Enhancing OSS Remediation with Patch Backporting

Lyuye Zhang
College of Computing and Data Science
Nanyang Technological University, Singapore
zh0004ye@e.ntu.edu.sg

***Abstract*—Nowadays existing vulnerability remediation methods mainly rely on version upgrades which often struggle to eliminate all vulnerabilities in complex scenarios. Therefore, this paper proposes a patch-porting-based tool aiming to automate the porting of patches to vulnerable versions, achieving remediation beyond traditional upgrading. The experiment indicates that 85% of the vulnerabilities successfully generated patches for 20 CVEs, and 70% passed validation, demonstrating that PPR can serve as a supplement to existing vulnerability remediation tools.**

***Index Terms*—Software Security**

I. Introduction

With the widespread adoption of open-source software, the issue of latent security vulnerabilities has garnered increasing attention. To ensure the security of third-party libraries (TPLs), Software Composition Analysis (SCA) is widely employed to identify TPLs and related security vulnerabilities to mitigate security risks. Academic tools like Coral [1], [2], Steady [3], OSSFP [4], [5] and Sembid [6], along with industrial tools such as Snyk [7], Dependabot [8], and OSV Fix [9] adjust dependency trees in software projects to remediate vulnerabilities.

However, relying solely on adjusting software dependency configurations often fails to eliminate vulnerabilities completely. This is primarily due to the continuous disclosure of new vulnerabilities at the latest versions and constrained ranges of version selection. These challenges highlight the need for a more comprehensive and unified remediation approach beyond upgrading. To address this, we propose an enhanced solution using patch porting to remediate remaining vulnerabilities. Our approach, PPR (Patch Porting for Remediation), builds on an existing tool by identifying unfixed vulnerabilities and porting patches to affected TPLs, facing challenges: (1) Versions with vulnerabilities may differ from patches, requiring the precise location of vulnerable code. (2) Many patches involve multi-spot diffs across various files and functions, demanding syntactic and semantic coherence across these locations.

To address these issues, a patch porting tool leveraging large language models (LLMs) aided by coherent semantic analysis is proposed to enhance existing remediation strategies. PPR employs an adaptive strategy by determining the scenarios where the patches must be adapted.

II. Methodology

Recent research by Yang et al. [10] systematically categorizes patch porting scenarios into four:

- **Scenario 1**: The context of the vulnerable file and the patch is consistent.
- **Scenario 2**: Only the patch insertion location differs, including file and line number.
- **Scenario 3**: In addition to location differences, namespaces also vary, such as variable names, function names, or spacing.
- **Scenario 4**: Both logic and code structure differ.

To handle the scenarios, an adaptive mechanism to first automatically identify the specific scenario, and then apply a corresponding remediation strategy. For Scenario 1, Git Apply [11] is used directly. For Scenario 2, only the porting location is required to be identified. Since Scenarios 3 and 4 both require extensive code adaptation and semantic understanding, we utilize an LLM (GPT-4 [12]) to handle the code adaptation for both.

- **Hunk Preprocessing**: Sometimes, multiple hunks in a patch need to work together, such as when a new variable is introduced from another file or library, requiring an *import* statement at the file's beginning. To identify related hunks, Program Dependence Graph (PDG) is used to analyze the relationships among hunks. Given the lack of specialized tools for analyzing Java source code, we selected Antlr [13] as the foundational tool to build PDG based on the Abstract Syntax Tree derived. If connected by PDG, these hunks likely need to be ported together to maintain semantic consistency. Since these connections can be transitive, all directly or indirectly connected hunks should be clustered together, with different clusters ported in parallel.
- **Locating the Hunk Porting Position**: To accurately locate where to port the patch, PPR follows these steps: First, PPR models the pre-patch code by constructing a PDG, especially focusing on code segments before the hunk. The PDG captures control flow and data flow to represent the code's semantics. Using graph isomorphism algorithms to identify positions with similar semantics to the pre-patch code, these structurally similar code segments are candidate locations for the patch port.
- **Patch Generation**: After determining the porting location, PPR utilizes an LLM to generate the patch code. PPR pro-

TABLE I: PPR Backportin Results

#CVE	#Generated	#Successful	%Generated	%Successful
20	17	14	85%	70%

vides LLM with the confirmed patch location and relevant hunk details, instructing it to create a patch for the new version of the code that maintains semantic accuracy and matches the style of the new codebase.

- **Patch Validation**: Patched code often relies on definitions in other files or modules, such as classes and global variables. PPR analyzes these dependencies in the patched code to ensure they remain available or adapts to new dependency structures as needed.

