DAFL: Directed Grey-box Fuzzing guided by Data Dependency

Tae Eun Kim, *KAIST;* Jaeseung Choi, *Sogang University;*
Kihong Heo and Sang Kil Cha, *KAIST*

https://www.usenix.org/conference/usenixsecurity23/presentation/kim-tae-eun

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
DAFL: Directed Grey-box Fuzzing Guided by Data Dependency

Tae Eun Kim
KAIST

Jaeseung Choi
Sogang University

Kihong Heo
KAIST

Sang Kil Cha
KAIST

A Artifact Appendix

This artifact appendix is a self-contained document which describes a roadmap for the evaluation of DAFL.

A.1 Abstract

DAFL is a directed grey-box fuzzer that leverages data dependency to guide the fuzzing process. DAFL's artifact provides a framework to run DAFL as well as the baseline fuzzers. The framework comprises an environment provided as a Docker image to run the fuzzers, the source code of DAFL, and the scripts to run the fuzzers and evaluate the results. This document describes how to set up the framework and replicate the experiment conducted in our paper.

A.2 Description & Requirements

In this section, we describe how to obtain our artifact, along with the hardware and software requirements to run our artifact.

A.2.1 Security, privacy, and ethical concerns

None

A.2.2 How to access

Our artifact is composed of two components: the Docker image and the framework to build and utilize it. The Docker image provides the environment to run individual fuzzing sessions by supporting all the necessary tools and dependencies. The framework builds the Docker image, orchestrates the fuzzing experiments, and evaluates the results.

The framework is accessible via a Zenodo link (`https://zenodo.org/record/8219904`). You can also download the same framework from the GitHub repository (`https://github.com/prosyslab/DAFL-artifact`). For the Docker image, we provide a pre-built Docker image via Docker Hub `https://hub.docker.com/r/prosyslab/dafl-artifact`. Nonetheless, you can also build the Docker image from scratch using the Dockerfile provided in the framework.

A.2.3 Hardware dependencies

We ran the experiment on the machines equipped with Intel(R) Xeon(R) Gold 6226R CPU (2.90GHz) with 64 cores and 192 GB of RAM. Each fuzzing session was run on a Docker container assigned with a single CPU core and 4GB of memory. As we repeated the experiment 40 times, each fuzzing session was run in parallel, utilizing 40 CPU cores at a time.

It is possible to run the experiment on a machine with fewer CPU cores and smaller RAM, but be sure to assign enough resources to each fuzzing session (e.g., 1 CPU core and 4GB of RAM for each fuzzing session).

The disk space requirement will vary depending on the volume of the experiment. However, we recommend at least 70 GB of disk space, since the provided Docker image alone is 25 GB and our main experiment results in 45 GB of data.

A.2.4 Software dependencies

Ubuntu 20.04, Docker, and Python 3.8 are required to run the artifact. The required Python dependencies can be installed by running the following command in the `DAFL-artifact` directory.

```
yes | pip3 install -r requirements.txt
```

A.2.5 Benchmarks

As described in our paper (Section 5.1), we used 41 vulnerabilities from the Beacon [1] paper.

A.3 Set-up

In this section, we describe how to set up the artifact for DAFL.

A.3.1 Installation

The following is the steps to install the artifact for DAFL.

1) Download the file `DAFL-artifact.tar.gz` from the provided Zenodo link. This file is the aforementioned

framework where you can build the Docker image, orchestrate the fuzzing experiments, and evaluate the results.

2) Extract the file to a directory of your choice.

```
tar -zxvf DAFL-artifact.tar.gz
```

3) Obtain the Docker image by:

1) pulling from Docker Hub

```
docker pull prosyslab/dafl-artifact
```

2) or building from scratch

```
docker build -t prosyslab/dafl-artifact -f Dockerfile .
```

This Docker image is the environment to run individual fuzzing sessions.

A.3.2 Basic Test

For a single fuzzing experiment, run the following command in the `DAFL-artifact` directory:

```
python3 scripts/reproduce.py run [target] [time budget] [iterations] "[list of fuzzers]"
```

where `[target]` is the name of the target, `[time budget]` is the time budget in seconds, `[iterations]` is the number of fuzzing iterations, and `[list of fuzzers]` is the list of fuzzers to run.

To run a simple functionality test, we recommend using the target `lrzip-ed51e14-2018-11496` with 60 seconds of time budget and 10 fuzzing iterations. For the list of fuzzers, use `"AFL AFLGo WindRanger Beacon DAFL"`. The command will look like the following:

```
python3 scripts/reproduce.py run lrzip-ed51e14-2018-11496 60 10 "AFL AFLGo WindRanger Beacon DAFL"
```

If set up was successful, a CSV file will be generated in the output directory with the name `output/lrzip-ed51e14-2018-11496-60sec-10iters`. This CSV file contains the results of the experiment, which is the median TTE of each fuzzer. In case of running 10 fuzzing sessions in parallel, this small experiment will take about 10 minutes.

A.4 Evaluation workflow

This section describes the operational steps and experiments which must be performed to evaluate DAFL's artifact.

