Evaluating and Improving Commit-based Static Analysis - Summary

Jan Philipp Wagner

Technische Universität Darmstadt

Abstract

Static analysis of software deals with the task of finding vulnerabilities in source code without actually executing it. In the recent past, progress has been made in this field by analyzing *git commits* instead of specific files. Here, the state of the art uses machine learning- and text mining techniques to predict whether a specific commit is likely to make the software vulnerable based on its text features and metadata. These tools, although promising, have not seen widespread adoption. Reasons for this might be that they are unavailable to the masses and have not been extensively validated. Therefore, this thesis has the goal of creating such a tool that has been validated and tested on ground truth data and is available to everyone. This is especially challenging because ground truth data for commits that introduced a vulnerability are extremely rare. Our approach to this problem is scouring the web for datasets that have been constructed manually, as well as automatically tracing vulnerabilities back to their origin. We find that previous methods for tracing vulnerabilities automatically only have a 39% accuracy instead of the 96.9% conjectured in previous works (VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits *[CCS'15]*). We are able to make minor improvements to raise the accuracy by 1.4%. Further, we find that our replication of the state of the art classifier is able to identify 27% of Vulnerability-Contributing Commits (VCCs) made to the Linux kernel repository in 2016 and 2017 with a 1.4% precision. We make improvements to this model to achieve the same recall with 2.2% precision, meaning that only 62% of the manual effort is required to identify the same amount of vulnerabilities.

1 Introduction

Static analysis of software deals with the task of finding vulnerabilities in source code without actually executing it. A fairly new approach to this field of research originated in the 2015 paper *VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits* [4]. VCCFinder uses a linear Support Vector Machine model that takes into account GitHub metadata

and combines it with code-metric data to identify *git commits* that are likely to introduce a vulnerability. The idea is that commit metadata can give loads of interesting insight into the nature of a patch such as the experience of the contributing developer or the frequency of how often the code changes.
Despite significantly lowering the false positive rate of conventional static analysis tools like Flawfinder [2] on the dataset created by Perl et al., VCCFinder (as well as similar tools [7]) has not seen widespread adoption in any active open source projects to the best of our knowledge. In addition to the VCCFinder GitHub repository[1] being incomplete, a reason for this might be the limited dataset the tool was tested on, as it contained non ground truth data. Therefore, we want to evaluate a replication of their model on a ground truth dataset we created ourselves. This is challenging, as past attempts to recreate VCCFinder failed due to the inability of building a sufficiently sized dataset for training [5]. We approach this problem by combining manually produced datasets from security researchers [3], [6] with data we gained from automatically tracing vulnerabilities back to their origin. In order to accomplish the latter, we utilize an improved version of the *git blame* based heuristic that was also used in the creation of the dataset VCCFinder was trained on.
Lastly, we contribute a model that is closely based on VCCFinder but improves its accuracy through the addition of new and more expressive metadata features and adjustments to the bag-of-words model that VCCFinder uses to represent code. We make the model as well as our dataset publicly available for the community to use[2].

2 Heuristic

The heuristic used by Perl et al. to create their dataset of VCCs works by first finding a large amount of fixing commits (e.g. commits that fixed a vulnerability) and then utilizing the *git blame* command to automatically map these fixing commits to VCCs (Figure 1). *Git blame* finds which commit last changed a specific line in a given file and so it can be used for tracing a vulnerability back to its origin. In their paper, Perl et al. claim that this heuristic has a 96.9%

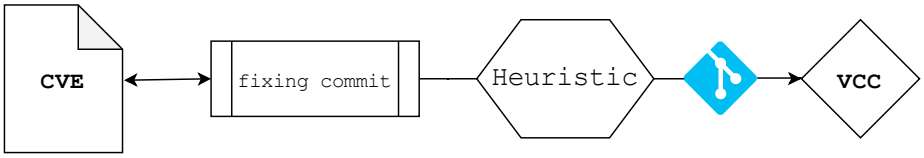

Figure 1: Overview of git blame based heuristic

accuracy. We evaluate this claim and find that the accuracy is closer to 40%. The evaluation is performed by comparing the VCCs found by the heuristic to

[1] https://github.com/hperl/vccfinder
[2] https://github.com/jp-wagner/vccrosshair

ground truth data we garnered from the Ubuntu CVE Tracker [6] as well as the Vulnerability History Project[3].
Our efforts to improve the heuristic are only moderately successful but with some alterations, such as using the -w flag when blaming and weighting specific blames heavier than others, we raise the accuracy from 39.3% to 40.7%. Additionally, we are able to identify a subset of vulnerabilities where the accuracy is 63%. We make use of this knowledge by assigning a greater weight to the VCCs we are more confident in in the training process of our classifier. Using our improved heuristic on the fixing commits garnered by Manuel Brack for his thesis [1], we are able to create a dataset of 4873 VCCs spanning 10 repositories. To the best of our knowledge, this is the largest dataset of its kind.

