Cascade: CPU Fuzzing via Intricate Program Generation

Flavien Solt, Katharina Ceesay-Seitz, and Kaveh Razavi, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity24/presentation/solt

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: Cascade: CPU Fuzzing via Intricate Program Generation

Flavien Solt
ETH Zurich

Katharina Ceesay-Seitz
ETH Zurich

Kaveh Razavi
ETH Zurich

A Artifact Appendix

A.1 Abstract

In our Artifact, we provide the source code of Cascade and the tested designs. We provide the framework for performing the experiments described in this paper and analyzing the corresponding results. The Artifact is a precomputed Docker image, where all the results are readily present and can be re-computed at will. It aims at showing that Cascade is open-source and functions as described in the paper. It additionally allows reproducing the most important results of the paper that motivate and evaluate Cascade.

A.2 Description & Requirements

Software requirements Docker, `tar` and `make` are the only software requirements. Reproducers are free to run the experiments natively by following the steps performed in the Dockerfile.

Hardware requirements To reproduce the experiments, we expect a machine with at least 16 GB of RAM and 200 GB of free storage. Machines of at least 64 cores are preferred, but this is not a hard requirement.

A.2.1 Security, privacy, and ethical concerns

The risk that Reproducers face corresponds to the risk of running code from generally popular open-source repositories inside a Docker container.

A.2.2 How to access

The artifact repository is available at `https://github.com/comsec-group/cascade-artifacts/tree/2b797b546629a2df6010abd96e293044cd3cd285`.

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

Docker, `tar` and `make`.
Questasim (commercial EDA software) is optional.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

1. Method 1 (computation-intensive): Clone the git repository, then run `make build`. More information can be found in the repository's Readme.
2. Method 2 (no computation): get the Docker image, then extract the pre-generated figures as specified in the repository's Readme.

A.3.2 Basic Test

In the cloned repository, you can run `make build` to start the Docker image build process. If you pulled the Docker image, you can find the figures in `/cascade-meta/figures/` in the Docker image as specified in the repository's Readme.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Long programs tend to be more effective for fuzzing CPUs. This is shown by Figures 7, 8, 9 and 16.
(C2): DifuzzRTL executables have low completion, prevalence and dependencies. This is shown by Figures 1, 2 and 11.
(C3): Cascade executables significantly improve prevalence and dependencies. This is shown by Figures 10 and 12.
(C4): Cascade improves coverage over state-of-the-art fuzzers. This is shown by Figures 13, 14 (optionally 15, yet Figure 13 already shows a comparison with DifuzzRTL).
(C5): Cascade finds most of the bugs very fast. This is shown by Figures 16 and 18.

A.4.2 Experiments

(E1): *[Re-build the Docker image] [5 human minutes, 24 machine hours] (Optional):*
How to: In the `cascade-artifacts` repository, execute `make build`.
Preparation: None.
Execution: See *How to*.
Results: The Docker image build must eventually succeed, as will be displayed by Docker.

(E2): *[Extract the plots] [5 human minutes, 5 machine minutes]:*
How to: Create a Docker container from the downloaded or built Docker image, then extract the plots using `docker cp` as documented in the Readme in the artifacts repository.
Preparation: Execute the `make run` command in the `cascade-artifacts` repository. This will ensure that a Docker container is running, whose identifier can be used for copying out the results.
Execution: Execute the `docker cp` instructions specified in the Readme of the `cascade-artifacts` repository.
Results: The figures will be copied into the `figures` directory of the `cascade-artifacts` repository.

(E3): *[Collect Questasim coverage] [4 human hours, 24 machine hours] (Optional):*
This optional experiment is only accessible to users with Questasim licenses. The estimated human hours correspond to the preparation of the environment, which may be non-trivial. Most of the machine time is spent in merging the coverage files, which must be done sequentially to obtain the coverage increase at program granularity.
How to: Follow the steps described in the `cascade-artifacts` Readme.
Preparation: Either have Questasim work in the Docker container, or reproduce the Docker steps on a bare-metal Linux machine. We recommend using Ubuntu to ensure closeness with the Dockerfile, so you are certain that your dependencies will work, yet there is no strict distribution requirement.
Results: `cascade-meta/figures/modelsim.png`

A.5 Notes on Reusability

- Regarding the experiments, as indicated in the `cascade-artifacts` Readme, some experiments may be run with reduced parameters to complete in a reasonable amount of time, while conserving the tendency shown in the paper.

- Regarding the fuzzer, new designs can be added in the same way as the evaluated designs are integrated. Please refer to `Readme.md`, `cascade-meta/design-processing/design_repos.json` and `cascade-meta/fuzzer/common/designcfgs.py`.

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/usenixsec2024/.