Toward Unbiased Multiple-Target Fuzzing with Path Diversity

Huanyao Rong, *Indiana University Bloomington;* Wei You, *Renmin University of China;* XiaoFeng Wang and Tianhao Mao, *Indiana University Bloomington*

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

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

USENIX Security '24 Artifact Appendix: Toward Unbiased Multiple-Target Fuzzing with Path Diversity

Huanyao Rong
Indiana University Bloomington

Wei You [*]
Renmin University of China

XiaoFeng Wang
Indiana University Bloomington

Tiaohao Mao
Indiana University Bloomington

A Artifact Appendix

A.1 Abstract

This is AFLRun, our directed fuzzer. In this appendix, we introduce how to use it. Additionally, we introduce how to reproduce our experiments.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Not applicable.

A.2.2 How to access

Please see our GitHub repository: AFLRun.

A.2.3 Hardware dependencies

The x86-64 architecture is required.

A.2.4 Software dependencies

Ubuntu with version higher than 20.04 is required. Besides, we also need `gcc` and `makefile` being installed.

A.2.5 Benchmarks

We leverage two benchmarks for our experiments: the Magma benchmark and a set of vulnerabilities commonly evaluated by directed fuzzers. The benchmark can be downloaded here.

A.3 Set-up

A.3.1 Installation

The AFLRun is tested with clang 16.0.3, the other version might work but might also be problematic. These are the steps to compile the LLVM project for AFLRun.

*Corresponding author.

```
# Clone LLVM project.
git clone --depth=1 https://github.com/llvm/llvm-project.git && \
        cd llvm-project && \
        git fetch origin --depth=1 \
                4a2c05b05ed07f1f620e94f6524a8b4b2760a0b1 && \
        git reset --hard 4a2c05b05ed07f1f620e94f6524a8b4b2760a0b1

# Download binutils.
wget https://ftp.gnu.org/gnu/binutils/binutils-2.39.tar.gz \
                -O binutils.tar.gz && \
        tar -xf binutils.tar.gz

# Download CMake.
wget https://github.com/Kitware/CMake/releases/download/v3.25.1/\
cmake-3.25.1-linux-x86_64.tar.gz -O cmake.tar.gz && \
        tar -xf cmake.tar.gz

# Compile and install LLVM project.
# Please change "/path/to/install" to your install path.
PATH_TO_INSTALL="/path/to/install"
mkdir build && cd build
export CXX=g++
export CC=gcc
../cmake-3.25.1-linux-x86_64/bin/cmake -G "Ninja" \
  -DLLVM_BINUTILS_INCDIR=$PWD/../binutils-2.39/include \
  -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD=host \
  -DLLVM_ENABLE_PROJECTS="clang;compiler-rt;lld" \
  -DCMAKE_INSTALL_PREFIX="$PATH_TO_INSTALL" \
  -DLLVM_INSTALL_BINUTILS_SYMLINKS=ON $PWD/../llvm/
ninja -j $(nproc) && ninja install
cd ../.. && rm -rf llvm-project
```

Listing 1: Compile LLVM.

Then we can compile AFLRun.

```
git clone https://github.com/Mem2019/AFLRun.git && cd AFLRun
git submodule update --init robin-hood-hashing/
export CC="$PATH_TO_INSTALL/bin/clang"
export CXX="$PATH_TO_INSTALL/bin/clang++"
make clean all
AFLRUN="$PWD"
```

Listing 2: Compile AFLRun.

Now we can use AFLRun to compile program.

```
# Set target file, the format is same as AFLGo.
export AFLRUN_BB_TARGETS="/path/to/BBtargets.txt"
# Names of target binaries to instrument,
# "::" means instrument all binaries.
export AFLRUN_TARGETS="bin1:bin2"
# Optional, directory to store data.
# If not set, a random directory will be created.
export AFLRUN_TMP="/tmp/"
export CC="$AFLRUN/afl-clang-lto"
export CXX="$AFLRUN/afl-clang-lto++"
$CC hello.c -o hello
```

Listing 3: Using AFLRUN.

For markdown version, please read the README.md in our repository.

