대규모 언어모델을 활용한 코드 취약점 리페어

한우림 [1], 유미선 [1], 백윤흥 [1]
[1] 서울대학교 전기정보공학부, 서울대학교 반도체 공동연구소

wrhan@sor.snu.ac.kr, msyu@sor.snu.ac.kr, ypaek@sor.snu.ac.kr

A Study on Code Vulnerability Repair via Large Language Models

Woorim Han[1], Miseon Yu[1], Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

Abstract

Software vulnerabilities represent security weaknesses in software systems that attackers exploit for malicious purposes, resulting in potential system compromise and data breaches. Despite the increasing prevalence of these vulnerabilities, manual repair efforts by security analysts remain time-consuming. The emergence of deep learning technologies has provided promising opportunities for automating software vulnerability repairs, but existing AI-based approaches still face challenges in effectively handling complex vulnerabilities. This paper explores the potential of large language models (LLMs) in addressing these limitations, examining their performance in code vulnerability repair tasks. It introduces the latest research on utilizing LLMs to enhance the efficiency and accuracy of fixing security bugs.

1. Introduction

Software vulnerabilities refer to security flaws, glitches, or weaknesses within software systems that attackers can exploit for malicious purposes. Specifically, attackers may exploit unaddressed security vulnerabilities in software to launch attacks and compromise a system to cause system damage or steal confidential data, leading to significant economic damage. Security vulnerabilities are becoming more widespread in contemporary software, with significant consequences for our society. Common Vulnerabilities and Exposures [1] reported a record-breaking discovery of 26,448 software vulnerabilities in 2022, marking a notable 59% increase from 2021 [2]. Thus, it is crucial to defend against software vulnerabilities through code repair. Unfortunately, it requires security analysts to invest substantial effort in manually addressing or repairing these vulnerable functions.

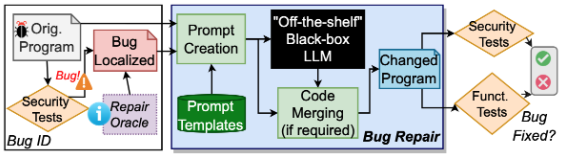
The advancements in deep learning have opened avenues for automatic approaches to software vulnerability repair, enabling efficient learning of the transition from vulnerable code to corrected code. Although researchers have proposed various Artificial Intelligence based approaches to assist under-resourced security analysts in better understanding the characteristics of vulnerabilities and finding them more quickly, these approaches still suffer from inaccurate repair results and struggle handling lengthy vulnerable code.

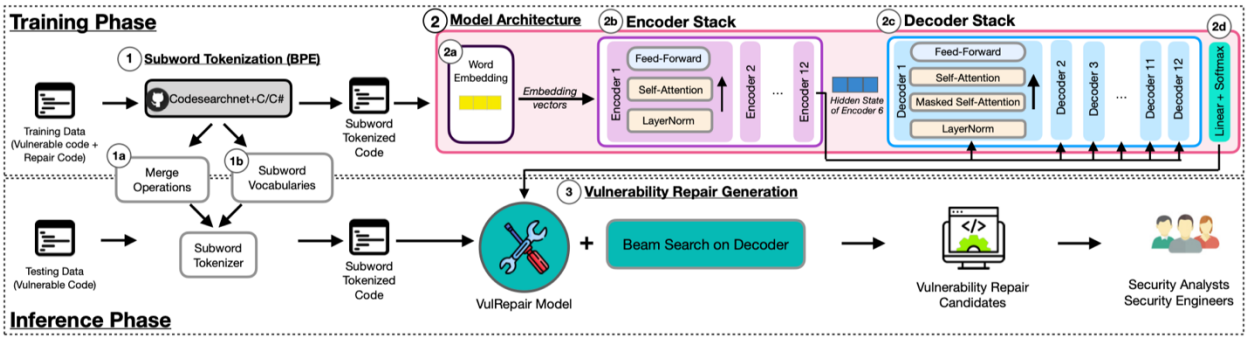
Recently, with large language models (LLMs) gaining popularity in various code intelligence tasks, there has been an increasing focus on using LLMs for code vulnerability repairs, aiming to address the limitations of previous AI-based approaches. Various recent works examine the repair capability of LLMs and utilize them to achieve state-of-the-art performance in learning-based code vulnerability repair.

