Ongoing Automated Data Set Generation for Vulnerability Prediction from Github Data

Torge Hinrichs[1]

Abstract: This paper describes the development of a continuous github repository analysis pipeline with the focus on creating a data set for vulnerability prediction in source code. Currently, used data sets consist only of source code functions or methods without additional meta information. This paper assumes that the surrounding code of vulnerable functions can be beneficial to the detection rate. In order to test this assumption, large data sets are needed that can be created using the proposed pipeline. Although the pipeline requires some improvements, in a first test run 1.5 million repositories could be analyzed and evaluated. The resulting data set will be published in the future.

Keywords: Vulnerability Prediction; Vulnerability Detection; Machine Learning; data set generation

1 Introduction

Detecting vulnerabilities in source code has been one of the most active fields in security research during the last years. Due to the latest advances in machine learning and data science more and more researchers are using these techniques for vulnerability detection / prediction. All approaches have one thing in common. They require large data sets of non- and vulnerable source code to train their models. Finding source code repositories for security analysis, analyzing and labeling the data is a tedious task that can not be performed by hand at a large scale. Even though some projects like "Project KB"[Po19] tried to craft such a data set manually, they "only" managed to analyze 205 distinct open source-projects leading to a remarkable 1282 commit entries of data. However, the amount of data gathered is minimal in comparison of other traditional machine learning approaches which use millions of data samples to train their networks. Automatically gathered data sets like "Draper Muse Data set"[Ru18] often consists of individual source code methods or functions with a label indicating the type of vulnerability or a binary classification without additional meta information. Even though this approach is beneficial for training machine learning models with remarkable results, it limits the scope of detection since the surrounding of the function can not be taken into consideration. In addition, creating data sets for vulnerability detection suffers from the "needle in the haystack" problem[SW13]. Vulnerabilities are less frequent and hard to detect, therefore the data sets contain large quantities of non-vulnerable in comparison to vulnerable data (scarce data)[BS19], which influences the overall performance of machine learning approaches. To tackle the lack of data sets for source code analysis with machine learning that also take the surrounding of a finding into consideration, this paper proposes an ongoing data analysis of public github.com

[1] HAW Hamburg, Berliner Tor 5, 20099 Hamburg, Germany torge.hinrichs@haw-hamburg.de

doi:10.18420/sicherheit2022_17

repositories to automatically identify and classify vulnerabilities with static analysis, saving the vulnerable lines and the commit id. This reduces the “needle in the haystack” problem in the long run and enables context analysis for future work. The following section provides a brief overview of the approach and methods used, followed by first analysis results and finally a discussion with future work.

2 Approach and Methods

The hypothesis of this contribution is that a static analysis pipeline is capable of performing all previously mentioned steps autonomously and can be scaled to a larger computing setup if needed. The following figure 1 shows a brief overview of the implemented components. Each block shown in the figure is a separate component that can be scaled if needed to

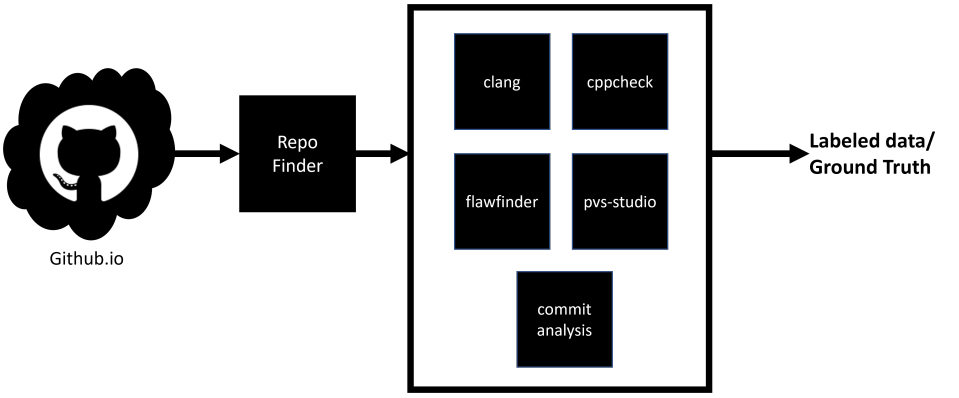

Fig. 1: Overview: Static analysis pipeline

prevent bottlenecks. Creating this approach can be split in 4 main tasks: selecting and providing the repository, analyzing with static analysis and labeling the findings. For the first task, a github interface was created that is capable of searching the repository stream provided by their API. By default, the stream is ordered by the repository id that increases for each repository created on the platform. The github interface also selects repositories that match the following constrains: The repository consists mainly of C/C++ code, due to the fact that most source code vulnerabilities occur in this language according a report by white source[Wh21]. In addition, the repository shall have a minimum size of at least 100 lines of C/C++ code. This assumes that repositories smaller than this do not contain enough complexity for context analysis. This assumption is based on manual reviews of small github repositories mostly containing snippets or small examples not suitable for machine learning. Repositories matching these criteria will automatically be cloned to a storage server and scheduled for analysis and a meta entry is created in a mongoDB database. During the analysis task, the pipeline automatically schedules an analysis block for each repository. The analysis consists of four state-of-the-art static analysis tools: clang, cppcheck, flawfinder, and pvs-studio. The selection is based on the open availability of these tools. They are also widely used in the creation of other data sets like Draper Muse. In

