MorFuzz: Fuzzing Processor via Runtime Instruction Morphing enhanced Synchronizable Co-simulation

Jinyan Xu and Yiyuan Liu, *Zhejiang University;* Sirui He, *City University of Hong Kong;* Haoran Lin and Yajin Zhou, *Zhejiang University;* Cong Wang, *City University of Hong Kong*

https://www.usenix.org/conference/usenixsecurity23/presentation/xu-jinyan

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: MorFuzz: Fuzzing Processor via Runtime Instruction Morphing enhanced Synchronizable Co-simulation

Jinyan Xu
Zhejiang University
phantom@zju.edu.cn

Yiyuan Liu
Zhejiang University
yiyuanliu@zju.edu.cn

Sirui He
City University of Hong Kong
sol.he@my.cityu.edu.hk

Haoran Lin
Zhejiang University
haoran_lin@zju.edu.cn

Yajin Zhou
Zhejiang University
yajin_zhou@zju.edu.cn

Cong Wang
City University of Hong Kong
congwang@cityu.edu.hk

A Artifact Appendix

A.1 Abstract

As introduced in the paper, MorFuzz discovers several new bugs across open-source RISC-V processors with different microarchitectures and significantly improves the efficiency and effectiveness of processor fuzzing. Our artifact provides binaries and scripts to reproduce those results. This appendix describes the steps to set up our prototype and run our evaluation experiments.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is available at https://github.com/sycuricon/MorFuzz/releases/tag/usenix23. Both MorFuzz pre-built binaries and the inputs generated by DifuzzRTL that we replayed are available at https://zenodo.org/record/8055696.

A.2.3 Hardware dependencies

MorFuzz uses commercial EDA software, so an x86 processor is required. We evaluate MorFuzz on a 48-core dual Intel Xeon Silver 4214 server with 256GB RAM. In addition, at least 280 GB of storage is required. Storing the input generated by DifuzzRTL requires 270 GB, and the evaluation also consumes about 10 GB of storage.

A.2.4 Software dependencies

Our prototype contains three components: an instruction generator, a co-simulation library, and a top-level fuzzing framework. We release the instruction generator and the co-simulation library of MorFuzz as pre-built binaries, they are compiled with GCC 10.2.1 on CentOS 7.9.2009. In order to run MorFuzz the same operating system and compiler are required. MorFuzz uses the Synopsys VCS, a commercial RTL simulator, to simulate processor designs. You need to purchase licenses from Synopsys to use VCS. In addition, in order to cross-compile RISC-V programs, a RISC-V toolchain is also required, which is available at the official riscv-gnu-toolchain repository on the GitHub.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

First, MorFuzz requires the following dependencies, and in order to run the experiment you also need to set up dependencies from riscv-dv and riscv-torture:

```
sudo yum -y groupinstall "Development Tools"
sudo yum -y install redhat-lsb libXScrnSaver
    centos-release-scl dtc
sudo yum -y install devtoolset-10
```

Second, clone the repository and execute the setup script.

```
git clone https://github.com/sycuricon/MorFuzz.git
cd MorFuzz
git checkout usenix23
git submodule update --init --recursive
export ARTIFACT_ROOT=$(pwd)
./scripts/setup.sh
```

Next, download the MorFuzz pre-built binaries `morfuzz_bin.zip` from https://zenodo.org/record/8055696 and unzip it. You also need to place the decompressed `morfuzz_bin` directory under the root directory of the MorFuzz repository.

Finally, download the input sets `difuzzrtl_[0-4].zip` generated by DifuzzRTL from https://zenodo.org/record/8055696 and unzip them. You do not need to copy them into the repository, making the `DIFUZZRTL_INPUT` environment variable point to one of the input sets is enough.

The final directory structure of the project is as follows:

```
MorFuzz
├── dep
├── morfuzz_bin
│   ├── cj
│   └── razzle
├── patch
├── scripts
└── src
```

A.3.2 Basic Test

Before executing each experiment, you need to point the `ARTIFACT_ROOT` environment variable to the directory where the MorFuzz repository was cloned and execute the `env.sh` script to set up the other environment variables.

```
export ARTIFACT_ROOT=#absolute path to MorFuzz#
cd $ARTIFACT_ROOT
source ./scripts/env.sh
```

After executing the script if there are no complaints about missing dependencies, you can execute the basic test script. The script invokes Rocket, BOOM, CVA6, and Spike in turn to execute a normal test case, and if the test passes the "`*** PASSED ***`" message will appear on the terminal.

```
./scripts/basic.sh
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): MorFuzz is compatible with different microarchitectures and identified new bugs. This claim is supported by experiment (E1).

(C2): MorFuzz can efficiently achieve better coverage than DifuzzRTL and other techniques. This claim is supported by experiment (E2).

(C3): MorFuzz is capable of generating more diverse inputs than DifuzzRTL, and is comparable to riscv-dv. This claim is supported by experiment (E3).

(C4): Instruction morphing and state synchronization can help MorFuzz achieve better coverage. This claim is supported by experiment (E4).

A.4.2 Experiments

(E1): Executing test cases that trigger discovered bugs on the corresponding processors to prove that MorFuzz can be used on different microarchitectures, for details see `src/table2/README.md`.
Estimated time: less than 5 human-minute, and less than 5 compute-minute.
Preparation: Go to the MorFuzz repository root directory, set up `ARTIFACT_ROOT` variable and execute `scripts/env.sh`.
Execution: Execute `scripts/tab2.sh`.
Results: Trigger bugs in Table 2. For a detailed analysis of each result, please refer to `src/table2/README.md`.

(E2): Evaluating coverage to prove that MorFuzz achieves better coverage, for details see `src/figure8/README.md`.
Estimated time: less than 5 human-minute, and 24 compute-hour.
Preparation: Go to the MorFuzz repository root directory, set up `ARTIFACT_ROOT` and `DIFUZZRTL_INPUT` variables and execute `scripts/env.sh`.
Execution: Execute `scripts/fig8.sh`.
Results: Reproduce Figure 8, you can find the figure at `scripts/output/fig8.pdf`.

(E3): Evaluating instruction diversity to prove that MorFuzz generates instructions with good diversity, for details see `src/figure9/README.md`.
Estimated time: less than 5 human-minute, and 24 compute-hour.
Preparation: Go to the MorFuzz repository root directory, set up `ARTIFACT_ROOT` and `DIFUZZRTL_INPUT` variables, and execute `scripts/env.sh`.
Execution: Execute `scripts/fig9.sh`.
Results: Reproduce Figure 9, you can find three heatmaps named `heatmap_<name>.pdf` in the `scripts/output` directory.

(E4): Evaluating coverage to prove that MorFuzz's subcomponents can help MorFuzz achieve better coverage, for details see `src/figure10/README.md`.
Estimated time: less than 5 human-minute, and 24 compute-hour.
Preparation: Go to the MorFuzz repository root directory, set up `ARTIFACT_ROOT` variable and execute `scripts/env.sh`.
Execution: Execute `scripts/fig10.sh`.
Results: Reproduce Figure 10, you can find the figure at `scripts/output/fig10.pdf`.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.