USENIX Security '25 Artifact Appendix: Logs In, Patches Out: Automated Vulnerability Repair via Tree-of-Thought LLM Analysis

Youngjoon Kim
Korea University

Sunguk Shin
Korea University

Hyoungshick Kim*
Sungkyunkwan University

Jiwon Yoon*
Korea University

A Artifact Appendix

A.1 Abstract

Paper: Research on automated vulnerability repair often requires extensive program analysis and expert input, making it challenging to deploy in practice. We propose SAN2PATCH, a system that generates patches using only sanitizer logs and source code, eliminating the need for costly program analysis or manual intervention. SAN2PATCH employs multi-stage reasoning with Large Language Models (LLMs) to decompose the patching process into four distinct tasks: vulnerability comprehension, fault localization, fix strategy formulation, and patch generation. Through tree-structured prompting and rigorous validation, SAN2PATCH can generate diverse, functionally-correct patches. Evaluations on the VulnLoc dataset show that SAN2PATCH successfully patches 79.5% of vulnerabilities, surpassing state-of-the-art tools like ExtractFix (43%) and VulnFix (51%) by significant margins. On our newly curated SAN2VULN dataset of 27 new vulnerabilities from various open-source projects, SAN2PATCH achieves a 63% success rate, demonstrating its effectiveness on modern security flaws. Notably, SAN2PATCH excels at patching complex memory-related vulnerabilities, successfully fixing 81.8% of buffer overflows while preserving program functionality. This high performance, combined with minimal deployment requirements and elimination of manual steps, makes SAN2PATCH a practical solution for real-world vulnerability remediation.

Artifact: This artifact includes not only the source code of SAN2PATCH proposed in the original paper, but also the VulnLoc benchmark with added functional tests and our newly constructed SAN2VULN benchmark. All artifacts are dockerized for easy setup, and we provide bash scripts to reproduce all experiments presented in Section 4 of the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

SAN2PATCH is an automated program repair tool that generates patches for vulnerabilities. As a result, the tool itself does not pose any security, privacy, or ethical concerns. To prevent any issues that may arise during the reproduction process, we provide a Docker environment for additional isolation.

A.2.2 How to access

Our artifact is permanently accessible through a DOI link[1], GitHub repositories[2,3], and pre-built Docker images available on Docker Hub[4,5]. Any download method is acceptable; however, we recommend downloading the Docker images directly from Docker Hub for the most seamless setup experience.

A.2.3 Hardware dependencies

There are no specific constraints as long as the environment supports Python 3.10 or Docker. However, we recommend a CPU with at least 4 cores, 16 GB or more of RAM, and at least 100 GB of available disk space for reliable performance. Note that the hardware used for the experiments in the original paper comprised an Intel Xeon Gold 5218 CPU with 32 cores (2.30 GHz) and 64 GB of RAM.

A.2.4 Software dependencies

If the artifact is executed directly on the host system, a Python 3.10 environment and Poetry version 2.1.1 or higher for Python environment management are required. When using the Docker image, Docker version 28 or higher is recommended.

A.2.5 Benchmarks

This artifact provides the benchmark used in the original paper as `san2patch-benchmark`. The benchmark consists of an extended version of VulnLoc with added functional tests and our newly constructed SAN2VULN benchmark containing recent vulnerabilities. To avoid dependency conflicts, the benchmark must be executed in an independent Docker container. When the `san2patch` Docker image is used, Docker-in-Docker must be employed to run the `san2patch-benchmark` Docker image within the `san2patch` container.

*Corresponding authors

[1] https://doi.org/10.5281/zenodo.15614211
[2] https://github.com/acorn421/san2patch
[3] https://github.com/acorn421/san2patch-benchmark
[4] https://hub.docker.com/r/acorn421/san2patch/
[5] https://hub.docker.com/r/acorn421/san2patch-benchmark/

A.3 Set-up

This section assumes the use of our pre-built `san2patch` Docker image. For instructions on running the artifact in a host environment, please refer to the `README.md` file within the `san2patch-main` repository.

