Using Machine Learning to Find Security Vulnerabilities in Software

Emil Karlsson emilk2@kth.se
Maximilian Georg Kurzawski mgku@kth.se

KTH Royal Institute of Technology

2023-04-28

SCA	Static Code Analysis
SDLC	Software Development Life Cycle
CWE	Common Weakness Enumeration, list of software and hardware weakness types
CTF	Capture the Flag, computer security competition
NLP	Natural Language Processing

Abstract

Security is essential and unavoidable in the field of software engineering. In particular, the existence of software vulnerabilities have lead to numerous cyber-attacks, evidenced for example by the 2014 Heartbleed bug [18, 14]. The process of minimizing the attack surface and mitigating damage of such an attack include identification, assessment and a remediation process [14]. However, traditional methods and tools used for identifying and assessing software vulnerabilities could overlook some attack vectors, such as phishing and credential issues. This essay aims to present the concept of a vulnerability detection framework powered by machine learning, and how it improves on conventional methods.

Introduction

As software increase in size and complexity, finding vulnerabilities has become a necessity. In a paper Grieco G. et al. [11] describes how large software systems, such as an operating system, could ship with thousands of binary executables, and that in many cases, a comprehensive set of tools for proper testing are missing. Thus, the paper describes how in practice, it may not feasible to eliminate all bugs in a software system, but rather to identify and then classify the importance of software bugs that could lead to software vulnerability. However, due to the nature of a software vulnerabilities, they are inherently hard to find. As described in the paper, a primary example of the root cause of a vulnerability could be a infinite loop. In order to grasp the role of machine learning in this issue, some core concepts in traditional vulnerability detection methods will be presented.

Static code analysis is a common traditional tool that scans for, among other features, vulnerabilities. For many software systems, it is part of their software development life cycle (SDLC). Static code analysis is meant to highlight possible vulnerabilities without executing any of the source code [5].

Dynamic code analysis and hybrid code analysis integrates execution of the program in the analysis. It aims to mimic a malicious user and simulate attacks in order to find software vulnerabilities [12].

Heartbleed is a major and well-known example of a software vulnerability that allowed stealing information normally protected by SSL/TLS [18]. It will be used as a real-life example of when analysis tools failed to identify a crucial software vulnerability.

This bug was caused by improper input validation of request messages causing a `buffer over-read`, meaning information is read outside the intended memory location. OpenSSL static code analysis tools were unable to detect the vulnerability due to the vast scale and complexity of the system [19, 2]. Wheeler D. even argues in a article [19] that current standard analysis approaches might not be able to detect a vulnerability that is based on the same bug as Heartbleed was. Wheeler further mentions that a dynamic code analysis would not detect Heartbleed either, as it was not designed to detect buffer over-reads.

Machine learning

Machine learning based frameworks, contrary to conventional methods, employs strategies such as *pattern detection* and *inductive reasoning* [8]. Utilizing such frameworks enables the detection of features that could potentially be evaded by a human, or intentionally be disregarded as uninformative [13]. Additionally, according to Garcia J. when provided with the right data set, machine learning aids in seeings the big picture of software, which enables a machine learning powered tool to make larger connections between various components in the software system [7].

In the RSA Virtual 2022 conference, IBM [17] presents a flip-side to the benefits of using a machine learning powered vulnerability management tool. It is argued that, however useful machine learning can be for vulnerability management, the tools themselves can become a target for new types of attacks. The attacks presented are divided in four groups:

- Evasion: modify input to influence a model
- Poisoning: add a backdoor to the training data
- Extraction: steal the machine learning model
- Inference: learn about private data within the model

An attack on the tools themselves implies an attack vector on its own, and according to Poremba S [17], the advantages and disadvantages should be balanced. The benefits of machine learning power tools includes relieving manual tasks and more accurate vulnerability detection, but comes with isolation responsibility to reduce their attack surfaces.

Finally, Durelli V. et al. describe another perspective how one could use machine learning to evaluate the effectiveness of already defined software tests [6]. This means that machine learning could be used in a rather patching fashion that enables it to be adopted even in older code bases. However, as pointed out in the paper, this idea has solely been seen as an extension, and that tests still need to be written using traditional methods, since a machine learning algorithm may not be able to identify the entire test-case evaluation process.

