UNIFUZZ: A Holistic and Pragmatic Metrics-Driven Platform for Evaluating Fuzzers

Yuwei Li, Shouling Ji, Yuan Chen, Sizhuang Liang, Wei-Han Lee, Yueyao Chen, Chenyang Lyu
Chunming Wu, Raheem Beyah, Peng Cheng, Kangjie Lu, Ting Wang

USENIX Security 2021
A plethora of fuzzing works have emerged in both industry and academia.

GitHub has over 3,000 fuzzing related repos.

Dblp, a famous computer science bibliography website, contains more than 200 fuzzing related papers since 2010.
Open questions about fuzzing technique

- How do these fuzzers perform in practice?
- How to compare different fuzzers under a fair and comprehensive set of performance metrics?
- Which fuzzing primitives or techniques are promising and should be promoted?
The previous works cannot answer these questions

- Many existing works do not conduct appropriate and sufficient experiments to provide trustworthy results.
  - Insufficient repetitions, not using statistical test
  - Inconsistency of environments
The previous works cannot answer these questions

- Many existing works do not conduct appropriate and sufficient experiments to provide trustworthy results.
  - Insufficient repetitions, not using statistical test
  - Inconsistency of environments

- The evaluations of the existing fuzzers are often biased due to the lack of uniform benchmarks.
  - The choice varies widely.

- The existing metrics may not be suitable nor comprehensive for evaluating fuzzers.
  - Never consider “overhead”
Challenges for conducting comprehensive evaluations

- Challenge 1: usability issues of the existing fuzzers
- Challenge 2: lack of pragmatic real-world benchmarks
- Challenge 3: lack of proper and comprehensive performance metrics
Our Solution
UNIFUZZ: a holistic and pragmatic metrics-driven platform for evaluating fuzzers

Overview of UNIFUZZ
We conducted large-scale tests on the usability of the existing fuzzers.

- 15+ serious flaws
- 35+ usable fuzzers
  - Dockerfile
  - Detailed documents

Table 7: The fuzzers incorporated in UnFUZZ.

| Fuzzer         | Mutation/Generation | Directed/Coverage | Target |
|----------------|----------------------|-------------------|--------|
| AFL [70]       | M                    | C                 | S/B    |
| AFLFast [29]   | M                    | C                 | S/B    |
| AFLGo [28]     | M                    | D                 | S      |
| AFLPRIN [71]   | M                    | C                 | B      |
| AFLSmart [59]  | M                    | C                 | S/B    |
| Angora [39]    | M                    | C                 | S/B    |
| CodeAlchemist [39] | G                | n.a.              | B      |
| Driller [64]   | M                    | C                 | B      |
| Domato [34]    | G                    | n.a.              | B      |
| Dharma [11]    | G                    | n.a.              | B      |
| Eclipse [32]   | M                    | C                 | S      |
| FairFuzz [45]  | M                    | C                 | S      |
| Fuzzilli [19]  | M                    | C                 | S      |
| Grammarinator [41] | G              | n.a.              | B      |
| Honggfuzz [36] | M                    | C                 | S      |
| Jsfuzz [23]    | M                    | C                 | S      |
| jsfunfuzz [22] | G                    | n.a.              | B      |
| LearnAFL [68]  | M                    | C                 | S      |
| MoonLight [46] | n.a.                 | n.a.              | n.a.   |
| MOPT [48]      | M                    | C                 | S/B    |
| NAUTILUS [27]  | G+M                  | C                 | S      |
| NEUZZ [62]     | M                    | C                 | S      |
| NEZHA [57]     | M                    | C                 | S/L    |
| Orthrus [61]   | n.a.                 | n.a.              | n.a.   |
| Peach [12]     | G                    | n.a.              | B      |
| PTFuzz [71]    | M                    | C                 | S      |
| QSYM [69]      | M                    | C                 | B      |
| QuickFuzz [38] | G+M                  | n.a.              | B      |
| radamsa [13]   | M                    | C                 | B      |
| slowfuzz [58]  | M                    | n.a.              | L      |
| Supercon [66]  | G+M                  | C                 | S      |
| T-Fuzz [56]    | M                    | C                 | S      |
| VUZer [60]     | M                    | C                 | B      |
| VUZer64 [60]   | M                    | C                 | B      |
| zuf [43]       | M                    | n.a.              | B      |

