Appendix C

Sourcerer Artifact

In this Appendix, we provide the requirements, instructions, and further details necessary to reproduce the experiments from our paper `Sourcerer: channeling the void`.

C.1 Description and requirements

The artifact contains the material to reproduce the results: Porting Effort (Section 7.1 – Table 1), Performance Overhead (Section 7.2 – Table 2), Source of Performance Overhead (Section 7.3 – Table 3), and Sourcerer as a Sanitizer for Fuzzing Campaigns (Section 7.4 – Figure 1. This material is released under the Apache License 2.0, in line with the LLVM project Sourcerer builds upon.

1. *Accessing the artifact*: We release the artifact on a public GitHub repository. While the main branch contains the latest version of the code.

2. *Hardware dependencies*: The artifact requires a machine with at least 128GB of RAM and a 1TB disk. 16GB of RAM is sufficient to run the SPEC CPU evaluations.

3. *Software dependencies*: The artifact was tested on Ubuntu 20.04 and require the ability to run Docker containers. An active internet connection is also necessary for the Chromium evaluation. Additionally, `curl, git, docker`, and `pip` should also be installed.

4. *Benchmarks*: The artifact requires a copy of the SPEC CPU 2006 & 2017 benchmarks.[1]

C.2 Artifact installation

The initial step is to clone the repository and build the Docker image. As the artifact is provided on top of a fork of the LLVM project, we recommend to only proceed to a shallow clone of the last 100 commits. This can be achieved by running the following bash command: `git clone`

[1] `https://www.spec.org/cpu2006/` and `https://www.spec.org/cpu2017/`

`$REPO --single-branch --branch main --depth 100 Sourcerer`. Additionally, please run `pip install -r requirements.txt` from inside the repo to install dependencies.

Each experiment is encapsulated in one or multiple Docker container. The Dockerfile is available at the root of the artifact repository. We do not provide support for running the experiments locally.

C.3 Experiment workflow

Our artifact aims at reproducing the results from three experiments presented in the paper. The first aims at evaluating the compatibility of Sourcerer with existing software by quantifing the number of extra classes to instrument on top of type++. The second experiment evaluates the performance overhead of Sourcerer over standard C++ with the help of the SPEC CPU 2006 and 2017 benchmarks as well as quantifies the added security guarantees provided by Sourcerer. Then, we conduct an ablation study to understand where the overhead originate from. Lastly, the fourth experiment demonstrate the performance and security benefits of Sourcerer during a fuzzing campaign on OpenCV.

We propose to run this experiments sequentially as they are presented in the paper. The artifact provides scripts to run the experiments and collect the results. The scripts will also generate tables and figure similar to the ones presented in the paper.

More complete and detailed instructions, as well a minimal example, are available in the README file of the repository. We highly recommend following the instructions there as copying and pasting the commands from this document might introduce errors.

C.4 Major claims

- (C1) *Compatibility*: Sourcerer is compatible with existing C++ codebases with a few minor changes. This is showcased in experiment E1 which evaluates the number of extra classes to instrument as part of Sourcerer. These results are presented in Table 1 in the paper.

- (C2) *Performance Overhead*: Sourcerer incurs a negligible performance while protecting a vast amount of additional unrelated casts. Our experiment E2 highlights this trend on the SPEC CPU benchmarks. In the paper, the results are available in Table 2. The overhead numbers can slightly differ from the ones in the paper due to the different hardware configurations, but we expect the global trend across the benchmark to remain consistent.

- (C3) *Ablation Study of the Performance Overhead*: To better assess the cost of associated to the different component of Sourcerer, we conducted an ablation study. The results are presented in Table 3 and can be reproduced through the experiment E3.
- (C4) *Fuzzing Campaign*: Sourcerer can be used to test software project. As a proof of concept, we run a fuzzing campaign on the OpenCV project. The results are presented in Figure 1 and can be reproduced through the experiment E4.

C.5 Evaluation

In this section, we provide the detailed steps to run the experiments and process the results to get the tables presented in the paper. Overall, this process requires around two to three days of computation time on a powerful server. These instructions are also available in the README file of the artifact repository.

Experiment 1 (E1) - Claim (C1): Compatibility Analysis:

[2 humans minutes + 2 compute-hours] The experiment evaluates the extra number of classes necessary for SPEC CPU 2006 and 2017 benchmarks. The experiment consists of compiling the benchmarks with Sourcerer to first collect the classes to instrument.

[Preparation] Ensure that the two SPEC CPU benchmarks `.iso` are available at the root of the cloned repository.

[Execution] Run the commands:

```
# Build the docker images and then run two analysis (Property 1 and Property 2) on
↪ the SPEC CPU benchmarks (around 2hours).
# Expected output: Different logs highlighting first the Docker builds and then the
↪ benchmarks compilation and analysis. Finally, a table similar to Table 1 will be
↪ printed.
./table1.sh
```

[Results] Upon completion, the script will generate a table identical to Table 1.

Experiment 2 (E2) - Claim (C2): Performance Overhead [2 humans minutes + 15 compute-hours] This experiment runs the SPEC CPU benchmarks with different flavors of cast checking. First, a baseline is established by running the benchmarks with no cast checking. Then, the benchmarks are run with Sourcerer and LLVM-CFI to measure the overhead. Lastly, an extra run for both Sourcerer and LLVM-CFI is performed to measure the number of cast protected.

[Preparation] Again, ensure that the two SPEC CPU benchmarks '.iso' are available at the root of the cloned repository.

[Execution] In a shell, run the following command:

```
# Build the docker images and then run all the benchmarks variation (around
    15hours).
# Expected output: Different logs highlighting first the Docker builds and then the
    benchmarks compilation and execution. Finally, a table similar to Table 2 will
    be printed.
./table2.sh
```

[Results] Upon completion, the script will generate a table similar to Table 2. There might be variations up to 10% in the performance and memory overhead numbers due to the different underlying machine. In particular, in a shared environment, the overhead might show inconsistencies. The number of casts protected should remain identical to the one presented in the paper.

Experiment 3 (E3) - Claim (C3): Ablation Study: [2 humans minutes + 15 compute-hours] This experiment evaluates the cost of the different operations in the type checking process of Sourcerer.

[Preparation] Again, ensure that the two SPEC CPU benchmarks '.iso' are available at the root of the cloned repository.

[Execution] In a shell, run the following command:

```
# Build the docker images and then run the micro-benchmark (around 15 hours).
# Expected output: Different logs highlighting first the Docker build and then the
    cost of the different operations.
./table3.sh
```

[Results] Upon completion, the script will generate a table similar to Table 3. The numbers presented should exhibit similar ratios to the ones in the paper.

Experiment 4 (E4) - Claim (C4): Fuzzing campaign: [2 humans minutes + 25 compute-hours] This experiment evaluates the fuzzing performance of two sanitizers: Sourcerer and ASan.

[Preparation] Please run the following script:

```
# Setup the fuzzing campaign
./fig1_requirements.sh
```

[Execution] In a shell, run the following command:

```
# Build the docker images
# Expected output: Different logs highlighting first the Docker builds, and finally,
↪  a figures similar to Figure 1 will be saved.
./fig1.sh
```

[Results] Upon completion, the script will generate figures similar to the ones in Figure 1. They are saved in the folder `fuzzing_pics`. The trends should be similar as in the paper.