USENIX Security '25 Artifact Appendix: COLLISIONREPAIR:First-Aid and Automated Patching for Storage Collision Vulnerabilities in Smart Contracts

Yu Pan[*†], Wanjing Han[*†], Yue Duan[‡], Mu Zhang[†]
[†]University of Utah, United States [‡]Singapore Management University, Singapore
[†]{yu.pan, wanjing.han, mu.zhang}@utah.edu [‡]yueduan@smu.edu.sg

A Artifact Appendix

A.1 Abstract

This artifact accompanies the paper "COLLISIONREPAIR: *First-Aid and Automated Patching for Storage Collision Vulnerabilities in Smart Contracts*". This repository contains scripts and results for evaluating smart contract patching. Detailed patching instructions are provided in `octopus/PATCHING.md`. The `README` file offers step-by-step guidance for running the evaluation.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

COLLISIONREPAIR is a system designed to patch upgradeable smart contract bytecode with storage collision vulnerabilities. All experiments in this study were conducted in controlled, sandboxed environments using Ganache with isolated snapshots of the Ethereum blockchain. As a result, traditional ethical concerns associated with cybersecurity research—such as experimentation on live systems, data privacy, or human subject involvement—were not applicable. The experiments posed no risk to real-world blockchain deployments or operations, even when executing vulnerable contracts or replaying attack transactions.

Importantly, COLLISIONREPAIR is a mitigation tool, not a detection mechanism—it reveals vulnerabilities only when an actual attack occurs. The vulnerable contracts evaluated in our study were sourced from the CRUSH dataset, where issues had already been disclosed and, in some cases, exploited. Our transaction replays confirm that COLLISIONREPAIR successfully blocks these previously observed attacks. No new vulnerabilities were discovered in the additional contracts we tested; should any new issues emerge, we will follow responsible disclosure practices, including notifying developers and CISA using publicly available contact information on Etherscan.

This research complies with all relevant legal and ethical standards. No additional regulatory approvals were required. Our work advances the field of blockchain and smart contract security without introducing new risks or ethical concerns.

A.2.2 How to access

https://doi.org/10.6084/m9.figshare.29150726

A.2.3 Hardware dependencies

- **Processor:** Any 64-bit processor, such as Intel Core Processor Series.
- **Memory:** 4 GB RAM or higher.
- **Storage:** 1 GB or more available disk space.

A.2.4 Software dependencies

Tool	Purpose
Node.js >= v16	For deploying contracts and running scripts
Python >= 3.10.16	For static verification and patching
Ganache	Local Ethereum testnet (must be running)
Octopus	Bytecode instrumentation engine

Table 1: Tool requirements and their purposes.

Software dependencies are listed in Table 1. For more details and relevant links, please refer to the README.

A.2.5 Benchmarks

Around 4,000 contracts were used to verify the correctness of patching and to evaluate transaction replay behavior after patching. Details of the project structure can be found in the `README` under the *Directory Structure* section.

Data The `contracts.txt` file lists all contract addresses to be processed. The `results/` directory stores all output data for each contract, including ABI, bytecode, transactions, and replay results.

[*]Yu Pan and Wanjing Han contributed equally to this work.

Scripts All main scripts are located in `evaluation/correctness/scripts/correctness/`. These include deployment, replay, patching, and result filtering scripts.

Core Patch Function The `patch.sh` script is provided to patch contracts in the correctness dataset. Run it from either the root directory or the `evaluation/correctness` directory to apply automated patching to all contracts listed in `contracts.txt`.

A.3 Set-up

A.3.1 Installation

Install Octopus Our patching tool is built on a modified version of `Octopus` (https://github.com/FuzzingLabs/octopus).

You can install it using either of the following methods:

```
cd octopus
sudo python3 setup.py install
cd -
```

or

```
cd octopus
pip install -r requirements.txt
cd -
```

Install Node.js Dependencies Navigate to the correctness evaluation directory and install the required Node.js packages:

```
cd evaluation/correctness
npm install
cd -
```

A.3.2 Basic Test

Verifying Patched Bytecode **Script:** `verify_patch.sh` `evaluation/scripts/verify_instrumentation.py`: This script takes a folder as input, automatically compares the original and patched bytecode, and displays the results.

```
chmod 777 verify_patch.sh
./verify_patch.sh
```

Expected outputs can be found in the `README` under the *Verifying Existing Results* section.

A.4 Evaluation workflow

A.4.1 Major Claims

COLLISIONREPAIR can successfully patch all contracts in the provided benchmark and can replay most real-world transactions both before and after patching in the Ganache environment.

A.4.2 Experiments

Applying the Patch Tool Before applying the patching tool, back up the existing `results/` directory to preserve previous outputs:

```
cp -r evaluation/correctness/results evaluation/
    correctness/results_backup
```

Deploy the StorageTracker Contract Ganache must be running at http://127.0.0.1:7545.
Deploy the monitoring contract and update the configuration:

```
node js_scripts/monitor/deploy_monitor.js
```

Run the Patching Script After deploying StorageTracker, apply the patch to all contracts in the dataset using the following commands:

```
chmod 777 patch.sh
./patch.sh
```

Example output:

```
INFO - [+] Runtime code detected
INFO - [+] Runtime code detected
WARNING - function signatures collision: ['symbol()',
     'link_classic_internal(uint64,int64)']
WARNING - function signatures collision: ['decimals
    ()', 'available_assert_time(uint16,uint64)']
...
```

Performance: Patching 4,000 contracts takes approximately 2 to 4 hours depending on hardware.

Regenerating transaction replay results is optional and detailed in the *Regenerating Dynamic Execution Results* section of the `README`.

A.5 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/.