2. Large Language Model

LLMs have gained widespread adoption in Natural Language Processing (NLP) owing to their remarkable performance, typically following the Transformer architecture and undergoing training on extensive datasets via self-supervised learning. Recent LLMs like ChatGPT and GPT-4 have demonstrated impressive performance in various code intelligence tasks. Thus, considerable efforts have been made towards leveraging the capability of LLMs for code generation or repair tasks.

(fig 1) The Testing Framework of [3]





(fig 2) An overview of VulRepair [4]

3. Code Vulnerability Repair via Large Language Models

A code vulnerability repair task involves identifying and fixing security vulnerabilities present in software code. These vulnerabilities can range from simple issues like improper input validation to more complex problems such as buffer overflows or injection attacks. The vulnerability repair task involves analyzing the codebase, identifying vulnerable areas, and applying appropriate fixes to mitigate security risks. Recent works typically utilize large language models for applying correct patches for identified security bugs.

3.1 Zero-shot Evaluation of Utilizing LLMs for Vulnerability Repair

Pearce et al. [3] provide the first zero-shot evaluation of LLMs for generating security fixes, demonstrating that pre-trained models can produce security fixes without additional training in simple vulnerable scenarios. This work focuses on understanding the repair ability of LLMs. As depicted in Fig. 1, the testing framework utilizes existing security tools to detect bugs within programs. These bug reports and the original programs are converted into prompts for producing potential fixes. Subsequently, the replacement code generated by the LLM is evaluated using external tools to determine if the suggestions effectively repair the original vulnerable program. The authors analyze and compare various prompts, contextual cues, and model parameters (i.e. temperature, sampling strategy) to utilize LLMs for generating repaired versions of insecure programs. Their main focus lies in engineering prompts, as the output of LLMs is highly sensitive to how prompts are provided; one word difference in the prompt could lead to substantial performance drop. They examine five different prompt templates to get the best out of models. Five reasonable templates with varying amount of context were tested. These templates range from providing no context to the LLM to offering extensive comments and hints, which may include subtle variations in wording and word order (such as 'fixed' vs. 'bugfix'), as well as the option to include or exclude the faulty code. Among various LLMs, OpenAI Codex models consistently outperformed others in generating secure and functionally correct patches. Moreover, LLMs tend to perform better with more context provided in the prompt. However, while black-box LLMs often produced correct outputs for simple vulnerable scenarios with CWEs, their performance in repairing real-world scenarios was insufficient compared to state-of-the-art methods that do not utilize LLMs.

3.2 Fine-tuning LLMs for Vulnerability Repair

Fu et al. [4] propose VulRepair, a T5-based automated vulnerability repair approach that addresses various limitations of the prior transformer-based approach, VRepair [5]. The primary constraints of VRepair included its limited capacity to learn the relative position information of code tokens within the input sequences and inability to generate novel tokens that were not present in a vulnerable function but are introduced in the vulnerability repair.

As illustrated if fig. 2, VulRepair consists of training and inference phase. In step 1, subword tokenization is performed using the Byte-Pairs Encoding (BPE) resolving the issue of generating new tokens. In step 2, VulRepair model is built based on the T5 architecture. It generates embedding vectors for each token in the subword-tokenized function and combines them into a matrix (2a). Here, to encode the positional information of each code token within the function, VulRepair utilizes relative position embedding to overcome the limitation of VRepair. Next, the matrix is passes through the T5 encoder stack (2b) and the output of the last encoder fed into each decoder (2c). Lastly, the output of the decoder stack goes through a linear layer with softmax activation to produce the vocabulary's probability distribution (2d). During the training phase, the T5-based VulRepair model is fine-tuned using the vulnerability repairs dataset (i.e. CVE-Fixes [6] and Big-Vul [7]). The fine-tuned model is used for generating software vulnerability repairs during the inference phase.

