BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

Filtering False Positives from Static Analysers Using Graph Neural Networks

Bc. Tomáš Beránek*

Abstract

This article discusses the use of graph neural networks to decrease the number of false positives from static analysis, specifically targeting the high rate of false positives produced by the Meta Infer static analyser—over 95%. The proposed solution involves two main pipelines: a Training Pipeline that converts the D2A dataset—a set of labeled Meta Infer's reports—into extended code property graphs on which the models are later trained, and an Inference Pipeline that integrates the trained models into practical software analysis. Preliminary results are encouraging, showing that our GNN model, with an AUROC of 0.84, is competitive with some other models being developed by strong industrial teams. Notably, while other existing solutions in this field are closed source, our GNN model offers a promising open source alternative even in the current stages of architecture selection and hyperparameter tuning. The article highlights the potential of GNNs in effectively ranking false positives, thus enhancing the utility of Meta Infer. It is also planned to open source the entire framework, contributing to broader research and applications in vulnerability detection and false positive reduction tasks.

*xberan46@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

[Motivation] Static analysis is commonly used in software development to detect various kinds of errors and vulnerabilities, leveraging its ability to consider all possible program paths and uncover even rarely manifesting errors missed by tests. However, its major drawback is the high number of false positives (i.e. false alarms). This work focuses on Meta Infer, a static analyzer whose results contain over 95 % false positives [1, 2]. The frequent need to verify these false positives often leads developers to disregard the results of static analysis. The aim of this article is to enhance the utility of Meta Infer by ranking the errors detected by Infer based on their likelihood of being true positive (i.e. not a false alarm).

[Existing solutions] There has already appeared previous attempts to use ML to reduce the number of FPs in Infer. Most notable models are from the creators of the D2A dataset (used in this article), specifically the **C-BERT** [3] and **vote** [2] models. The latest **vote-new** [3] model achieves nearly perfect scores on smaller projects, but this score declines as the number of samples in the project increases. Unfortunately, these models are closed source, and thus it is not possible to verify the results or use the models.

[Proposed solution] The solution proposed in this article is based on deep graph neural networks (GNNs). However, since the D2A dataset is not in a graph format, it was first necessary to create a pipeline for its transformation. Additionally, another pipeline was developed to generate graphs from real C software based on Infer reports, and apply the created model to them. All components will be open sourced.

[Contributions] This article consists of several parts and thus has multiple different outputs. Arguably, the main output is the **Training Pipeline** and the **Graph D2A** it creates—a graph version of the D2A dataset. Another output is the **Inference Pipeline**, which can apply Infer with GNN models to real C software. Both pipelines improve existing methods by generating graphs from the C language using conditional compilation, which is very important for real-world software. Last but not least, the output includes the **trained model** itself. The models are still in the phase of architecture selection and hyperparameter tuning, however, initial experiments are already achieving very promising results with an average **AUROC of 0.84**.

2. System for False Positive Filtering

The proposed system for reducing FPs in Infer utilizes **GNNs**, which achieve **top results in the field of vulnerability detection** [4]. The choice of GNNs is also based on the fact that many **characteristics of source code can be naturally represented using graphs**—for instance, abstract syntax trees or control flow graphs are used in compilers, underscoring the usefulness of these representations. Another advantage of GNNs is that they inherently handle **arbitrary input size**, which is not the case with, e.g., the C-BERT model, leading to some difficulties [3].

The **D2A** dataset [2] contains **reports from 6 real-world software from Infer**. Since D2A was created automatically, the labels **do not have 100 % accuracy**. D2A is in a text format and is thus incompatible with GNNs. Therefore the proposed system includes a Training Pipeline that converts D2A into its graphical counterpart—Graph D2A. To facilitate the deployment of models on real projects, an Inference Pipeline was also created, capable of running Infer on any C project and applying the models to its output.

2.1 Training Pipeline

The input to the Training Pipeline (see Figure 1) is the D2A dataset and the repository on which D2A was created. Using information from D2A (bug trace, compilation commands, etc.) and code from the repository, **LLVM IR**—a language-independent intermediate representation—is generated for each sample. The conversion to LLVM IR has several reasons:

1. the compilation **resolves macros**, which existing approaches ignore [5, 6, 4, 7, 8] because they rely solely on the Joern tool,
2. LLVM IR is a simpler language, therefore the graphs are **less complex**, **though larger**,
3. there are numerous tools available for converting LLVM IR into various graph formats.