addition, a commit text analysis was developed that scans commit messages in order to find security relevant keywords based on the BSI-glossary [Bu21]. The last step in the pipeline is to map the reports of the different analysis tools to a common label that is associated with the corresponding Common Weakness Enumeration (CWE). This is achieved by a look up table for each tool.

3 Results

The pipeline was deployed and ran for three months as a first test. During this period 1.5 million repositories where analyzed with an overall yield of 2.3% which results in 35.000 eligible repositories with at least one hit in the analysis tools. The performance of the repository selection was constrained to about 2000 repositories per hour. This is caused by API restrictions on github. The overall performance of each analysis block, however, was measured and is shown in figure 2.

Tool	Avg. Evaluation time [s / Repository]
clang	30
cppcheck	70
flawfinder	10
pvs-studio	30
commit message	60

Fig. 2: Analysis time comparison

The deviation in analysis effort is caused by the varying feature set of the tools used. Each tool was configured to use all available checks. Therefore, cppcheck had to perform the most amount of checks and this, in conclusion, results in the longest analysis time per repository. In addition, the execution time of the individual analysis had large variations. This time is not only determined by the files or lines of code in a repository, but also by the amount of commits. For example the Linux kernel with its over one million commits took more than six hours to scan on the hardware used.

However, evaluating the performance of the commit text analysis showed some issues of this approach. The selection of keywords only based on the BSI-Glossary led to a high number of false positives as keywords like “stackoverflow” are often used in URL to the equal named developer help website “stackoverflow.com”. Overall, the execution of this analysis was rather slow in comparison to its low benefit.

The labeling method, in comparison, has no mentionable issues. It was created independently based on the documentation of each individual tool used. The label for each sample is determined by the analysis result. If one tools detects a possible vulnerability the sample is labeled accordingly, also the number of tools that back this label is stored.

The resulting data samples are stored in a mongoDB database. In hindsight, this has to be changed to a more storage friendly solution. A No-SQL database worked well to store

various different shaped data fields and allows for querying the database, but takes up a lot of disk space. This can easily be improved by using an optimized or compressed storage solution like “hdf5”. This looses the ability to query, but not necessary in a machine learning application anyway.

4 Discussion

In this paper, a concept for creating an automated data set generation for vulnerability data from github repositories was described. This concept was implemented and a first test phase over a three-month period was performed. During that time 1.5 million repositories were analyzed. An evaluation of the pipeline showed that some analysis blocks are more effective than others compared to their throughput and overall effectiveness. In addition, the implementation showed that the concept can be used to generate security relevant data samples continuously. However, this pipeline can still be improved in future work. First, the pipeline should to be set up production ready so an ongoing analysis can be performed. In addition, the overall performance of the analysis should be increased by analyzing the individual blocks and optimize the configuration of each tool. Next, up-scaling the pipeline to a larger infrastructure would also be beneficial to increase the throughput. Finally, storage should be more suitable for machine learning applications. The benefits of having a query-able database is not needed for learning, instead the handling of large data sets in less disk space should be focused on. This can be achieved by transforming a database into another format like “.hdf5”. After the improvements of the pipeline, the final data set will be released to the public.

Bibliography

[BS19] Babbar, Rohit; Schölkopf, Bernhard: Data scarcity, robustness and extreme multi-label classification. 108(8):1329–1351, 2019.

[Bu21] Bundesamt für Sicherheit in der Informationstechnik: , Glossar der Cyber-Sicherheit, 2021. access 2021-11-12, https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Glossar-der-Cyber-Sicherheit/Functions/glossar.html.

[Po19] Ponta, Serena E.; Plate, Henrik; Sabetta, Antonino; Bezzi, Michele; Dangremont, C´edric: A Manually-Curated Dataset of Fixes to Vulnerabilities of Open-Source Software. In: Proceedings of the 16th International Conference on Mining Software Repositories. May 2019.

[Ru18] Russell, Rebecca L.; Kim, Louis; Hamilton, Lei H.; Lazovich, Tomo; Harer, Jacob A.; Ozdemir, Onur; Ellingwood, Paul M.; McConley, Marc W.: , Automated Vulnerability Detection in Source Code Using Deep Representation Learning, 2018.

[SW13] Shin, Yonghee; Williams, Laurie: Can traditional fault prediction models be used for vulnerability prediction? 18(1):25–59, 2013.

[Wh21] WhiteSource: The State of Open Source Vulnerabilities 2021. Technical report, WhiteSource, 2021.