State of the Art

GitHub

In the beginning of 2022, GitHub upgraded their *code scanning* by incorporating machine learning. Prior to the upgrade, the code scanning was entirely based on *CodeQL* [8], a code analysis engine developed by themselves [10]. This enabled GitHub to detect and fix vulnerabilities using a deep logical analysis and deductive reasoning, based on knowledge by security experts [8]. The added machine learning component relies primarily on CodeQL outputs. GitHub's machine learning algorithms analyzes and learns from the extracted features [8], see figure 1.

As of writing this essay, GitHub's machine learning powered code scanning feature supports the programming languages JavaScript and TypeScript, and covers many of the well-known vulnerability types [9]:

- Cross-site scripting (XSS, CWE-79)
- Path injection (CWE-22, CWE-23, CWE-36, CWE-73, CWE-99)
- NoSQL injection (CWE-943)
- SQL injection (CWE-89)

GitHub summarizes every vulnerability found for the developer to overview, and potentially take action. The list can be found under *Security/Code scanning alerts*, and every additional vulnerability found by the machine learning algorithms are marked with the tag *Experimental*. This highlights an important distinction; while GitHub does scan the code using machine learning based tools, it does not take any action on the code itself.

ChatGPT

A different, arguably less technical approach to find software vulnerabilities is ChatGPT. As of writing this essay, ChatGPT is capable of simulating realistic malicious attacks from both users and computer systems. Additionally, ChatGPT can be used to analyze large data set, reports, and logs, which implies it is capable of identifying some vulnerabilities in software. Furthermore, as ChatGPT uses natural language processing (NLP), it takes upon a different role than previously mentioned tools. Rather than inspecting code triggered by a script in a DevOps pipeline, it can operate along side a developer or DevOps engineer [4]. In a blog post Choudhary V., a penetration tester at NCR Corporation [3], describes how he used ChatGPT to debug CTF challenges (a type of software vulnerability code challenge) and how ChatGPT could successfully identity buffer overflow vulnerabilities. The following is an output from Choudhary's experiments, highlighting some of ChatGPT's core points to find vulnerabilities:

```
The function ‘process_bin_update‘ in the
source code appears to have a buffer
overflow vulnerablity. The ‘vlen‘ variable
is calcualcated as the difference between
...
which could cause a buffer overflow if it
is used as a size of length without proper
checking
```

ChatGPT's ability to assist a developer or DevOps engineer in software vulnerability testing makes it a valuable tool for researchers and professionals. Rather than writing code, keywords or other specific meta data, natural language queries can be made with sophisticated description in human readable format. It can make the vulnerability detection process more intuitive and user friendly. It has been found vastly more efficient than letting the developer or DevOps engineer analyze the code themselves [1].

On the other hand, according to Cyfirma [4], a cybersecurity company specializing in vulnerability intelligence, ChatGPT may not identify or accurately classify a newly discovered software vulnerabilities, which might lead to false positives and false negatives. As they argue; ChatGPT is only as good as the data is is trained on.

Another approach to use ChatGPT in the context of software vulnerability management is to ask it to write the initial code itself. Nair M. et al. evaluated how ChatGPT could be used to generate executable code for 10 noteworthy CWE's [16].

Discussions

Static code analysis and machine learning-based methods are not necessarily mutually exclusive, as has already been demonstrated by tools like GitHub's scanning feature. Machine learning-based code analysis also has the benefit of being more adaptable and dynamic than conventional rule-based methods. A machine learning model can learn from prior experiences and modify its analysis in response to new data, as opposed to depending on a fixed set of predetermined rules and checks. This implies that the model can change and improve its ability to detect new sorts of vulnerabilities as they are uncovered. Using machine learning for code analysis is not without its difficulties, of course. Finding high-quality training data that effectively represents the kinds of vulnerabilities that the model is intended to detect is one of the main hurdles. The developers and security specialists who are in charge of training the model must exert a lot of time and skill to do this.

The potential for false positives and false negatives is yet another issue with machine learning-based code analysis. Even if a piece of code is not dangerous, the model may identify it as a potential vulnerability since it is looking for patterns and anomalies in the code. On the other hand, it might overlook some vulnerabilities that are subtler or more complicated. To balance sensitivity and specificity, the model needs to be carefully calibrated and tuned.

The possibility of adversarial attacks with machine learning-based code analysis is another potential issue. An attacker might be able to alter the code in such a way as to avoid detection by the model if they can determine how the model generates its predictions. This emphasizes the significance of using a combination of machine learning-based technologies and other security procedures, such as manual code review and penetration testing.

