jul 2021), 106552. https://doi.org/10.1016/j.infsof.2021.106552

[16] Shashank Srikant, Sijia Liu, Tamara Mitrovska, Shiyu Chang, Quanfu Fan, Gaoyuan Zhang, and Una-May O'Reilly. 2021. Generating Adversarial Computer Programs using Optimized Obfuscations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=PH5PH9ZO_4

[17] Zhou Yang, Jieke Shi, Junda He, and David Lo. 2022. Natural Attack for Pre-Trained Models of Code. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1482–1493. https://doi.org/10.1145/3510003.3510146

[18] Noam Yefet, Uri Alon, and Eran Yahav. 2020. Adversarial Examples for Models of Code. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 162 (nov 2020), 30 pages. https://doi.org/10.1145/3428230

[19] Huangzhao Zhang, Zhiyi Fu, Ge Li, Lei Ma, Zhehao Zhao, Hua'an Yang, Yizhe Sun, Yang Liu, and Zhi Jin. 2022. Towards Robustness of Deep Program Processing Models—Detection, Estimation, and Enhancement. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 50 (apr 2022), 40 pages. https://doi.org/10.1145/3511887

[20] Huangzhao Zhang, Zhuo Li, Ge Li, L. Ma, Yang Liu, and Zhi Jin. 2020. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models. In *AAAI Conference on Artificial Intelligence*.

[21] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. *Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks*. Curran Associates Inc., Red Hook, NY, USA.

Received 2023-06-05; accepted 2023-08-11