ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

USENIX'23 Artifact Appendix: PolyFuzz: Holistic Greybox Fuzzing of Multi-Language Systems

Wen Li, Jinyang Ruan, Guangbei Yi, Long Cheng, Xiapu Luo, and Haipeng Cai

A Artifact Appendix

A.1 Abstract

This artifact contains a functional version of PolyFuzz and the necessary dataset for the evaluation. To facilitate the usage of this artifact, we have prepared a Docker image with the necessary components to execute the artifact and visualize the result. Artifact users can compare the results obtained from executing this artifact with those presented in our paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, or ethical concerns with using this artifact.

A.2.2 How to access

We provided three ways to access our artifact package:

1. DOI is provided through FigShare
https://doi.org/10.6084/m9.figshare.20022893.v1

2. An evolving version is maintained on GitHub
The GitHub repository is:
https://github.com/Daybreak2019/PolyFuzz.git
A specific tag is provided:
https://github.com/Daybreak2019/PolyFuzz/releases/tag/v6.0

3. A docker image is provided with PolyFuzz installed
docker pull daybreak2019/polyfuzz:v1.1

A.2.3 Hardware dependencies

The host machine may need at least 16GB memory and 256GB hard disk space.

A.2.4 Software dependencies

PolyFuzz is mainly developed and tested on LLVM 11.0, Soot 4.3.0, Python 3.8/9 (and Python3-dev), and OpenJDK 8/11. For ease of use of PolyFuzz, we have prepared a Docker image with all compilation and run-time dependencies installed.

Moreover, real-world benchmarks have their own particular/additional dependencies. Hence, to fully reproduce the results in the paper, users should install these dependencies successfully, which have been provided via relevant scripts in the PolyFuzz repository on GitHub.

More specifically, we note that the Docker image includes all the libraries/frameworks underlying PolyFuzz; thus it can be used for experimenting with other real-world subjects as well (i.e., saving the time/trouble for installing ubuntu, llvm, etc.) On the other hand, since our real-world subjects are sizable, including the complete compilation and run-time environment (e.g., all the third-party library dependencies) for all of them in the single Docker image would make it clumsy to deploy conveniently. Using a traditional virtual machine would aggregate this concern since they are even heavier. This is why we chose to include in the Docker image only the setup for the subjects in which any vulnerabilities were discovered by PolyFuzz at the paper submission time. Users can still use the scripts in the repository to set up the environments for other benchmarks; we have tested the scripts on our servers.

A.2.5 Benchmarks

For demonstration purposes, we use 5 multi-language benchmarks with vulnerabilities detected as concrete examples and have installed all of their dependencies in the Docker image. Specifically, the installed benchmarks include 4 Python-C benchmarks (i.e., Pillow, Libsmbios, Ultrajson, Bottleneck) and 1 Java-C benchmark (i.e., Jansi).

A.3 Set-up

A.3.1 Installation

- Step 1: Config AFL++

```
git clone https://github.com/Daybreak2019/PolyFuzz.git
sudo PolyFuzz/AFLplusplus/afl-system-config
```

- Step 2: Download the Docker image and run a Docker container based on the image

```
docker pull daybreak2019/polyfuzz:v1.1
docker run -it daybreak2019/polyfuzz:v1.1
```

- Step 3: Update PolyFuzz to the latest version and build the project within the container

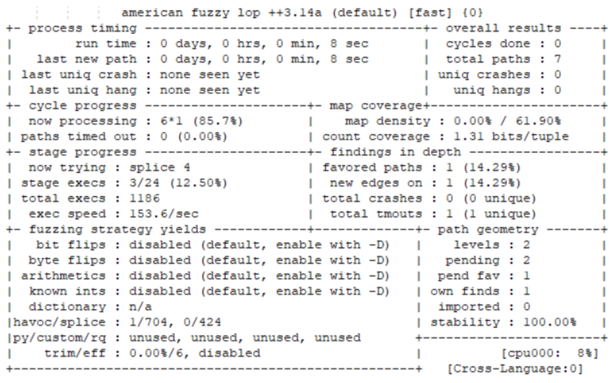
  ```
  cd /root/PolyFuzz
  git pull
  . build.sh
  ```