A.4.1 Major Claims

Our paper makes the following claims:

(C1): DAFL is more effective in reproducing target crashes compared to the baseline fuzzers. This is proven by the experiment (E1) described in Section 5.2 of our paper whose results are illustrated in Table 2 and Figure 5.

(C2): Thin slicing shows better fuzzing performance compared to the naive approach. This is proven by the experiment (E2) described in Section 5.3 of our paper whose results are illustrated in Figure 7.

(C3): Two major components of DAFL, selective coverage instrumentation and semantic relevance scoring, both contribute to the fuzzing performance. This is proven by the experiment (E3) described in Section 5.4 of our paper whose results are illustrated in Figure 8.

A.4.2 Experiments

We have used a vast amount of resources to conduct the experiments for our paper. For example, we ran a 24-hour fuzzing session with 6 fuzzers on 41 targets, each repeated 40 times for the main experiment (E1) described in Section 5.2 of our paper. If run on a single machine capable of running 40 fuzzing sessions in parallel, this experiment would take 246 days of fuzzing time. We believe that it is not an easy task to replicate the experiments in our paper at a full scale.

Thus, apart from the instructions to exactly replicate our paper's experiments, we additionally provide instructions to run a scaled down version for each of the experiments to enable a feasible evaluation. The scaled down version of the experiments comprises fewer fuzzers, fewer targets, fewer iterations, and shorter time limit. For example, it excludes the targets where all the fuzzers failed to produce a median TTE within 24 hours. It also early terminates the fuzzing session based on the previously observed median TTE of each target. We also provide a minimal version of each experiment, which runs a small subset of targets. For more details on the scaled down version and the minimal version, please refer to the `README` file.

We believe that the scaled down version of the experiments is sufficient to validate the claims made in our paper. However, please note that the results of the alternative versions of the experiments are more prone to fluctuations due to the reduced number of iterations.

The expected time for each of the experiments is calculated under the assumption of running the experiment on a machine where 40 fuzzing sessions can be run in parallel. Each experiment results in a CSV file which contains the median TTE of each fuzzer for each target and a bar plot in the same format as in the paper.

(E1): [Effectiveness of DAFL] [246 compute-day + 70GB disk]: This is the main experiment described in Section

5.2 of our paper, which compares the effectiveness of DAFL with the baseline fuzzers.
How to: In the `DAFL-artifact` directory, run the following command:

```
python3 scripts/reproduce.py run tbl2 86400 40
```

Results: The result is in the form of a CSV file, which is located in the output directory (refer to the `README` file for the exact location of this file). This CSV file contains the median TTE of each fuzzer for each target. Additionally, a bar plot will be generated from the CSV file.

(E1: scaled down): [8 compute-days + 30GB disk]: The scaled down version of (E1).
How to: In the `DAFL-artifact` directory, run the following command:

```
python3 scripts/reproduce.py run tbl2-scaled 86400 10
```

Results: Same as (E1), with fewer targets and fuzzers.

(E2): [Effectiveness of thin slicing] [93 compute-day + 60GB disk]: This is the experiment described in Section 5.3 of our paper, which compares the effectiveness of thin and naive slicing approaches.
How to: In the `DAFL-artifact` directory, run the following command:

```
python3 scripts/reproduce.py run fig7 86400 40
```

Results: In same format as in (E1).

(E2: scaled down): [6 (or 2) compute-days + 30GB disk]: The scaled down version of (E2).
How to: In the `DAFL-artifact` directory, run the following command:

```
python3 scripts/reproduce.py run fig7-scaled 86400 10
```

If you have already run (E1: scaled down), the results of AFL and DAFL will be automatically reused as long as you have the results of (E1: scaled down) under the expected output directory, `output/tbl2-scaled-86400sec-10iters`. Thus, the expected runtime will be reduced to 1 compute-day.
Results: Same as (E2), with fewer targets.

(E3): [Effectiveness of DAFL's components] [124 compute-day + 60GB disk]: This is the experiment described in Section 5.4 of our paper, which evaluated the effectiveness of DAFL's components.
How to: In the `DAFL-artifact` directory, run the following command:

```
python3 scripts/reproduce.py run fig8 86400 40
```

Results: In same format as in (E1).

(E3: scaled down): [8 (or 4) compute-days + 30GB disk]: The scaled down version of (E3).
How to: In the `DAFL-artifact` directory, run the following command:

```
python3 scripts/reproduce.py run fig8-scaled 86400 10
```

As so in (E2: scaled down), if you have already run (E1: scaled down), the results of AFL and DAFL will be automatically reused as long as you have the results of (E1: scaled down) under the expected output directory, `output/tbl2-scaled-86400sec-10iters`. Thus, the expected runtime will be reduced to 2 compute-days.

Results: Same as (E3), with fewer targets.

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/.

References

[1] Heqing Huang, Yiyuan Guo, Qingkai Shi, Peisen Yao, Rongxin Wu, and Charles Zhang. Beacon: Directed grey-box fuzzing with provable path pruning. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 36–50, 2022.