autofz: Automated Fuzzer Composition at Runtime

Yu-Fu Fu, Jaehyuk Lee, and Taesoo Kim, *Georgia Institute of Technology*

https://www.usenix.org/conference/usenixsecurity23/presentation/fu-yu-fu

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: autofz: Automated Fuzzer Composition at Runtime

Yu-Fu Fu Jaehyuk Lee Taesoo Kim

Georgia Institute of Technology

A Artifact Appendix

A.1 Abstract

autofz's artifact contains the source code and all the benchmarks used in the evaluation section of the paper. This artifact appendix is used to outline the steps to retrieve the artifact and how to use it to reproduce the experiments. Furthermore, we provide the instructions to extend the framework (i.e. add ad new fuzzer or a new benchmark to autofz).

A.2 Description & Requirements

The artifact contains the following components.

1. autofz source code
2. A pre-built docker image containing autofz, fuzzers, and benchmarks.
3. A VM image which includes all the necessary changes to the host environment and can be used to launch the aforementioned docker image.

A.2.1 Security, privacy, and ethical concerns

During fuzzing, we modify some kernel parameters which docker shares with the host. For example, we enable core dump and disable ASLR for the whole system. Therefore, we recommend that running autofz inside a VM.

A.2.2 How to access

1. Source code https://github.com/sslab-gatech/autofz
2. Source code with commit hash https://github.com/sslab-gatech/autofz/tree/b9a795dda252aa37406d593434b710b0fbedd177
3. Docker image: https://hub.docker.com/r/fuyu0425/autofz with SHA256 digest f39fb70af5db and tag v1.0.1.
4. VM image: https://doi.org/10.5281/zenodo.7865366

A.2.3 Hardware dependencies

During the evaluation, we use a cluster of Ubuntu 20.04 machines equipped with AMD Ryzen 9 3900 (12C/24T), 32 GB RAM, and 512 GB SSD disk space. To use the provided docker image or VM image, 30 GB disk space is required.

A.2.4 Software dependencies

To use the docker image, a working Docker/Podman under Linux is required. Alternatively, to use the VM image, VirtualBox/VMware is required.

A.2.5 Benchmarks

All benchmarks required for evaluation are already in the docker image.

A.3 Set-up

We provide a detailed set-up process in README.md in the provided GitHub repository. However, building all fuzzers and benchmarks will takes a lot of time and resource. Therefore, we recommend using either the pre-built docker image or the VM image (preferred).

A.3.1 cgroup v2 downgrade

autofz uses cgroup v1; therefore, a manual downgrade from v2 to v1 might be required in newer operating systems. This can be done by adding "systemd.unified_cgroup_hierarchy=0" to the kernel command line (e.g. via "/etc/default/grub").

A.3.2 Basic Test

To make sure autofz is installed successfully, type the following command:

```
autofz --help
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): We demonstrate that autofz can achieve better coverage against different target binaries compared with individual fuzzers (Figure 3 in the paper). The result is supported by E1.

(C2): We demonstrate that `autofz` can achieve better coverage against different target binaries compared with other collaborative fuzzing techniques EnFuzz and Cupid (Figure 7 in the paper). The result is supported by E2.

A.4.2 Experiments

Setup. To execute the experiments, we need to pull docker images and launch a docker container by the following commands.

```
  docker pull fuyu0425/autofz:v1.0.1
  docker tag fuyu0425/autofz:v1.0.1 autofz
```

```
  docker run --rm --privileged -it autofz
/bin/bash.
```

Note that, the result is not preserved after exiting the container. To preserve the fuzzing output, we need to mount a docker volume.

```
  docker run --rm --privileged -v
$PWD:/work/autofz -w /work/autofz -it
autofz /bin/bash.
```

After entering the docker, we need to tune the necessary kernel parameters and create a cgroup for `autofz`; we pack all commands in a script `/init.sh` and can be executed by the following command. Note the security concern mentioned in §A.2.1.