A.3.2 Basic Test

To validate whether the environment is ready, a simple test can be run as follows:

```
cd /root/PolyFuzz/langspec/python/tests/case4
./build.sh
./sasg_entry.sh
```

The results should be similar as shown in Figure 1. Then "CTRL + C" can be used to quit the fuzzing.



```
           american fuzzy lop ++3.14a (default) [fast] {0}
+- process timing ------------------------------------+- overall results ----+
|        run time : 0 days, 0 hrs, 0 min, 8 sec       |  cycles done : 0     |
|   last new path : 0 days, 0 hrs, 0 min, 8 sec       |  total paths : 7     |
| last uniq crash : none seen yet                     | uniq crashes : 0     |
|  last uniq hang : none seen yet                     |   uniq hangs : 0     |
+- cycle progress ---------------------+- map coverage+----------------------+
|  now processing : 6*1 (85.7%)        |    map density : 0.00% / 61.90%     |
| paths timed out : 0 (0.00%)          | count coverage : 1.31 bits/tuple    |
+- stage progress ---------------------+- findings in depth -----------------+
|  now trying : splice 4               | favored paths : 1 (14.29%)          |
| stage execs : 3/24 (12.50%)          |  new edges on : 1 (14.29%)          |
| total execs : 1186                   | total crashes : 0 (0 unique)        |
|  exec speed : 153.6/sec              |  total tmouts : 1 (1 unique)        |
+- fuzzing strategy yields ------------+-------------+- path geometry -------+
|   bit flips : disabled (default, enable with -D)   |    levels : 2         |
|  byte flips : disabled (default, enable with -D)   |   pending : 2         |
| arithmetics : disabled (default, enable with -D)   |  pend fav : 1         |
|  known ints : disabled (default, enable with -D)   | own finds : 1         |
|  dictionary : n/a                                  |  imported : 0         |
|havoc/splice : 1/704, 0/424                         | stability : 100.00%   |
|py/custom/rq : unused, unused, unused, unused       +-----------------------+
|    trim/eff : 0.00%/6, disabled                    |          [cpu000:  8%]
+----------------------------------------------------+   [Cross-Language:0]
```

Figure 1: A basic test for PolyFuzz

A.4 Evaluation workflow

We provided scripts for automating all the experiments during the evaluation. Specifically in this artifact, we setup the environment for the experiments on 5 multi-language benchmarks, including 4 Python-C benchmarks (i.e., Pillow, Libsmbios, Ultrajson, Bottleneck) and 1 Java-C benchmark (i.e., Jansi). For each benchmark, the expected fuzzing time is 24 hours.

A.4.1 Major Claims

For the 5 multi-language benchmarks, PolyFuzz should be able to reproduce the vulnerabilities reported in Table 10 of the paper.

A.4.2 Experiments

(E1): [Experiment on Pillow] [30 human-minutes + 24 compute-hour + 128GB disk]: users should rebuild Pillow in the container, then start fuzzing for 24 hours.
How to: Rebuild Pillow and run PolyFuzz on it.
Preparation: None

Execution: Run commands as follows:
1. Build Pillow

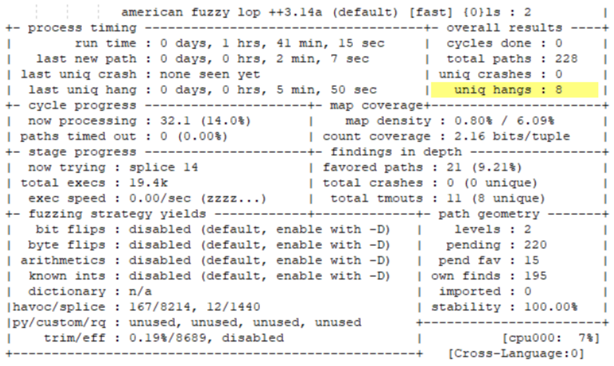
```
cd /root/PolyFuzz/benchmarks/script/multi-benches/Pillow
./build.sh
```