Reflection

Heartbleed was a complicated and multifaceted problem that was difficult to identify and, using standard approaches, impossible to detect. While it is hard to determine whether a machine learning framework could have spotted the vulnerability before it was exploited, there are some signs that it may have been a beneficial tool in this situation. For example, a study published in the IEEE Xplore Digital Library [19] found that a machine learning algorithm was able to detect vulnerabilities in OpenSSL with high accuracy, including some that were missed by traditional static code analysis. However, it is important to note that machine learning is not a silver bullet solution for vulnerability detection, and there are challenges and limitations associated with this approach. Ultimately, a comprehensive and multi-faceted approach that incorporates multiple tools and strategies is likely the most effective way to prevent vulnerabilities like Heartbleed from occurring in the future.

There is additional advantage to employing language models for security research, such as ChatGPT. Many persons working in the field of software engineering have never received sufficient training in developing safe software, let alone in identifying software vulnerabilities. ChatGPT may be used to give advise on how to guard against specific sorts of assaults. Aaron Mulgrew's recent blog article investigated the usage of ChatGPT to develop dangerous malware [15]. The usage of chatGPT to create malicious software was addressed in a recent blog article by Aaron Mulgrew. ChatGPT refuses to comply due to the fact that it was not designed to do so. Nevertheless, there were some important takeaways and even guidance, by ChatGPT on how to protect against particular threats. This makes it possible for developers who have not received the necessary training to do so to get guidance on how to construct safe software and spot flaws.

Conclusion

While GitHub's and other static code analysis tools, as well as ChatGPT are excellent resources for locating software vulnerabilities, static code analysis will always be constrained by the scope of the vulnerabilities it is designed to find. The use of ChatGPT and other large language models, on the other hand, can be utilized as a coding helper and may aid in the discovery of software flaws. But as was previously mentioned, the effectiveness of a language model is always correlated with the quality of the training data.

Key takeaway

Machine learning is an effective tool to help find software vulnerabilities, but it cannot replace the requirement for a skilled expert to validate the results and possibly suggest whole new patches.

References

[1] Mohammad Aljanabi et al. "ChatGpt: Open Possibilities". In: *Iraqi Journal For Computer Science and Mathematics* 4.1 (2023), pp. 62–64.

[2] Marco Carvalho et al. "Heartbleed 101". In: *IEEE Security & Privacy* 12.4 (2014), pp. 63–67. DOI: 10.1109/MSP.2014.66.

[3] Vivek Choudhary. *CHATGPT: Enhancing code security and detect vulnerabilities.* URL: https://www.linkedin.com/pulse/chatgpt-enhancing-code-security-detect-vivek-choudhary/.

[4] CYFIRMA CYFIRMA. *CHATGPT AI in Security Testing: Opportunities and challenges.* URL: https://www.cyfirma.com/outofband/chatgpt-ai-in-security-testing-opportunities-and-challenges/.

[5] Ryan Dewhurst. *Static code analysis.* URL: https://owasp.org/www-community/controls/Static_Code_Analysis.

[6] Vinicius H. S. Durelli et al. "Machine Learning Applied to Software Testing: A Systematic Mapping Study". In: *IEEE Transactions on Reliability* 68.3 (2019), pp. 1189–1212. DOI: 10.1109/TR.2019.2892517.

[7] Jess Garcia. *Me, My Adversary; AI: Investigating; hunting with machine learning.* URL: https://www.rsaconference.com/library/presentation/usa/2021/me-my-adversary-ai-investigating-hunting-with-machine-learning.

[8] Tiferet Gazit. *Leveraging machine learning to find security vulnerabilities.* Feb. 2022. URL: https://github.blog/2022-02-17-leveraging-machine-learning-find-security-vulnerabilities/.

[9] Tiferet Gazit and Alona Hlobina. *Code scanning finds more vulnerabilities using machine learning.* Feb. 2022. URL: https://github.blog/2022-02-17-code-scanning-finds-vulnerabilities-using-machine-learning/.

[10] github codeql github. URL: https://codeql.github.com/.

[11] Gustavo Grieco et al. "Toward Large-Scale Vulnerability Discovery Using Machine Learning". In: *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy.* CODASPY '16. New Orleans, Louisiana, USA: Association for Computing Machinery, 2016, pp. 85–96. ISBN: 9781450339353. DOI: 10.1145/2857705.2857720. URL: https://doi.org/10.1145/2857705.2857720.