1 S: source code, B: binary.
2 L: user needs to write libFuzzer code.
## Pragmatic benchmark suite

| Type   | Program                | Version | Arguments                                                                 |
|--------|------------------------|---------|---------------------------------------------------------------------------|
| Image  | exiv2                  | 0.26    | @ @                                                                      |
|        | gdk-pixbuf-pixdata (gdk)| gdk-pixbuf 2.31.1 | @ @ /dev/null                                                            |
|        | imginfo                | jasper 2.0.12 | -f @ @                                                                   |
|        | jhead                  | 3.00    | @ @                                                                      |
|        | tiffsplit              | libtiff 3.9.7 | @ @                                                                      |
| Audio  | lame                   | lame 3.99.5 | @ @ /dev/null                                                             |
|        | mp3gain                | 1.5.2-r2 | @ @                                                                      |
|        | wav2swf                | swftools 0.9.2 | -o /dev/null @@                                                          |
| Video  | ffmpeg                 | 4.0.1   | (-y @ @ -c:v \ mpeg4 -c:a copy -f \ mp4 /dev/null)                      |
|        | flvmeta                | 1.2.1   | @ @                                                                      |
|        | mp42aac                | Bento4 1.5.1-628 | @ @ /dev/null                                                             |
| Text   | cflow                  | 1.6     | @ @                                                                      |
|        | infotocap              | ncurses 6.1 | -o /dev/null @@                                                          |
|        | jq                     | 1.5     | . @ @                                                                    |
|        | mjujs                  | 1.0.2   | @ @                                                                      |
|        | pdftotext              | xpdf 4.00 | @ @ /dev/null                                                             |
|        | sqlite3                | 3.8.9   | (sidin)                                                                  |
| Binary | nm                     | binutils 5279478 | (-A -a -l -S -s \ - -special-syms \ - -synthetic \ - -with-symbol-versions \ -D @ @) |
|        | objdump                | binutils 2.28 | -S @ @                                                                   |
| Network| tcpdump                | 4.8.1 + libpcap 1.8.1 | -e -vv -nr @ @                                                        |

- **20 real-world benchmark programs**
  - 6 functionality types
  - 12+ vulnerability types

- **convenient offline results analysis**
  - bug triage
  - severity analysis
  - CVE matching
Comprehensive performance metrics

Six categories of performance metrics

- Quantity of unique bugs
  - Statistical test
- Quality of the bugs
  - Severity of the bugs, rareness of the bugs
- Speed of finding the bugs
- Stability of finding the bugs
- Coverage
- Overhead
Evaluations
We conducted large-scale evaluations on the state-of-the-art fuzzers.

- 8 state-of-the-art fuzzers: AFL, AFLFast, Angora, Honggfuzz, MOPT, QSYM, T-Fuzz, VUzzer64.
- large-scale: \textbf{200,000+} CPU hours
- 6 categories of performance metrics
Summary of interesting findings

- No fuzzer outperformed the others among all the benchmark programs.
  - Fuzzers may have preference over some specific programs.
- The synthetic benchmark programs may not be able to reflect a fuzzer’s performance on the real-world programs.
- A single metric may lead to unilateral conclusions.
- More factors can affect the fuzzing evaluation results than what we thought.
No fuzzer outperformed the others among all the benchmark programs.

- Fuzzers may have preference over some specific programs.

The synthetic benchmark programs may not be able to reflect a fuzzer’s performance on the real-world programs.

A single metric may lead to unilateral conclusions.

More factors can affect the fuzzing evaluation results than what we thought.
No fuzzer outperformed the others among all the benchmark programs.

Angora performed the best on **exiv2**.

QSYM performed the best on **tcpdump**.

The number of unique bugs found by each fuzzers.
Summary of interesting findings

- No fuzzer outperforms the others among all the benchmark programs.
  - Fuzzers have preference over some specific programs.

- The synthetic benchmark programs may not be able to reflect a fuzzer’s performance on the real-world programs.

- A single metric may lead to unilateral conclusions.

- More factors can affect the fuzzing evaluation results than what we thought.
Synthetic benchmark VS. Real-world benchmark

The #unique bugs on the real-world programs.

