
# USENIX Security '25 Artifact Appendix: Place Protections at the Right Place: Targeted Hardening for Cryptographic Code against Spectre v1

Yiming Zhu, Wenchao Huang*, Yan Xiong
University of Science and Technology of China

## A Artifact Appendix

### A.1 Abstract

This appendix provides an overview of each component of the artifact and describes how to use them to reproduce the results presented in the paper. We offer a complete artifact package that includes source code, workloads and documentation.

The artifact consists of five folders: `bench`, `Poc`, `include`, `src`, and `doc`.

The `src` and `include` folders contain the source code for our proposed hardening techniques: LightSLH, LightFence, and LightCut.

The main directory includes a `README.md` file that provides details about key dependencies, instructions for compiling the code, and a simple guide on how to use it.

The `doc` folder contains a `README.md`, including descriptions on how to annotate security properties for the code under analysis and how to mark entry functions for analysis. Additionally, it provides two examples to help users understand how to utilize our tools effectively.

The `Poc` folder includes a proof-of-concept implementation of the proposed side channel described in Section 7.2 of the paper.

Our experimental results can be reproduced by following the step-by-step instructions provided in the `README.md` file located in the `bench` folder. This file outlines the process for downloading the required dependencies and workloads (all of which are open-source).

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

This tool is a code hardening utility designed to enhance code security through static analysis. It does not involve any destructive operations or disable any security mechanisms. Therefore, evaluators can execute this tool without concerns regarding the security of their machines, data privacy, or other ethical issues.

*Corresponding Authors

#### A.2.2 How to access

The artifact is available at https://doi.org/10.5281/zenodo.15569395. It can be downloaded as a zip file, which contains all the necessary files to run the experiments described in the paper.

#### A.2.3 Hardware dependencies

None. However, the proof of concept described in Section 7.2 has been tested only on an Intel Xeon® CPU E5-2680.

#### A.2.4 Software dependencies

The artifact has been tested on Ubuntu 22.04 systems. To install our proposed tool (i.e., LightHardening), the following software dependencies are required:

- GCC and G++ (version 12 or higher), CMake (version 3.25 or higher), Make, and Wget. These dependencies can be installed by running: `apt install gcc-12 g++-12 make wget`. Note that the default Ubuntu apt repository may not include the latest version of CMake. It is recommended to use the official installation script for CMake: `wget https://github.com/Kitware/CMake/releases/download/v3.25.0/cmake-3.25.0-linux-x86_64.sh`, `chmod +x cmake-3.25.0-linux-x86_64.sh`, and `sudo ./cmake-3.25.0-linux-x86_64.sh -prefix=/usr/local`. You can verify the installation by running `cmake -version`.
- LLVM and Clang. Our tool is implemented as an LLVM IR pass and supports multiple LLVM versions. However, to reproduce our experimental results, LLVM version 14.0.4 is required. Due to the specific version requirement for LLVM and the directory structure expected by subsequent execution scripts, please refer to the **Install LLVM** section in `bench/README.md` for detailed instructions on installing LLVM 14.0.4. Note that installing LLVM can be time-intensive, potentially requiring 1-2 hours to complete.

To reproduce our experiments, the following additional software dependencies are required:

- Python 3, Git. These dependencies can be installed by running: `apt install python3 git python3-pip`.
- Dependencies for LLSCT: gperftools, libunwind, ninja, pkg-config, and Python packages: pandas, seaborn. These dependencies can be installed by running: `apt install google-perftools libunwind-dev ninja-build pkg-config` and `pip3 install pandas seaborn`.
- LLSCT. Please refer to the **Install LLSCT** section in `bench/README.md` for detailed instructions on installing LLSCT. Note that LLSCT is a tool directly modified from LLVM, and its installation may require 1-2 hours to complete.
- Google Benchmark. Please refer to the **Install Google Benchmark** section in `bench/README.md` for detailed instructions on installing Google Benchmark.