A.3.1 Installation

We recommend using the provided Docker images for easier environment setup. Pulling and running the `san2patch` Docker image automatically handles all necessary configurations, including launching the required `san2patch-benchmark` Docker image.

```
docker pull acorn421/san2patch
docker run -it --privileged --name san2patch acorn421/san2patch
```

Running the image automatically starts the Docker server and pulls the `san2patch-benchmark` Docker image. Pulling the image may require a noticeable amount of time (approximately 30 minutes). Once the image has been successfully launched, set the LLM API keys directly in `/app/.env` as shown below. For more information on configuring the environment, refer to the `README.md` in `san2patch-main`.

```
# Required: LLM API Keys (at least one)
OPENAI_API_KEY=your_openai_api_key_here
ANTHROPIC_API_KEY=your_anthropic_api_key_here
GOOGLE_API_KEY=your_google_api_key_here
```

A.3.2 Basic Test

```
python ./run.py Final run-patch --vuln-ids "CVE-2016-1839" --
      model "gpt-4o" --experiment-name "test"
```

To verify whether the test ran successfully, check the following directory:

```
cd ./benchmarks/final/final-test/gen_diff_test
```

A.4 Evaluation workflow

After launching the Docker image and completing the setup, a bash shell will be available. In this shell, scripts prepared in the `/app/experiments` directory can be executed to reproduce the results of each RQ in the Evaluation section of the original paper.

A.4.1 Major Claims

(C1): SAN2PATCH *achieves a 79.5% patch success rate (31/39 vulnerabilities) on the VulnLoc benchmark, significantly outperforming existing approaches. This is demonstrated by the evaluation in Section 4.1.*

(C2): SAN2PATCH *effectively patches recent and unseen vulnerabilities in the* SAN2VULN *benchmark at low cost ($0.48 per attempt) and within practical timeframes (under 9 minutes per patch), as shown by the results in Section 4.2.*

(C3): *The performance of* SAN2PATCH *improves with more advanced LLMs, as its effectiveness relies on LLM capabilities. This is demonstrated in Section 4.3.*

(C4): *The ToT-based patch generation methods, No Context and* SAN2PATCH*, yield a higher proportion of functionally correct patches, as reported in Section 4.4.*

(C5): SAN2PATCH *effectively patches various types of vulnerabilities, as demonstrated by the experiments in Section 4.5.*

(C6): *An ablation study shows that removing stages such as Vulnerability Comprehension and How-To-Fix from* SAN2PATCH *reduces performance, indicating that all stages are essential. This is proven by the experiment in Appendix A.*

A.4.2 Experiments

How to Run Each experiment can be executed by running the corresponding script file. If execution does not proceed as expected, ensure that the `san2patch-benchmark` Docker container is running properly inside the `san2patch` Docker-in-Docker container.

Results The results of each experiment are saved in:

```
/app/benchmarks/final/final-test/gen_diff_{experiment_name}
```

where `experiment_name` is a unique identifier for each experiment, as specified in the script files.

Key result files can be found under each ID of the vulnerability directory:

- `res.txt` – Summary of all attempts, including status codes.
- `*_success.diff` – The final patch that passed all automated tests (vulnerability and functionality tests)
- `*_success.artifact` – All intermediate information generated during the successful patching process.

To check the automated validation results of the experiment, refer to `res.txt`. To simply count the number of successful patches, use the following command:

```
cd /app/benchmarks/final/final-test/gen_diff_{experiment_name}
ls ./**/res.txt | xargs -I {} bash -c "echo '===== {} ====';
     cat {}; echo;" | grep success | wc -l
```

To conduct manual validation, examine each `*_success.diff` file associated with vulnerabilities

that were successful in automated validation. Detailed criteria for manual validation can be found in the supplementary material[6].

Please refer to the `README.md` in `san2patch-main` for further details on additional result files.

