Stateful Greybox Fuzzing

Jinsheng Ba, *National University of Singapore;* Marcel Böhme, *Monash University and MPI-SP;* Zahra Mirzamomen, *Monash University;* Abhik Roychoudhury, *National University of Singapore*

https://www.usenix.org/conference/usenixsecurity22/presentation/ba

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

Our artifact is a pure software for fuzzing protocol implementations. It has no hardware requirement and very few software dependencies (Ubuntu Linux system, Python, and Docker). All of our approaches described in our paper are implemented in this artifact. The major claim is that our artifact is able to cover more of the state space. Without any manual intervention, the results can be found in artifact's execution log. We also provide scripts that may be used to reproduce the results.

A.2 Artifact check-list (meta-information)

- **Algorithm:** A new state approximation method for fuzzing protocol implementation.
- **Program:** We used the benchmark FuzzBench (https://github.com/bajinsheng/SGFuzz_Fuzzbench)
- **Compilation:** Clang >= 6.0
- **Run-time environment:** Ubuntu 20.04
- **Metrics:** State transition coverage, branch coverage
- **How much disk space required (approximately)?:** 20GB
- **How much time is needed to prepare workflow (approximately)?:** 1 hour
- **How much time is needed to complete experiments (approximately)?:** 2 days
- **Publicly available (explicitly provide evolving version reference)?:** https://github.com/bajinsheng/SGFuzz
- **Code licenses (if publicly available)?:** Apache License 2.0
- **Archived (explicitly provide DOI or stable reference)?:** Source code: https://github.com/bajinsheng/SGFuzz/tree/8f45141 Experimental data: https://zenodo.org/record/5555955

A.3 Description

A.3.1 How to access

git clone https://github.com/bajinsheng/SGFuzz
cd SGFuzz
git checkout 8f45141

A.3.2 Hardware dependencies

N/A

A.3.3 Software dependencies

- Ubuntu (>=16.04)
- Docker (>=20.10.7)
- Python (>=3.8)

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

We provide a script to compile and configure our artifact, and only two steps are needed to setup:

1. git clone https://github.com/bajinsheng/SGFuzz
2. cd SGFuzz && ./build.sh

A.5 Evaluation and expected results

We explain our major claims, corresponding results in the paper, and the detailed steps to reproduce them.

A.5.1 Key Claims

We made these key claims in our paper:

1. **State Transition Coverage**. SGFUZZ is able to cover more of the state space. SGFuzz significantly outperform LibFuzzer on state transition coverage in 23 hours.
2. **Branch Coverage**. SGFuzz slightly outperform LibFuzer on branch coverage in 23 hours.
3. **State Identification Effectiveness**. Changed variables with name constants are accurate approximation of state variables.
4. **Prevalence of Stateful Bugs**. Stateful bugs are prevalent in protocol implementations.
5. **Prevalence of State Variables**. Named constants are widely used for state variables.

A.5.2 Key Results

We have these key results in our paper to support our claims.

1. **State Transition Coverage**. SGFuzz covers 33x more sequences of state transitions than LibFuzzer in 23 hours on average. (Research Question 1)
2. **Branch Coverage**. SGFuzz achieves 2.20% more branch coverage than LibFuzzer in 23 hours on average. (Research Question 2)

3. **State Identification Effectiveness**. An average 99.5% of nodes in the State Transition Tree (the data structure constructed in SGFuzz for tracing states) in 23 hours are referring to values of actual state variables. (Research Question 3)

4. **Prevalence of Stateful Bugs**. Every four in five bugs that are reported in OSS-Fuzz for protocol implementations among our subjects are stateful. (Appendix Section 3)

5. **Prevalence of State Variables**. Top-50 most widely used open-source protocol implementations define state variables with named constants. (Appendix Section 4)

A.5.3 Prerequistities to reproduce

We have integrated our code into the FuzzBench framework, so only the dependencies of FuzzBench are necessary to evaluate our code. Please refer to the following commands to install and configure the FuzzBench.

```
git clone https://github.com/bajinsheng/
 → SGFuzz_Fuzzbench
cd SGFuzz_Fuzzbench
git submodule update --init
sudo apt-get install build-essential
 → python3.8-dev python3.8-venv
make install-dependencies
source .venv/bin/activate
```

More information about the installation of Fuzzbench can be found at: https://google.github.io/fuzzbench/getting-started/prerequisites/.

