USENIX Security '26 Artifact Appendix: *DRVFuzz*: Data-Sensitive RISC-V CPU Fuzzing

Zehong Yu
Tsinghua University

Yuanliang Chen
Renmin University of China

Zhen Yan
Tsinghua University

Xudong Zhang
Tsinghua University

Zhensheng Xian
Tsinghua University

Yu Jiang
Tsinghua University

A Artifact Appendix

A.1 Abstract

We provide the source code of *DRVFuzz* along with a Docker-based evaluation setup to facilitate reproducing the results presented in our paper and to support *DRVFuzz*'s deployment for evaluating future RISC-V CPUs. These artifacts demonstrate that *DRVFuzz* is a fully functional data-sensitive CPU fuzzing tool capable of generating SDModel-guided test programs, executing differential fuzzing campaigns, and comparing coverage and transition-count results against existing RISC-V fuzzers.

A.2 Description & Requirements

Artifact contents. The artifact contains the following components.

- *DRVFuzz* source code, including the SDModel sensitive-data generator, execution-state labeler, transition extractor, and transition-guided seed feedback.
- Bundled simulator binaries and a RISC-V GNU toolchain used by the Docker image.
- A `README.md` with quick-start commands and and manual examples.

A.2.1 Security, privacy, and ethical concerns

DRVFuzz is a local RISC-V CPU fuzzing framework that generates and executes test programs on simulator-based CPU models. The artifact runs fuzzing workloads inside Docker and does not access production systems or third-party services. We recommend running it in an isolated container or VM. The reported bugs were handled through coordinated disclosure.

A.2.2 How to access

Our artifacts are accessible for permanent access on Zenodo at https://doi.org/10.5281/zenodo.20343558. Alternatively, they can be accessed via GitHub at https://github.com/YzhDDDing/DRVFuzz.

A.2.3 Hardware dependencies

We recommend an x86_64 Linux host with at least 8 CPU cores, 32 GiB RAM, and 100 GiB free disk space for reduced experiments. Full 24-hour campaigns benefit from 200 GiB free disk space and a CPU comparable to the paper's evaluation machine. macOS hosts can use Docker Desktop; Apple Silicon users should run the container with `-platform linux/amd64`.

A.2.4 Software dependencies

The host needs Docker, `unzip`, and standard shell utilities. The Docker image contains the remaining dependencies, including Python packages, the RISC-V GNU toolchain, *DRVFuzz*, and simulator binaries. Internet access is needed only to build the image, unless a prebuilt image is provided.

A.2.5 Benchmarks

The experiments require six open-source RISC-V CPU implementations as benchmark targets: BOOM-V3, BOOM-V4, Rocket, CVA6, Kronos, and Srv32. The artifact includes their RTL sources, simulation environments, and CPU configuration files.

A.3 Set-up

A.3.1 Installation

Unpack the artifact file and build the Docker image:

```
unzip DRVFuzz_artifact.zip
cd DRVFuzz_Impl
docker build -t drvfuzz-ae:latest .
```

If Docker cannot resolve package mirrors, build with host networking:

```
docker build --network=host -t drvfuzz-ae:latest .
```

Start an interactive container and mount a host output directory:

```
mkdir -p runs
docker run --rm -it -v "$PWD/runs:/runs" \
  drvfuzz-ae:latest
```

On Apple Silicon macOS hosts, use:

```
docker run --platform linux/amd64 --rm -it \
  -v "$PWD/runs:/runs" drvfuzz-ae:latest
```

A.3.2 Basic Test

Inside the container, run:

```
DRVFuzz --help
DRVFuzz diff \
  --testcase-config configs/rv64.toml \
  --riscv-impl spike cva6 \
  --run-dir /runs/smoke_cva6_spike \
  --impl-timeout configs/timeout.toml \
  --run-options configs/run_options.toml \
  --concurrency 1 \
  --max-runs 5 \
  --sdmodel \
  --transition-guided
```

The expected result is a successful short differential-fuzzing run under `/runs/smoke_cva6_spike`. The directory contains the generated programs, Spike and CVA6 execution logs, and SDModel transition reports. If a mismatch is found during the smoke test, *DRVFuzz* also writes the corresponding failure report and reproducer.

A.4 Evaluation workflow

A.4.1 Major Claims

C1. *DRVFuzz* detects data-sensitive bugs in real-world RISC-V CPUs. In the paper, *DRVFuzz* found 22 previously unknown bugs across BOOM-V3, BOOM-V4, Rocket, CVA6, Kronos, and Srv32; 20 were assigned CVEs. This claim is supported by Tables 2 and 3.

C2. *DRVFuzz* achieves higher microarchitectural coverage than Cascade and DiveFuzz on the six evaluated CPUs. This claim is supported by Figure 8.