(E1): *VulnLoc Benchmark Evaluation [4 scripts]:*
This experiment evaluates the vulnerability patching capability of SAN2PATCH on the VulnLoc benchmark, supporting claims **(C1)**, **(C4)**, and **(C5)**.
Estimated Time: Each script is executed with 4-core parallel processing for 6 hours, for a total of 6×4 hours.
Estimated Cost: Less than $50 per script with GPT-4o, requiring less than $\$50 \times 4$ in total.
Execution: Run the provided scripts for each method as follows:

```
bash ./experiments/rq1/tot_vulnloc.sh
bash ./experiments/rq1/cot_vulnloc.sh
bash ./experiments/rq1/no_context_vulnloc.sh
bash ./experiments/rq1/zeroshot_vulnloc.sh
```

Results: The results are stored in the corresponding output directories:

```
ls ./benchmarks/final/final-test/
     gen_diff_usenix_tot_vulnloc
ls ./benchmarks/final/final-test/
     gen_diff_usenix_cot_vulnloc
ls ./benchmarks/final/final-test/
     gen_diff_usenix_no_context_vulnloc
ls ./benchmarks/final/final-test/
     gen_diff_usenix_zeroshot_vulnloc
```

(E2): SAN2VULN *Benchmark Evaluation [1 script]:*
This experiment evaluates the vulnerability patching capability of SAN2PATCH on the SAN2VULN benchmark, supporting claim **(C2)**.
Estimated Time: The script is executed with 4-core parallel processing for 4 hours, for a total of 4 hours.
Estimated Cost: Less than $40 in total with GPT-4o.
Execution: Run the provided script as follows:

```
bash ./experiments/rq2/usenix_tot_san2vuln.sh
```

Results: The results are stored in the following output directory:

```
ls ./benchmarks/final/final-test/usenix_tot_san2vuln
```

(E3): *LLM Model Comparison [5 scripts]:*
This experiment compares the vulnerability patching capability of SAN2PATCH using different LLM models, supporting claim **(C3)**.
Estimated Time: Each script is executed with 4-core parallel processing for 6 hours, for a total of 6×5 hours per script.
Estimated Cost: Less than $50 per script with GPT-4o, requiring less than $\$50 \times 5$ in total.
Execution: Run the provided scripts for each model as follows:

```
bash ./experiments/rq3/gpt_4o_mini.sh
bash ./experiments/rq3/gpt_35.sh
bash ./experiments/rq3/sonnet_35.sh
bash ./experiments/rq3/gemini_15_pro.sh
bash ./experiments/rq3/gemini_15_flash.sh
```

Results: The results are stored in the corresponding output directories:

```
ls ./benchmarks/final/final-test/
     gen_diff_usenix_gpt_4o_mini
ls ./benchmarks/final/final-test/gen_diff_usenix_gpt_35
ls ./benchmarks/final/final-test/gen_diff_usenix_sonnet_35
ls ./benchmarks/final/final-test/
     gen_diff_usenix_gemini_15_pro
ls ./benchmarks/final/final-test/
     gen_diff_usenix_gemini_15_flash
```

(E4): *Ablation Study [2 scripts]:*
This experiment investigates the impact of removing certain reasoning stages from SAN2PATCH, supporting claim **(C6)**.
Estimated Time: Each script is executed with 4-core parallel processing for 6 hours, for a total of 2×6 hours.
Estimated Cost: Less than $50 per script with GPT-4o, requiring less than $\$50 \times 2$ in total.
Execution: Run the provided scripts as follows:

```
bash ./experiments/ablation/no_comprehend_k5.sh
bash ./experiments/ablation/no_howtofix_k5.sh
```

Results: The results are stored in the following output directories:

```
ls ./benchmarks/final/final-test/
     gen_diff_usenix_no_comprehend
ls ./benchmarks/final/final-test/
     gen_diff_usenix_no_howtofix
```

A.5 Notes on Reusability

Occasionally, the Python script may not terminate automatically even after all experiments have finished. This does not indicate a failure; the experiments are still considered successfully completed.

When running experiments in parallel, it is possible to exceed the API rate limit. It is recommended to monitor the logs to ensure that rate limiting does not occur during execution.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.

[6] https://doi.org/10.5281/zenodo.15654492