Note that the FuzzBench framework depends on docker, so it is hard to run FuzzBench within docker.

A.5.4 Steps to reproduce

1. **State Transition Coverage**.

(1) `SGFuzz's results.` Executing this command in the root of SGFuzz_FuzzBench folder:

```
sudo make run-sfuzzer-h2o_h2o-fuzzer-http2
```

After prompting some building information (several minutes for the first time), the fuzzing status will be gradually shown in the terminal, like this:

```
#2 INITED cov: 641 ft: 642 corp: 1/12569b
    exec/s: 0 rss: 38Mb states: 13 leaves: 2
#3 NEW    cov: 649 ft: 659 corp: 2/24Kb
    lim: 12569 exec/s: 0 rss: 39Mb states: 13
    leaves: 2 L: 12569/12569 MS: 1 CopyPart-
```

The number of *leaves* represents the number of unique state transition sequences observed in the current fuzzing campaign.

(2) `LibFuzzer's results.` As a reference, the results of LibFuzzer have to be got manually, because of the lack of *leaves* information. We copy the generated corpus from LibFuzzer to SGFuzz, and observe the *leaves* information.

Starting an interactive docker shell for LibFuzzer:

```
sudo make debug-libfuzzer-h2o_h2o-fuzzer-http2
```

In the docker container, running the LibFuzzer:

```
$ROOT_DIR/docker/benchmark-runner/startup-runner.sh
```

After 23 hours, in the docker container, typing 'CTRL+C' to stop LibFuzzer. In the host, copying the generated corpus from docker to host:

```
sudo docker cp docker-id-libfuzzer:/out/corpus
 → .
```

The *docker-id-libfuzzer* needs to be replaced by the actual hash id of the docker container. Then starting a docker container for SGFuzz:

```
sudo make debug-sfuzzer-h2o_h2o-fuzzer-http2
```

In the host, copying the corpus to the new docker container:

```
sudo docker cp corpus docker-id-sgfuzz:/out
```

The *docker-id-sgfuzz* should be replaced by the SGFuzz's docker hash id as well. In the SGFuzz's docker container, running SGFuzz to observe the results:

```
./h2o-fuzzer-http2 corpus/
```

In the output, the line with the **INITED** represents the total number of state transition sequences observed in LibFuzzer's campaign:

```
#1137 INITED cov: 1456 ft: 5274 corp: 375/2703Kb
    exec/s: 12 rss: 340Mb states: 235 leaves: 47
```

(3) `Evaluation.` Comparing the number of *leaves* indicated in each fuzzing campaign. Note that our experiments were conducted in 23 hours, so we may notice a substantial gap in state transition coverage between SGFuzz and LibFuzzer after **several hours**, not a few minutes.

(4) `More subjects.` Changing *h2o_h2o-fuzzer-http2* to *curl_curl_fuzzer*, *mbedtls_fuzz_dtlsserver*, *gstreamer_gst-discoverer* in the commands and redo steps 1-3 to evaluate other subjects.

(5) `Variance.` Our results are based on average number across 20 runs. Beware of variance! Difference between the two highest- and lowest-coverage runs may be up to 50% because of the randomness in fuzzing.

2. **Branch Coverage**.

The same steps as the state transition coverage experiment. The only difference is that the branch coverage information can directly got from the output of LibFuzzer, so we directly run

```
sudo make run-libfuzzer-h2o_h2o-fuzzer-http2
```

instead of the step (2) in State Transition Coverage. The branch coverage information is indicated as number of *cov* in the output.

3. **State Identification Effectiveness**.

Please check the folder *RQ3_State_Iden_Effic* at https://zenodo.org/record/5555955, which includes all state variables and the variables that are included in the STT.

4. **Prevalence of Stateful Bugs**.

Please check the folder *A3_Bug_Preva* at https://zenodo.org/record/5555955, which includes all state variables and the variables that are included in the STT.

5. **Prevalence of State Variables**.

Please check the folder *A4_Top50* at https://zenodo.org/record/5555955, which includes all state variables and the variables that are included in the STT.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220119.