2. Run fuzzing on Pillow

```
cd drivers/fig_process
./sasg_entry.sh
```

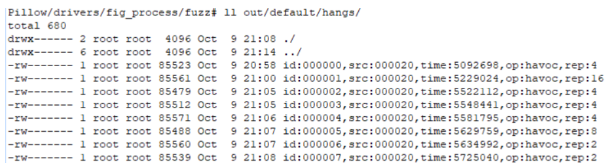
Results: PolyFuzz should report crashes/hangs in the end. The corresponding test cases would be generated under the directory fuzz/out/default/crashes and fuzz/out/default/hangs. As an example, Figure 2 shows the fuzzing results of Pillow, which is a snapshot of the fuzzing at 1 hour 41 mins. To validate whether these *hangs* are true positives, we can use the following commands to run the tests one by one:

- 1. entry the *fuzz* directory
  `cd /root/PolyFuzz/benchmarks/script/multi-benches/Pillow/drivers/fig_process/fuzz`
- 2. list the tests of *hangs* (results as shown in Figure 3.)
  `ll out/default/hangs/`
- 3. run a single test with the driver (results as shown in Figure 4.)
  `python ../fig_process.py out/default/hangs/id:...`



```
           american fuzzy lop ++3.14a (default) [fast] {0}ls : 2               |
+- process timing ------------------------------------+- overall results ----+
|        run time : 0 days, 1 hrs, 41 min, 15 sec     |  cycles done : 0     |
|   last new path : 0 days, 0 hrs, 2 min, 7 sec       |  total paths : 228   |
| last uniq crash : none seen yet                     | uniq crashes : 0     |
|  last uniq hang : 0 days, 0 hrs, 5 min, 50 sec      |   uniq hangs : 8     |
+- cycle progress ---------------------+- map coverage+----------------------+
|  now processing : 32.1 (14.0%)       |    map density : 0.80% / 6.09%      |
| paths timed out : 0 (0.00%)          | count coverage : 2.16 bits/tuple    |
+- stage progress ---------------------+- findings in depth -----------------+
|  now trying : splice 14              | favored paths : 21 (9.21%)          |
| total execs : 19.4k                  | total crashes : 0 (0 unique)        |
|  exec speed : 0.00/sec (zzzz...)     |  total tmouts : 11 (8 unique)       |
+- fuzzing strategy yields ------------+-------------+- path geometry -------+
|   bit flips : disabled (default, enable with -D)   |    levels : 2         |
|  byte flips : disabled (default, enable with -D)   |   pending : 220       |
| arithmetics : disabled (default, enable with -D)   |  pend fav : 15        |
|  known ints : disabled (default, enable with -D)   | own finds : 195       |
|  dictionary : n/a                                  |  imported : 0         |
|havoc/splice : 167/8214, 12/1440                    | stability : 100.00%   |
|py/custom/rq : unused, unused, unused, unused       +-----------------------+
|    trim/eff : 0.19%/8689, disabled                 |          [cpu000:  7%]
+----------------------------------------------------+   [Cross-Language:0]
```

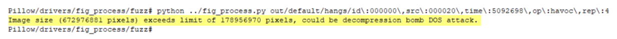
Figure 2: Example of fuzzing result on Pillow



```
Pillow/drivers/fig_process/fuzz# ll out/default/hangs/
total 680
drwx------ 2 root root  4096 Oct  9 21:08 ./
drwx------ 6 root root  4096 Oct  9 21:14 ../
-rw------- 1 root root 85523 Oct  9 20:58 id:000000,src:000020,time:5092698,op:havoc,rep:4
-rw------- 1 root root 85561 Oct  9 21:00 id:000001,src:000020,time:5229024,op:havoc,rep:16
-rw------- 1 root root 85479 Oct  9 21:05 id:000002,src:000020,time:5522112,op:havoc,rep:4
-rw------- 1 root root 85512 Oct  9 21:05 id:000003,src:000020,time:5548441,op:havoc,rep:4
-rw------- 1 root root 85571 Oct  9 21:06 id:000004,src:000020,time:5581795,op:havoc,rep:4
-rw------- 1 root root 85488 Oct  9 21:07 id:000005,src:000020,time:5629759,op:havoc,rep:8
-rw------- 1 root root 85560 Oct  9 21:07 id:000006,src:000020,time:5634992,op:havoc,rep:2
-rw------- 1 root root 85539 Oct  9 21:08 id:000007,src:000020,time:5725040,op:havoc,rep:2
```