A.3.2 Basic Test

Please refer to Listing 4. After running the command, we should see the UI similar to that of AFL++.

```
sudo bash -c "echo core >/proc/sys/kernel/core_pattern"
export AFL_NO_AFFINITY=1
./afl-fuzz -i ./corpus -o ./output -m none -d -- ./hello @@
```

Listing 4: How to run AFLRUN.

A.4 Evaluation workflow

A.4.1 Major Claims

- AFLRUN can reproduce known vulnerabilities faster.
- AFLRUN can assign energy among targets in a fairer manner.
- AFLRUN does not incur significant overhead when the number of targets is large.

A.4.2 Experiments

Magma Benchmark.
Go to directory `magma/tools/captain` (Listing 5).

```
cd magma/tools/captain
```

Listing 5: Go to `magma/tools/captain`.

To run all experiments at once, simply run `./run.sh`. However, it is recommended to run only two fuzzers at once, because Magma sometimes may fail and cause the fuzzing session stops accidentally. Running two fuzzers allows us to restart the sessions that failed only instead of restarting everything. Besides, it is recommended to firstly set `TIMEOUT=10m` to firstly build the docker image and to try if the fuzzing is successful.

For example, to run only AFLRUN and AFLRUN, change `FUZZERS` variable in `captainrc` to (Listing 6):

```
FUZZERS=(aflrun_noctx aflrun_noctxdiv)
```

Listing 6: Set fuzzers.

Additionally, `WORKERS` is the total number of CPU cores for running the experiment. You may need to change this according to number of cores in your machine (i.e., `echo $(($(nproc)-10)))`.

After changing configurations, you may run `./run.sh` to start the experiment (Listing 7).

```
sudo bash -c "echo core >/proc/sys/kernel/core_pattern"
./run.sh
```

Listing 7: Run experiment.

After the experiment, to collect data, run the following command (Listing 8). Note that if you changed the `WORKDIR` in the `captainrc`, you may also need to change the `workdir` here.

```
python3 ../benchd/exp2json.py --workers 10 workdir/ workdir/out.json
python3 ../benchd/survival_analysis.py -n 10 -t 86400 \
        workdir/out.json > workdir/out.csv
```

Listing 8: Obtain the results.

To merge all fuzzers that are run in different campaign, use the script `merge_fuzzers.py` (Listing 9).

```
python3 ../benchd/merge_fuzzers.py all.json \
        workdir1/out.json workdir2/out.json workdir3/out.json
python3 ../benchd/survival_analysis.py -n 10 -t 86400 \
        all.json > all.csv
```

Listing 9: Merging different experiments.

The `.csv` file already contains the final results. To draw the LaTeX table of all fuzzers, use the `draw_table.py` script (Listing 10).

```
python3 ../benchd/draw_table.py all.json all.csv
```

Listing 10: Draw table.

Fairness Study.
For this experiment, we also enter `magma/tools/captain`, but we run `./run.sh impartialityrc` instead. The `impartialityrc` is the configuration file for this experiment, and the result work directory will be `impartiality`.

The experiment is similar, after the experiment, to generate the figure, we should run the command in Listing 11.

```
python3 ../benchd/imparitiality.py impartiality/ar/ temp
ls *.png # All figures will be stored as PNG files.
```

Listing 11: Generate figure.

Overhead Study.

To reproduce our overhead measurement, we can go to `magma/tools/captain` similarly, and then run `./run.sh overheadrc`. The working directory `overhead` will be produced. We can then finally run the script to draw the LaTeX table (Listing 12):

```
python3 ../benchd/overhead.py overhead/ar/
```

Listing 12: Overhead results.

Additional Experiment.

We still use the Magma environment for this experiment. The experiment directory is `magma-extra`. The process of running the experiment is generally same as the previous Magma experiment, but we need to generate the pseudo Magma monitor files according to the crashes. This allows us to use the same scripts to obtain the relevant results. To achieve this, after running the fuzzing campaigns with the working directory `workdir` being generated, we should run `gen_magma_monitor.py` on the directory. (Listing 13)

```
python3 ../benchd/gen_magma_monitor.py workdir 80
```

Listing 13: Additional experiments.

Then the post process to obtain the results is same as the one of Magma described previously.

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/usenixsec2024/.