For evaluation, VulRepair was compared to VRepair with the evaluation metric of %Perfect Prediction. VulRepair achieved a Perfect Prediction of 44%, 21% more accurate than the VRepair. VulRepair demonstrates the ability to repair 745 out of 1,706 widely recognized real-world vulnerabilities.

3.3 Utilizing Multiple LLMs for Better Vulnerability Repair

Zhou et al. [8] introduce VulMaster, a CodeT5-based automatic vulnerability repair method designed to handle the entire vulnerable code, regardless of its length. VulMaster has three key components. The first component is its' Fusion-in-Decoder (FiD) framework to overcome the input length constraint of Transformer-based models. Next, to capture the structural characteristics of the vulnerable code, VulMaster incorporates the Abstract Syntax Tree (AST) as part of its input. Lastly, it extensively leverages expert knowledge from the CWE website to further enhance its repair capabilities. To utilize the expert knowledge of the CWE system, VulMaster provide a novel CWE knowledge Extraction method. In CWE web pages, besides CWE names, there are vulnerable code examples for each CWE. However, the knowledge of "how to fix" the security bugs are in natural language texts rather than source code. Here, ChatGPT is utilized to generate various correct fixes for the vulnerable code examples. This information, the potential fixes generated by ChatGPT, is also used as the input of the codeT5-based repair model to provide detailed guidance. For evaluation, VulMaster was compared with VulRepair [4] and VRepair [5]. Extensive evaluation demonstrates the effectiveness of VulMaster.

4. Conclusion and Future Work

Several approaches for utilizing the large language models in code vulnerability repair were introduced. Although the SOTA repair method, VulMaster, outperforms all the previous approaches, VRepair and VulRepair, the perfect prediction score remains low. Various tuning methods for LLMs can be applied to further enhance the bug fixing ability.

Moreover, most vulnerability repair tasks are evaluated using the PP(Perfect Prediction) metric and the BLEU score which measure the similarity between the generated patch and the ground truth. However, BLEU scores do not validate the correctness of the patches and PP scores disregard the possibility of other valid outcomes. Thus, future work should consider evaluating generated patches with plausible metrics (e.g. unit tests).

ACKNOWLEDGEMENT

This work was supported by the BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2024 and was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2023-00277326). Also, this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program to nurture the best talents (IITP-2023-RS-2023-00256081) grant funded by the Korea government(MSIT) and was supported by Inter-University Semiconductor Research Center (ISRC).

References

[1] CVE Community. 2023. Official website of Common Vulnerabilities and Expo-sures. https://www.cve.org/.

[2] ED TARGETT. 2022. We analysed 90,000+ software vulnerabilities: Here's what we learned. https://www.thestack.technology/analysis-of-cves-in-2022-software-vulnerabilities-cwes-most-dangerous/.

[3] H. Pearce, B. Tan, B. Ahmad, R. Karri and B. Dolan-Gavitt, "Examining Zero-Shot Vulnerability Repair with Large Language Models," in 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2023 pp. 2339-2356.

[4] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. VulRepair: a T5-based automated software vulnerability repair. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 935–947.

[5] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2022. Neural transfer learning for repairing security vulnerabilities in c code. IEEE Transactions on Software Engineering 49, 1 (2022), 147–165.

[6] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering. 30-39.

[7] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. AC/C++ code vulnerability dataset with code changes and CVE summaries. In Proceedings of the 17th International Conference on Mining Software Repositories. 508-512.

[8] Xin Zhou, Kisub Kim, Bowen Xu, DongGyun Han, and David Lo. Out of Sight, Out of Mind: Better Automatic Vulnerability Repair by Broadening Input Ranges and Sources. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE). IEEE Computer Society, 872–872.