III. Evaluation

As shown in Table I, our experiment involved a total of 20 CVEs, with a successful patch generation rate of 85% and a successful manual verification rate of 70%. The reasons behind failures were analyzed: ① Program Analysis Accuracy Issues: The analysis is limited to one file, with interprocedural analysis confined to functions involved in the patch, omitting dependencies on other related functions. ② Porting Location Selection Issues: the erroreous porting location caused by multiple similar locations led to failed backporting. This analysis indicates that improving interprocedural analysis capabilities and enhancing context information acquisition could help increase patch generation accuracy.

IV. Related Work

Recent advancements in patch porting include TSBPORT [10], PPatHF [14], and FixMorph [15].Both TSBPORT and FixMorph utilize program analysis to identify target locations for patch application. PPatHF employs a fine-tuned LLM to generate ported code. However, these approaches rely on code similarity to locate segments for porting, rather than delving into code semantics. There exists other studies [16], [17], [18] leveraging LLM for vulnerability fixing and analysis. Due to different scenarios and target programming languages, these tools were not included in the evaluation.

V. Conclusion

This paper presents PPR, a patch porting tool based on LLM, which effectively addresses open-source vulnerabilities that cannot be remediated by version adjustment tools. Experimental results has demonstrated its feasibility and effectiveness. Future work will focus on enhancing its inter-procedural analysis capabilities.

References

[1] L. Zhang, C. Liu, Z. Xu, S. Chen, L. Fan, L. Zhao, J. Wu, and Y. Liu, "Compatible remediation on vulnerabilities from third-party libraries for java projects," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 2540–2552. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00212

[2] L. Zhang, C. Liu, S. Chen, Z. Xu, L. Fan, L. Zhao, Y. Zhang, and Y. Liu, "Mitigating persistence of open-source vulnerabilities in maven ecosystem," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 191–203.

[3] "Eclipse Steady," https://projects.eclipse.org/proposals/eclipse-steady, 2023.

[4] J. Wu, Z. Xu, W. Tang, L. Zhang, Y. Wu, C. Liu, K. Sun, L. Zhao, and Y. Liu, "Ossfp: Precise and scalable c/c++ third-party library detection using fingerprinting functions," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 270–282.

[5] L. Zhao, S. Chen, Z. Xu, C. Liu, L. Zhang, J. Wu, J. Sun, and Y. Liu, "Software composition analysis for vulnerability detection: An empirical study on Java projects," in *Proceedings of the 2023 31th acm sigsoft international symposium on foundations of software engineering*, 2023.

[6] L. Zhang, C. Liu, Z. Xu, S. Chen, L. Fan, B. Chen, and Y. Liu, "Has my release disobeyed semantic versioning? static detection based on semantic differencing," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3551349.3556956

[7] "Snyk," https://snyk.io/, 2023.

[8] "Dependabot," https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates, 2023.

[9] "Osv-scanner fix," https://google.github.io/osv-scanner/experimental/guided-remediation/, 2024.

[10] S. Yang, Y. Xiao, Z. Xu, C. Sun, C. Ji, and Y. Zhang, "Enhancing oss patch backporting with semantics," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2366–2380.

[11] "Git Apply," https://git-scm.com/docs/git-apply , 2024.

[12] OpenAI, "Chatgpt," https://openai.com, 2021, gPT-4 model.

[13] "Antlr," https://www.antlr.org/ , 2024.

[14] S. Pan, Y. Wang, Z. Liu, X. Hu, X. Xia, and S. Li, "Automating zero-shot patch porting for hard forks," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 363–375.

[15] R. Shariffdeen, X. Gao, G. J. Duck, S. H. Tan, J. Lawall, and A. Roychoudhury, "Automated patch backporting in linux (experience paper)," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 633–645.

[16] Y. Sun, D. Wu, Y. Xue, H. Liu, W. Ma, L. Zhang, Y. Liu, and Y. Li, "Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning," *arXiv preprint arXiv:2401.16185*, 2024.

[17] L. Zhang, K. Li, K. Sun, D. Wu, Y. Liu, H. Tian, and Y. Liu, "Acfix: Guiding llms with mined common rbac practices for context-aware repair of access control vulnerabilities in smart contracts," *arXiv preprint arXiv:2403.06838*, 2024.

[18] J. Hu, L. Zhang, C. Liu, S. Yang, S. Huang, and Y. Liu, "Empirical analysis of vulnerabilities life cycle in golang ecosystem," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.