```
  sudo /init.sh
```

More detail is in the *running* section of `README.md`.

(E1): [`autofz` v.s. individual fuzzers] [32000 compute-hours + 200 GB disk]: Generate the 24-hour fuzzing output of `autofz` and individual fuzzers on 12 benchmark programs for 10 repetitions for Figure 3 in the paper.

How to: Use `autofz` with different command line arguments to run all the fuzzing. `README.md` in the repository has more information about the arguments.

Execution: To run `autofz` on a target (e.g. `exiv2`), use the following command:

```
  autofz -o output-exiv2-autofz -t exiv2
 -T 24h -f all
```

To run a individual fuzzer (e.g. AFL) on a target (e.g. `exiv2`), use the following command:

```
  autofz -o output-exiv2-afl -t exiv2 -T
  24h -f afl --focus-one afl
```

Output directory specified by `-o` needs to be different for each fuzzing repetition.

Results: For each fuzzing run, `autofz` will generate a log file in JSON format, which includes all the coverage and the number unique bugs information.

Additionally, there is a directory called `eval` in the fuzzer output directory. The directory stores the results of crash deduplication and ASAN output of each crash.

(E2): [`autofz` v.s. EnFuzz/Cupid] [7680 compute-hours + 200 GB disk]: Generate the 24-CPU-hour fuzzing output of `autofz`-10, `autofz`-6, Cupid-4, and EnFuzz-6 on 8 benchmark programs for 10 repetitions for Figure 7 in the paper.

Execution: To run `autofz`-10 on a target (e.g. `exiv2`), use the following command:

```
  autofz -o output-exiv2-autofz10 -t
  exiv2 -T 24h -f all -j10 --parallel
```

To run `autofz`-6 on a target (e.g. `exiv2`), use the following command:

```
  autofz -o output-exiv2-autofz6
 -t exiv2 -T 24h -f afl fairfuzz
 qsym aflfast lafintel radamsa -j6
 --parallel
```

To run Cupid-4 (EnFuzz-Q) on a target (e.g. `exiv2`), use the following command:

```
  autofz -o output-exiv2-cupid4 -t exiv2
 -T 24h -f afl fairfuzz qsym aflfast
 --enfuzz 300 -j4 --parallel
```

Results: Note that, in the paper, we draw the graph based on **CPU hours**. Therefore, if we use 10 CPU cores (by specifying `-j 10`), only the first 2.4 hours is draw on the graph.

Because both experiments take enormous resource to replicate, we recommend choosing only a subset of benchmarks. Please note that fuzzing is an inherently a random process; therefore, the reproduced result might not be the same as we have reported in the paper. To alleviate this problem, we recommend increasing the fuzzing repetition (e.g. 10 times as we did) and similar results are expected.

A.4.3 Inspect log files of `autofz`

The log file of `autofz` is in JSON format and can be easily parsed by standard libraries in most programming languages. To inspect the log file (e.g. exiv2.json), we recommend using a tool called jq (https://github.com/stedolan/jq), which can be installed by the package manager in most Linux distributions. We already installed it in both the docker image and the VM image.

There are many fields in the log file. One of them is “log”, which can be retrieved by the following command.

```
  jq .log exiv2.json
```

The output is an array and each element of the array contains the coverage (“bitmap” field) and unique bugs information and the timestamp for that record. By default, a new log entry is appended for every 60 seconds.

To get the results based on rounds, we can use the following commands.

```
  jq .round exiv2.json
```

The output is also an array and each element is the result of one round. Each element records information for different phases in one round (e.g. the coverage before/after preparation/focus phases, resource allocation metadata and current difference threshold θ.)

In the provided VM, we provided one of the fuzzing log with the path

/home/autofz/output_exiv2/exiv2.json

A.4.4 Plotting the figures

We also include the scripts to draw the figures used in the paper.

```
  autofz-draw -o output-draw -t exiv2 -d
exp -T 24h -pdf
```

Above commands is used to draw figure 3 in the paper but only for exiv2.

We have more detailed explanation of each argument in the provided repository.

Note that for timeout parameter -T, it specifies **CPU Time**; therefore, for Figure 7, -T 3h is enough if you use 10 CPU cores.

A.5 Notes on Reusability

We have included instructions to extend autofz (add new fuzzers or new benchmarks) in the provided repository.

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/.