The generated LLVM IR is then **sliced** by LLVM Slicer based on slicing criteria extracted from D2A. Subsequently, a code property graph (CPG) is created—a frequently used representation (and its modifications) in vulnerability detection [4, 9, 8]. The binary CPG is enhanced by the Joern tool to include a call graph, data types, etc. For each D2A sample, a CPG with this additional information is generated and stored in a CSV format as Graph D2A. These graphs, however, require data preprocessing according to the chosen ML library. In this work, we chose **TFGNN**. Preprocessing thus involves applying feature engineering and converting the samples into the commonly used **tfrecords** format. **Feature engineering** is a crucial phase of the Training Pipeline—its successful execution can **significantly reduce** the graphs and **eliminate useless and redundant data**, thereby facilitating learning. Conversely, a flawed design can prevent training altogether. Feature engineering transforms raw graphs into a format that we will refer to as **extended code property graphs**.

Table 1. Model comparison based on the AUROC.

	libtiff	nginx	httpd
our GNN	0.84	0.88	0.74
vote[2]	0.89	0.77	0.77
C-BERT[3]	0.94	0.89	0.82
vote-new [3]	0.98	0.93	0.90

2.2 Inference Pipeline

The Inference Pipeline relies primarily on **compiler wrappers** that were designed in the author's bachelor's thesis [1]. These capture compilation commands and use them to run Infer while also generating LLVM IR. The rest of the Inference Pipeline (see Figure 2) is fundamentally the same as the Training Pipeline.

3. Experiments

Our model is still in the early stages of its development, with various architectures being tested and hyperparameters being tuned. However, it is possible to demonstrate the capabilities of the best model so far (see Figure 3). This model has been trained on projects **libtiff**, **httpd**, and **nginx** (part of D2A). Figure 4 shows the ROC curves for the individual projects. Table 1 compares models from [2, 3] with our GNN. It is evident that vote-new and C-BERT perform better across all tested projects. However, our GNN is comparable to the older vote, which indicates that GNNs are a promising approach for reducing FPs. Figure 5 shows the results for the intended use-case of these models—precision for the top N% of reports. E.g., for the **top 5% ranking increased the precision ∼6x**.

4. Conclusions

The experiments show that our model currently only matches the performance of older vote models. However, these are only initial models, which suggests that GNNs are certainly suitable for reducing FPs. Also, existing models are closed source, making even the current GNN model a very good open source alternative. This article presents not only the model but also Graph D2A, which facilitates the training of GNN models for future research in FPs reduction. Additionally, the Inference Pipeline significantly eases the deployment of future models.

5. Acknowledgements

I would like to thank my supervisor prof. Ing. Tomáš Vojnar, Ph.D. for numerous and very helpful pieces of advice.

References

[1] Tomáš Beránek. Practical application of facebook infer on systems code, 2021.

[2] Yunhui Zheng, Saurabh Pujar, Burn Lewis, Luca Buratti, Edward Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. D2a: A dataset built for ai-based vulnerability detection methods using differential analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 111–120. IEEE, 2021.

[3] Saurabh Pujar, Yunhui Zheng, Luca Buratti, Burn Lewis, Yunchung Chen, Jim Laredo, Alessandro Morari, Edward Epstein, Tsungnan Lin, Bo Yang, et al. Analyzing source code vulnerabilities in the d2a dataset with ml ensembles and c-bert. *Empirical Software Engineering*, 29(2):48, 2024.

[4] Sahil Suneja, Yunhui Zheng, Yufan Zhuang, Jim Laredo, and Alessandro Morari. Learning to map source code to software vulnerability using code-as-a-graph. *CoRR*, abs/2006.08614, 2020.

[5] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei, and Bin Li. Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection. *Information and Software Technology*, 136:106576, 2021.

[6] Zhibin Guan, Xiaomeng Wang, Wei Xin, and Jiajie Wang. Code property graph-based vulnerability dataset generation for source code detection. In Guangquan Xu, Kaitai Liang, and Chunhua Su, editors, *Frontiers in Cyber Security: Third International Conference, FCS 2020, Tianjin, China, November 15–17, 2020, Proceedings*, pages 584–591, Singapore, 2020. Springer, Springer Singapore.

[7] Wang Xiaomeng, Zhang Tao, Wu Runpu, Xin Wei, and Hou Changyu. Cpgva: code property graph based vulnerability analysis by deep learning. In *2018 10th International Conference on Advanced Infocomm Technology (ICAIT)*, pages 184–188. IEEE, 2018.

[8] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32, 2019.

[9] Yuelong Wu, Jintian Lu, Yunyi Zhang, and Shuyuan Jin. Vulnerability detection in c/c++ source code with graph representation learning. In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1519–1524. IEEE, 2021.