ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

USENIX'23 Artifact Appendix: FISHFUZZ: Catch Deeper Bugs by Throwing Larger Nets

Han Zheng, Jiayuan Zhang, Yuhang Huang, Zezhong Ren, He Wang,
Chunjie Cao, Yuqing Zhang, Flavio Toffalini, Mathias Payer

A Artifact Appendix

A.1 Abstract

The artifact of FISHFUZZ contains the source code, supportive materials along with the documents and scripts to reproduce the results. We release the artifact to help user reproduce the two-stage evaluation results in the paper without too much manual effort.

A.2 Description & Requirements

The FISHFUZZ artifact consists of the following components:

Source Code FISHFUZZ is a novel input prioritization strategy, to demonstrate the generality, we implement FISHFUZZ based on AFL and AFL++ respectively (FF_{AFL} and FF_{AFL++}). In this artifact, we release both of them and provide a wrapper to automate the compilation.

Supportive Materials Due to page limitation, we didn't include all raw data in the paper. In this artifact, we attach the (1) raw data for p-value calculation (2) evaluation results for different hyperparameters (3) full list of new bugs FISHFUZZ found.

Reproduction Scripts In this artifact we provide the scripts and the dockerfile that help users automatically build the benchmark, start evaluations and analysis the results. The benchmark included in the artifact is the two-stage benchmark in the paper.

A.2.1 How to access

- The artifact is available at https://github.com/HexHive/FishFuzz.
- The version we used for artifact evaluation is https://github.com/HexHive/FishFuzz/commit/911637cdf7448b97eccf1c9664ef318aff884b63.

A.2.2 Hardware dependencies

In the evaluation, we use a server equipped with Xeon Gold 5218(22M Cache, 2.30 GHz), 64 GB memory and 1TB disk space. To reproduce this evaluation, more than 50GB disk space is required.

A.2.3 Software dependencies

We plug this evaluation into the docker environment, therefore the user only need to have a Linux server with docker and git installed.

A.2.4 Benchmarks

The benchmark included in the artifact is two-stage benchmark in the paper, which consists of 7 real-world programs that have various types of input format. We provide a dockerfile that automated the build process.

A.3 Set-up

We provide a detailed README in the folder 'paper/artifact'. By following the instructions listed, the evaluation results could be easily reproduced.

Tips: we allocate one cpu core for each fuzzer-benchmark pair, and by default the evaluation contains 4 fuzzers * 7 benchmarks, which requires 28 cores to run all campaigns at the same time. Therefore the users could remove some programs or fuzzers according to their hardware bandwidth.

```

# for two-stage
export BENCHMARK_NAME=two-stage
export IMAGE_NAME=fishfuzz:ae-twostage

git clone git@github.com/Hexhive/FishFuzz && \
cd FishFuzz/paper/artifact/$BENCHMARK_NAME

# build base docker images
docker build -t $IMAGE_NAME .

# create scripts for fuzzers to run
python3 scripts/generate_script.py \
        -b "$PWD/runtime/fuzz_script"
```

Listing 1: Build Evaluation Image

To prepare the artifact evaluation, user should first download the repo and build the docker image, as depicted in Listing 1. This step is expected to take about 1.5h. Afterward, run the generate_script.py to generate the scripts for fuzzers to run.

Afterward, we provide a script to generate the docker commands, considering many servers didn't have enough cores (>= 28 cores), we only print the command and the user could copy-paste to run. After 24h, the evaluations are done and we should follow the given instructions for post-processing (Listing 2)

```

# generate command & copy-paste
python3 scripts/generate_runtime.py \
        -b "$PWD/runtime"

# wait 24h and stop all
docker rm -f $(docker ps -a -q
    -f "ancestor=$IMAGE_NAME")

# give w/r permission
sudo chown -R $(id -u):$(id -g) runtime/out

# copy evaluation results to results folder
mkdir results/
python3 scripts/copy_results.py \
      -s "$PWD/runtime" \
      -d "$PWD/results/" -r 0


...
```

Listing 2: Run the Evaluation

Finally, create a new container, copy the results dir into root dir and run the analysis scripts as follow (Listing 3)

```

# create container and mount results
cp -r scripts/ results/
docker run -it -v $PWD/results/:/results \
--name validate_twostage $IMAGE_NAME bash

# run analysis
python3 scripts/analysis.py -b /results \
-c scripts/asan.queue.json -r 0 -d /results/log/0
python3 scripts/analysis.py -b /results \
-c scripts/asan.crash.json -r 0 -d /results/log/0

# plot the results,
python3 scripts/print_result.py \
        -b /results/log/ -r 0 -t all

...
```

Listing 3: Evaluation Results Analysis

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): FF_{AFL} and FF_{AFL++} should achieve better coverage than their origin prototype AFL and AFL++. This is proven by experiment and the original results can be found in Table 5 in the paper.

(C2): FF_{AFL} and FF_{AFL++} should find more bugs than the direct competitors (AFL and AFL++). This is proven by the experiment and the original results are available in Table 7 and Figure 5-b.

A.4.2 Experiments

In the analysis step in subsection A.3, a coverage report along with bug reports are generated and can clearly demonstrate our claims C1 and C2.

In the paper we conduct 10 rounds of evaluation to reduce the possible variance, however, due to the time limitation, we suggest reducing the round to run, and 3 rounds might be sufficient. In that case the artifact evaluation can be finished in 3 days given a server equipped with 2 Xeon Gold 5218, 64GB memory and 1TB disk.

Besides, the bug report only consider the callstack and ASan bug type, which might still have lots of duplications, therefore in the paper we manually compare some stack traces to get the actual number of UNIQUE bugs. But the report itself is sufficient enough to support our claim C2.

A.5 Notes on Reusability

To make FISHFUZZ more usable, we develop an all-in-one wrapper that automated all the compilation steps for both FF_{AFL} and FF_{AFL++}. Users can refer to the manual in the README.

Besides, we're also working on the FISHFUZZ fuzzbench integration for better evaluating the capability of FISHFUZZ. Given that lots of fuzzbench targets didn't support LTO mode, we're working on implementing a non-LTO mode FISHFUZZ for the intergration and we're keeping pushing it forward. An experimental fuzzbench configuration file can be found in paper/fuzzbench folder.

A.6 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/.