3 Classifier

Since the VCCFinder repository is incomplete, we have to replicate the tool to the best of our abilities. Known differences to the original include only taking into account *git* metadata instead of additional GitHub metadata (star count, fork count, etc.) and potentially minor differences in the preprocessing of the data points. Despite these differences we expect similar results.
For our own classifier we add new metadata features, such as how many different files a commit altered and how often the author of a commit worked on the same files before, as well as use better delimiters for the tokenization of the source code of a commit.
In the evaluation of the two classifiers we aim to replicate a real life scenario by testing them on every commit made to the Linux kernel in 2016 and 2017. This set consists of 127,602 unclassified commits and 129 VCCs. We specifically choose the Linux kernel because this is the repository we have the most ground truth data for [6]. However, we would like to highlight that we do not know how many unknown/not CVE worthy vulnerabilities are contained in this set. Thus, our true positives must be considered a lower bound and the false positives an upper bound.
When matching the true positive rate of the mature open-source static analysis tool Flawfinder, we find that our classifier reduces its false positive rate by 83% (Table 1). This means that it would only take a sixth of the manual effort to identify the same amount of vulnerabilities. Furthermore, the manual effort required by our replication of VCCFinder is reduced by 38%.

[3] `https://github.com/VulnerabilityHistoryProject`

	True Positives	False Positives	False Negatives	True Negatives
Flawfinder	35	9138	94	118,436
VCCFinder	35	2501	94	125,073
Our model	35	1533	94	126,041

Table 1: Comparison of the classifiers with Flawfinder.

Figure 2 displays the precision and recall graph for VCCFinder and our classifier on the test set. As depicted in the graph, our classifier has a closer point to the optimum (upper right corner) as well as a larger area under the curve than our replication of VCCFinder (VCCFinder: 0.011, ours: 0.018). This indicates that our classifier is an overall improvement over VCCFinder.

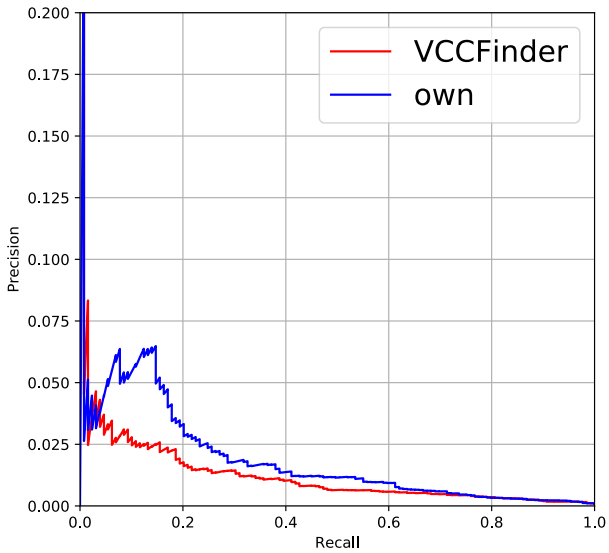

Figure 2: Precision and recall graph of VCCFinder and our classifier.

4 Conclusion

In this thesis, we showed that the heuristic for mapping a fixing commit to its VCC used in previous works is not as accurate as was previously assumed. Furthermore, we introduced a heuristic of our own that shows slightly better results with the added benefit of a confidence measurement that indicates when

the commit blamed by the heuristic is more likely to be correct. Combining these automatically traced VCCs with VCCs that were manually found by security researchers, we created a large scale dataset that we make available to the community. Finally, we used this dataset to train a classifier that can to a degree successfully distinguish between commits that are prone to introduce a vulnerability and commits that are not. When comparing our classifier to other commit-based static analysis tools, we evaluated that it greatly outperforms the mature open-source tool Flawfinder and also significantly raises the performance of our replication of the classifier it was closely based on.

5 Disclaimer

The exact results described in this summary can be found in a revised version of the thesis. Due to a maintenance of the server used for our evaluation that was unannounced to the author (force majeure), the thesis that was officially handed in contains slightly different results. These results were drawn from an outdated version of our dataset. For the sake of submitting the official document, what follows is the original version of the thesis. You can find the revised version in the GitHub repository[4].

References

[1] Manuel Brack. *A large-scale statistical Analysis of Vulnerability Liftetimes in Open-Source Software*. `https://fileserver.tk.informatik.tu-darmstadt.de/NA/Thesis_MB.pdf`.

[2] *D. A. Wheeler. Flawfinder*. `https://dwheeler.com/flawfinder/`. Accessed: 2020-03-18.

[3] Andrew Meneely et al. "An empirical investigation of socio-technical code review metrics and security vulnerabilities". In: *Proceedings of the 6th International Workshop on Social Software Engineering*. 2014, pp. 37–44.

[4] Henning Perl et al. "Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 426–437.

[5] Fraser Price. *Analyzing Vulnerabilities in the Chromium Browser*. `https://www.doc.ic.ac.uk/~livshits/papers/theses/fraser_price.pdf`.

[6] *Ubuntu CVE Tracker*. `https://git.launchpad.net/ubuntu-cve-tracker`. Accessed: 2020-03-18.

[7] Limin Yang, Xiangxue Li, and Yu Yu. "VulDigger: A Just-in-Time and Cost-Aware Tool for Digging Vulnerability-Contributing Changes". In: *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE. 2017, pp. 1–7.

[4] `https://github.com/jp-wagner/vccrosshair`