#### A.2.5 Benchmarks

We use OpenSSL, Libsodium, NaCL, and PQClean as our workloads. Enter the `bench` folder and run `bash collect_source_code.sh` to prepare these workloads. The script is pre-configured with the specific versions of the libraries used. The downloaded cryptographic source code will be located in the `bench/crypto_code` directory.

### A.3 Set-up

#### A.3.1 Installation

To set up the environment, enter to the bench folder of LightHardening and follow the **Build LightHardening** part in `bench/README.md` for build our tools.

Upon successful execution, the `libSpectrePass.so` file will be generated in the `build` folder. This file represents the LLVM IR pass that implements the proposed hardening techniques.

Next, navigate to the `bench/wrapper` folder and execute `bash compile_wrap.sh`. This step prepares the `.ll` files required for subsequent analysis.

#### A.3.2 Basic Test

Navigate to the `bench/wrapper` folder and execute the following command: `bash compile.sh aes 16 OpenSSL3.3.0 AES`.

This command compiles the OpenSSL 3.3.0 AES workload with all the hardening tools used in our experiments, including LightSLH, LightFence, LightCut, SSLH, Fence, and LLSCT.

Upon successful execution, the following files will be generated in the `bench/wrapper/OpenSSL3.3.0/aes` folder: `aes_test_LightSLH`, `aes_test_LightFence`, `aes_test_LightCut`, `aes_test_SSLH`, `aes_test_fence`, `aes_test_llsct`, and `aes_test_origin`.

### A.4 Evaluation workflow

#### A.4.1 Major Claims

**(C1):** The analysis information of LightSLH, as illustrated in Figure 6 of the paper.
**(C2):** The overheads of various protection tools are compared, as shown in Figure 7 of the paper.
**(C3):** A one-bit side channel is constructed using the assignment within a loop, as described in Section 7.2 of the paper.

#### A.4.2 Experiments

**(E1):** Reproduce C1. [5 compute-minutes]
**Preparation:** Ensure all dependencies are installed and the code is compiled as described in the **Set-up** section. Navigate to the `bench/wrapper` folder.
**Execution:** Execute the following command: `bash bench.sh compile`
**Results:** The analysis information for each workload, corresponding to Figure 6 in our paper, including runtime and processed instruction data, will be saved in `bench/wrapper/compile_information.txt`.

**(E2):** Reproduce C2. [30 compute-minutes]
**Preparation:** Ensure E1 has been completed successfully, and navigate to the `bench/wrapper` folder.
**Execution:** Execute the following commands sequentially:
```
bash bench.sh run
python3 stat_bench.py > performance.txt
```
**Results:** The runtime of the code generated by different defense methods for each workload will be recorded in `wrapper/result.json`. Additionally, `wrapper/performance.txt` will summarize the overhead introduced by different defense mechanisms compared to unprotected code. Note: The measurement of CPU time may vary depending on processor environment, cache settings, and other factors. As a result, the obtained results may exhibit slight differences compared to those presented in Figure 7.

**(E3):** Reproduce C3. [1 compute-minutes]
**Preparation:** Navigate to the `Poc` folder and ensure that LLVM and Clang are installed.
If you have installed LLVM 14.0.4 as described in Section A.2.4, you can add the LLVM binary directory to your `PATH` by running: `export PATH=$(realpath ../bench/llvm-project-14.0.4.src/install/bin):$PATH`. Alternatively, you can use `apt install clang` to install a different version of Clang, as this part of the reproduction does not require a specific Clang version.
**Execution:** Follow the instructions in the `Poc/README.md` file to run the proof-of-concept code.

**Results:** When compiled without forbidden branch prediction, there is approximately an 80% chance of obtaining output consistent with the input when running the Python script in the `Poc` folder. In contrast, when compiled with forbidden branch prediction, the probability drops to 50%, which is equivalent to random guessing.

## 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/.