The #unique bugs on the synthetic programs (LAVA-M).

T-Fuzz and VUzzer64 had better performance on the synthetic benchmark programs than on the real-world benchmark programs.
No fuzzer outperforms the others among all the benchmark programs.

- Fuzzers have preference over some specific programs.

- The synthetic benchmark programs may not be able to reflect a fuzzer’s performance on the real-world programs.

- A single metric may lead to unilateral conclusions.

- More factors can affect the fuzzing evaluation results than what we thought.
A single metric may lead to unilateral conclusions.

For **#unique bugs** metric, QSYM performed the best on *jhead*.
For **#line coverage** metric, Angora performed the best on *jhead*. 

The **#unique bugs**

The **#line coverage**
Summary of interesting findings

- No fuzzer outperforms the others among all the benchmark programs.
  - Fuzzers have preference over some specific programs.
- The synthetic benchmark programs may not be able to reflect a fuzzer’s performance on the real-world programs.
- A single metric may lead to unilateral conclusions.
- More factors can affect the fuzzing evaluation results than what we thought.
Fuzzers usually have different instrumentation methods.

- Compile-time instrumentation, e.g., AFL, Angora.
- Dynamic binary instrumentation., e.g., VUzzer.

Thus, the same tested benchmark program are compiled into different binaries!

We found that Angora cannot find some bugs on the program *infotocap* due to its instrumentation method, not its capability in finding bugs.
Fuzzers usually have different instrumentation methods.

- Compile-time instrumentation, e.g., AFL, Angora.
- Dynamic binary instrumentation, e.g., VUzzer.

Thus, the same tested benchmark program are compiled into different binaries!

We found that Angora cannot find the bugs on the program *infotocap* due to its instrumentation method, not its capability in finding bugs.

Using *cross validation* when analyzing the crash samples, e.g., re-executing the crash samples with different compiled binaries to check whether these crash samples can be reproduced on all the binaries.
Different crash analysis tools are used in validating the bugs triggered by the crash samples.

Using different analysis tools may lead to different evaluation results, e.g., \#unique bugs.

Table 13: Validated crash samples by different tools.

| Bug Type                                      | Number | Rate  |
|----------------------------------------------|--------|-------|
| Neither ASAN or GDB can validate             | 40,122 | 12.2% |
| Only GDB can validate                        | 47,910 | 14.5% |
| Only ASAN can validate                       | 40,267 | 12.2% |
| Both ASAN and GDB can validate               | 201,558| 61.1% |
| Total                                        | 329,857| 100%  |

When using ASAN to validate the crashes, the result is that only Honggfuzz can discover bugs.

Figure 11: The number of unique bugs discovered on ffmpeg with GDB and ASan.
Different crash analysis tools are used in validating the bugs triggered by the crash samples. Using different analysis tools may lead to different evaluation results, e.g., \#unique bugs.

Using multiple analysis tools may mitigate these biases!

When using ASAN to validate the crashes, the result is that only Honggfuzz can discover bugs.

Table 13: The number of unique bugs discovered on ffmpeg with GDB and ASan.

| Bug Type                                    | AFL | AFLFast | Angora | Honggfuzz | MOPT  | QSYM  | T-Fuzz | VUzzer64 |
|---------------------------------------------|-----|---------|--------|-----------|-------|-------|--------|----------|
| Neither ASAN nor GDB                       | 129,564 | 61.1%      |
| Only GDB                                   | 129,564 | 61.1%      |
| Only ASAN                                  | 129,564 | 61.1%      |
| Both ASAN and GDB can validate             | 201,558 | 61.1%      |
| Total                                      | 329,857 | 100%       |
Conclusion
We proposed and implemented UNIFUZZ, a holistic, and pragmatic metrics-driven platform for evaluating fuzzers in a comprehensive and fair manner.

UNIFUZZ has incorporated 35 usable fuzzers, 20 real-world benchmark programs and 6 categories of performance metrics.

We conducted extensive evaluations on the 8 state-of-the-art fuzzers and got many interesting findings.

We have open sourced UNIFUZZ to facilitate the future fuzzing research and welcome the community contributions.
https://github.com/unifuzz/overview