C3. *DRVFuzz* explores more data-sensitive execution-state transitions than Cascade and DiveFuzz. This claim is supported by Table 4, which reports average unique transitions per 1000 instructions.

C4. Both components, SDModel and transition-guided fuzzing, contribute to bug-finding effectiveness. This claim is supported by the ablation study in Figure 9.

A.4.2 Experiments

(E1): *[Build and smoke test.] [5 human minutes, 2 machine hours] (Optional):*
How to: follow the installation and basic functionality test above.
Preparation: Build the Docker environment and ensure that the required dependencies, RISC-V toolchains, simulators, and benchmark configurations are available.
Execution: Run the basic functionality test with a short SDModel-enabled, transition-guided CVA6-vs-Spike differential fuzzing campaign.
Results: Docker build succeeds, the CLI prints its help message, and a short SDModel-enabled, transition-guided CVA6-vs-Spike differential-fuzzing run completes with per-run logs and transition reports.

(E2): [Bug-finding campaign.] [10 human minutes, 24 machine hours]:
How to: run *DRVFuzz* differential fuzzing with SDModel and transition-guided feedback. For example, for Rocket:

```
DRVFuzz diff
--testcase-config configs/rv64.toml
--riscv-impl spike rocket
--run-dir /runs/bug_rocket_full
--impl-timeout configs/timeout.toml
--run-options configs/run_options.toml
--concurrency 1
--sdmodel
--transition-guided
```

For BOOM-V3, BOOM-V4, and CVA6, replace `rocket` with `boom-v3`, `boom-v4`, or `cva6`. For Kronos and Srv32, use `configs/rv32.toml` and replace the implementation name with `kronos` or `srv32`.
Preparation: Select the target RISC-V CPU benchmark and prepare the corresponding configuration files. Ensure that the CPU simulator and Spike reference model are correctly configured for differential testing.
Execution: Launch DRVFuzz with SDModel and transition-guided fuzzing enabled. The campaign generates test programs, executes them on both the target CPU and Spike, and detects architectural mismatches.
Results: If a mismatch is found, DRVFuzz creates a per-run directory containing logs, generated assembly, execution outputs, and minimized reproducers. Full bug rediscovery is stochastic; the expected paper-scale outcome is the trend reported in Table 3, where DRVFuzz detects substantially more bugs than Cascade and DiveFuzz.

(E3): *[Coverage comparison.] [5 human minutes, 24 machine hours]:*
How to: `run_coverage_compare /runs/coverage_24h`
Preparation: Prepare the coverage comparison environment and ensure that the required benchmark results

from the fuzzing campaigns are available.
Execution: Run the coverage comparison script to collect branch and toggle coverage results for the evaluated RISC-V CPUs.
Results: Each target directory contains coverage results, which can be further used to plot coverage trends. The expected paper-scale result is the same trend as Figure 8: DRVFuzz achieves the highest final branch and toggle coverage on the evaluated targets.

(E4): *[Transition-count comparison.] [5 human minutes, 1 machine hour]:*
How to: `run_transition_count_compare`
Preparation: Prepare the transition-count evaluation environment and collect the normalized execution inputs required for transition analysis.
Execution: Run the transition-count comparison script to measure the number of unique transitions triggered by different fuzzing approaches.
Results: The expected paper-scale trend is that DRVFuzz achieves the largest unique-transition density on every target, matching Table 4.

(E5): *[Component ablation.] [10 human minutes, 24 machine hours]:*
How to: Use the same differential fuzzing command as E2 while toggling the exposed flags. The full *DRVFuzz* configuration enables both `-sdmodel` and `-transition-guided`. The $DRVFuzz_{m^-}$ variant omits `-sdmodel`; the $DRVFuzz_{g^-}$ variant uses `-sdmodel` but omits `-transition-guided`; the $DRVFuzz_{\text{mode}}$ variant uses `-sdmodel -mode-guided`.
Preparation: Prepare the same benchmark configurations and execution environment used in E2. Configure the desired DRVFuzz variant by enabling or disabling the corresponding command-line flags.
Execution: Run differential fuzzing campaigns for each DRVFuzz variant under identical experimental settings.
Results: Compare the number of detected bugs and detection time among different configurations. The full configuration is expected to find bugs faster and more frequently than the variants, consistent with Figure 9.

A.5 Notes on Reusability

The artifact is designed so that new RISC-V CPUs can be integrated by adding a simulator wrapper, target configuration, memory layout, and trace extraction interface. The SDModel and transition-count logic are shared across targets. The comparison scripts intentionally preserve generated seeds, logs, transition reports, and coverage histories under `/runs` so that evaluators can inspect or replay individual samples.

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/usenixsec2026/.