Figure 3: *Hang* tests of fuzzing result on Pillow

(E2): [Experiment on Libsmbios] [30 human-minutes + 24 compute-hour + 128GB disk]: users should rebuild Libsmbios in the container, then start fuzzing for 24 hours.
How to: Rebuild Libsmbios and run PolyFuzz on it.



```
Pillow/drivers/fig_process/fuzz# python ../fig_process.py out/default/hangs/id\:000000\,src\:000020\,time\:5092698\,op\:havoc\,rep\:4
Image size (672976881 pixels) exceeds limit of 178956970 pixels, could be decompression bomb DOS attack.
Pillow/drivers/fig_process/fuzz#
```

Figure 4: A single test result of Pillow with driver fig_process.py

Preparation: None
Execution: Run commands as follows:

- Build Libsmbios
  cd /root/PolyFuzz/benchmarks/script/multi-benches/libsmbios
  ./build.sh
- Run fuzzing on Libsmbios
  cd drivers/op_mem
  ./sasg_entry.sh

Results: PolyFuzz should report crashes/hangs in the end.

(E3): [Experiment on Ultrajson] [30 human-minutes + 24 compute-hour + 128GB disk]: users should rebuild Ultrajson in the container, then start fuzzing for 24 hours.
How to: Rebuild Ultrajson and run PolyFuzz on it.
Preparation: None
Execution: Run commands as follows:

- Build Ultrajson
  cd /root/PolyFuzz/benchmarks/script/multi-benches/ultrajson
  ./build.sh
- Run fuzzing on Ultrajson
  cd drivers/encode
  ./sasg_entry.sh

Results: PolyFuzz should report crashes/hangs in the end.

(E4): [Experiment on Bottleneck] [30 human-minutes + 24 compute-hour + 128GB disk]: users should rebuild Bottleneck in the container, then start fuzzing for 24 hours.
How to: Rebuild Bottleneck and run PolyFuzz on it.
Preparation: None
Execution: Run commands as follows:

- Build Bottleneck
  cd /root/PolyFuzz/benchmarks/script/multi-benches/bottleneck
  ./build.sh
- Run fuzzing on Bottleneck
  cd drivers/random_shape2
  ./sasg_entry.sh

Results: PolyFuzz should report crashes/hangs in the end.

(E5): [Experiment on Jansi] [30 human-minutes + 24 compute-hour + 128GB disk]: users should rebuild Jansi in the container, then start fuzzing for 24 hours.
How to: Rebuild Jansi and run PolyFuzz on it.
Preparation: None
Execution: Run command as follows:

- Build Jansi
  cd /root/PolyFuzz/benchmarks/script/multi-benches/jansi
  ./build.sh
- Build fuzzing drivers of Jansi
  cd drivers
  ./build.sh
- Run fuzzing on Jansi
  cd OutStream
  ./sasg_entry.sh

Results: PolyFuzz should report crashes/hangs in the end.

A.5 Notes on Reusability

Considering the time cost of fuzzing experiments, in the artifact package, we only demonstrated 5 of the multi-language benchmarks used in our paper. However, for all of the benchmarks, we have provided similar scripts (under directory PolyFuzz/benchmarks/script) to the ones demonstrated in the appendix; users can follow the steps above to run PolyFuzz on other benchmarks.

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/`.