[12] Jeremiah Grossman. *Static and Dynamic Analysis for Web Applications - OWASP.* URL: https://owasp.org/www-pdf-archive/Atlanta_March_2010_Presentation.pdf.

[13] Erik Kaminski. *Is AI-based vulnerability management really that efficient?* Aug. 2021. URL: https://aithority.com/machine-learning/is-ai-based-vulnerability-management-really-that-efficient/.

[14] Triet H. M. Le, Huaming Chen, and M. Ali Babar. "A Survey on Data-Driven Software Vulnerability Assessment and Prioritization". In: *ACM Comput. Surv.* 55.5 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3529757. URL: https://doi.org/10.1145/3529757.

[15] Aaron Mulgrew. *I built a Zero Day virus with undetectable exfiltration using only ChatGPT prompts.* https://www.forcepoint.com/blog/x-labs/zero-day-exfiltration-using-chatgpt-prompts. 2023. URL: https://www.forcepoint.com/blog/x-labs/zero-day-exfiltration-using-chatgpt-prompts.

[16] Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. *Generating Secure Hardware using ChatGPT Resistant to CWEs.* Cryptology ePrint Archive, Paper 2023/212. https://eprint.iacr.org/2023/212. 2023. URL: https://eprint.iacr.org/2023/212.

[17] Sue Poremba. *Experts weigh in: Pros and cons of Machine Learning for AI security.* Apr. 2023. URL: https://securityintelligence.com/articles/ml-ai-security-pros-cons/.

[18] Inc. http://www.synopsys.com/ Synopsys. *The heartbleed bug.* URL: https://heartbleed.com/.

[19] David A. Wheeler. "Preventing Heartbleed". In: *Computer* 47.8 (2014), pp. 80–83. DOI: 10.1109/MC.2014.217.

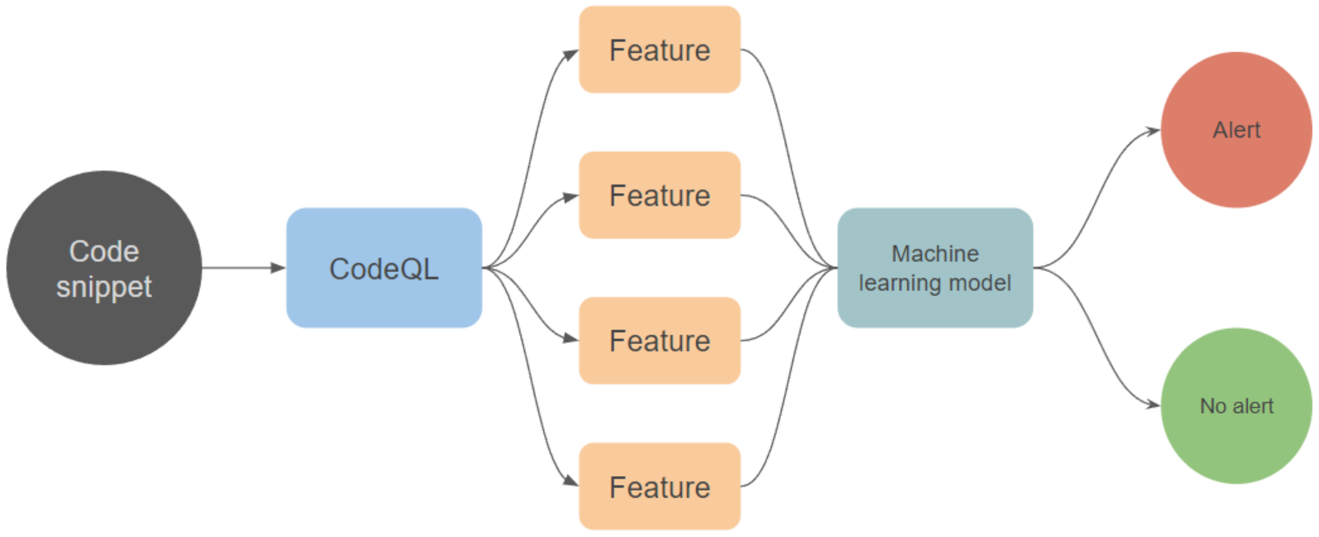

Figure 1: GitHub Code Scanning