Evaluating FixReverter with bug fix patterns and benchmarking with FuzzBench/RevBugBench

Daniel Gelencsér, Raashid Khan, Brian Arnesto Sitorus

Universiteit
Leiden
The Netherlands

Introduction

The Goal of Fuzzing

- Stress test application until it:
  - Crash
  - Weird Behaviour
  - Loses Resources

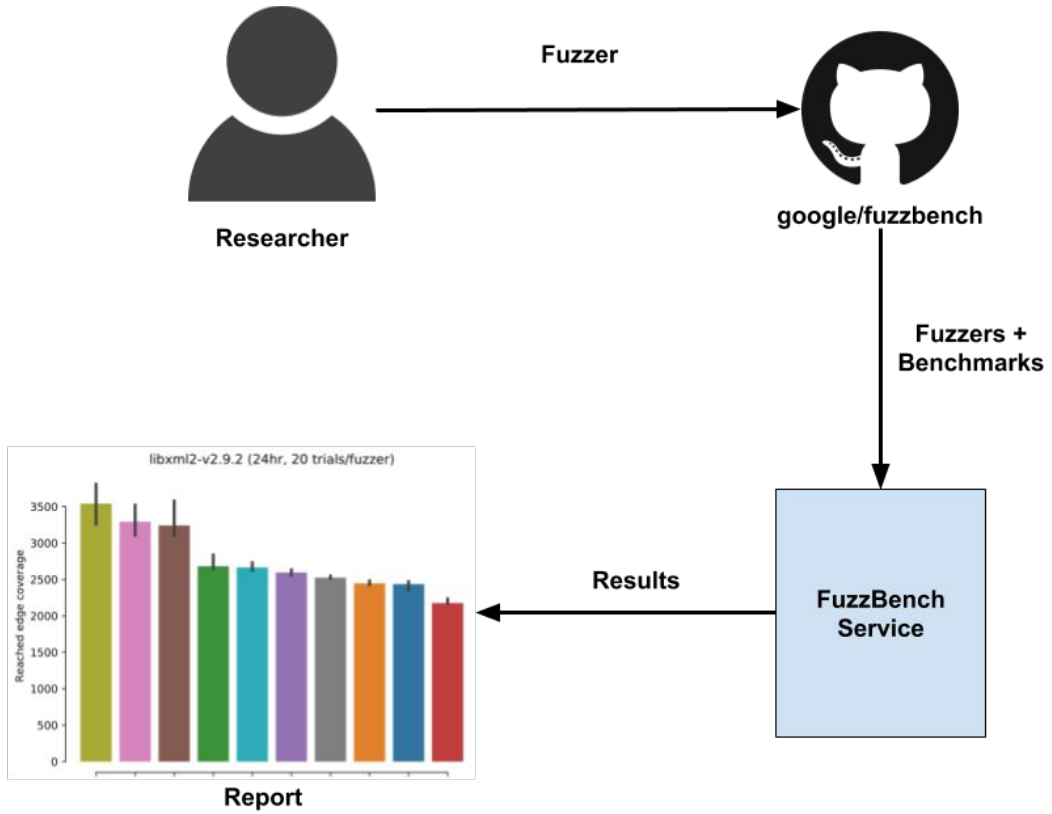
Why Fuzzing is Important

1. Fuzzing tests provide a complete picture of the tested software and system.
2. Avoid zero day exploit
3. Fuzzing is relatively inexpensive in terms of both cost and time
4. Fuzzing finds bugs that conventional testing or manual audits wouldn't.

FuzzBench

- Benchmarking as a Service
- free to evaluate fuzzers
- fast and reliable benchmarking
- reproducible experiments
- High system requirements

  (e.g. 96 core Google Compute Engine)



Experiment summary with different fuzzers

By avg. score

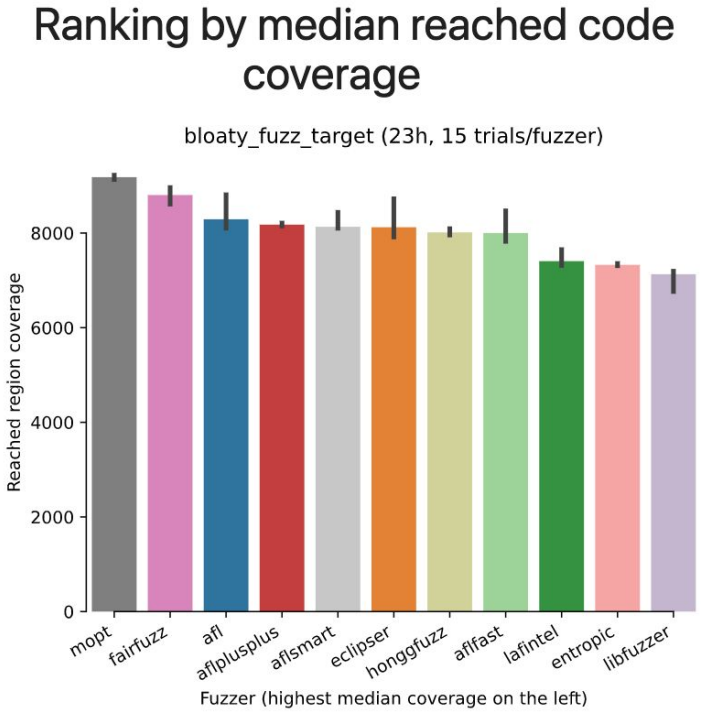
fuzzer	average normalized score
honggfuzz	97.90
aflplusplus	96.61
entropic	94.17
eclipser	94.12
libfuzzer	90.57

By avg. rank

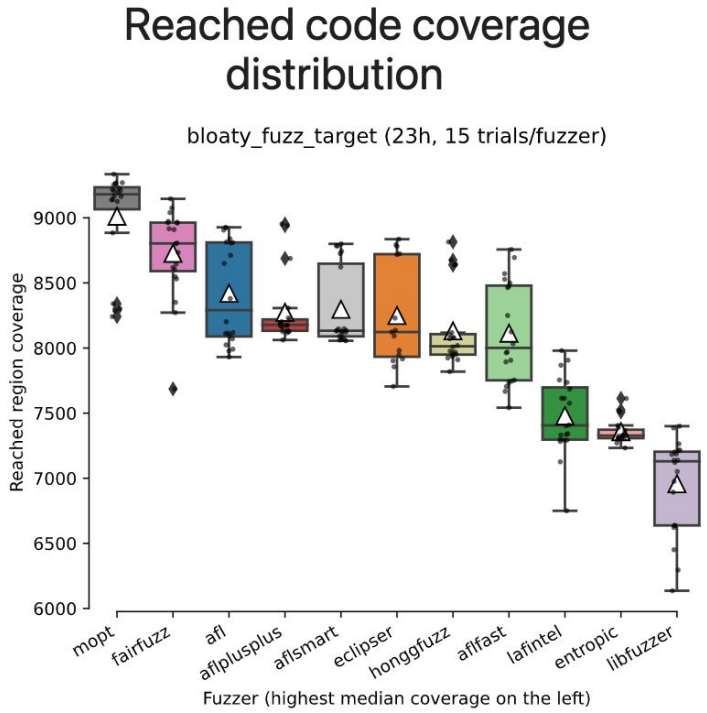
fuzzer	average rank
aflplusplus	3.74
honggfuzz	4.19
entropic	4.36
eclipser	4.79
afl	5.29

Results - 2021-04-11

bloaty_fuzz_target summary

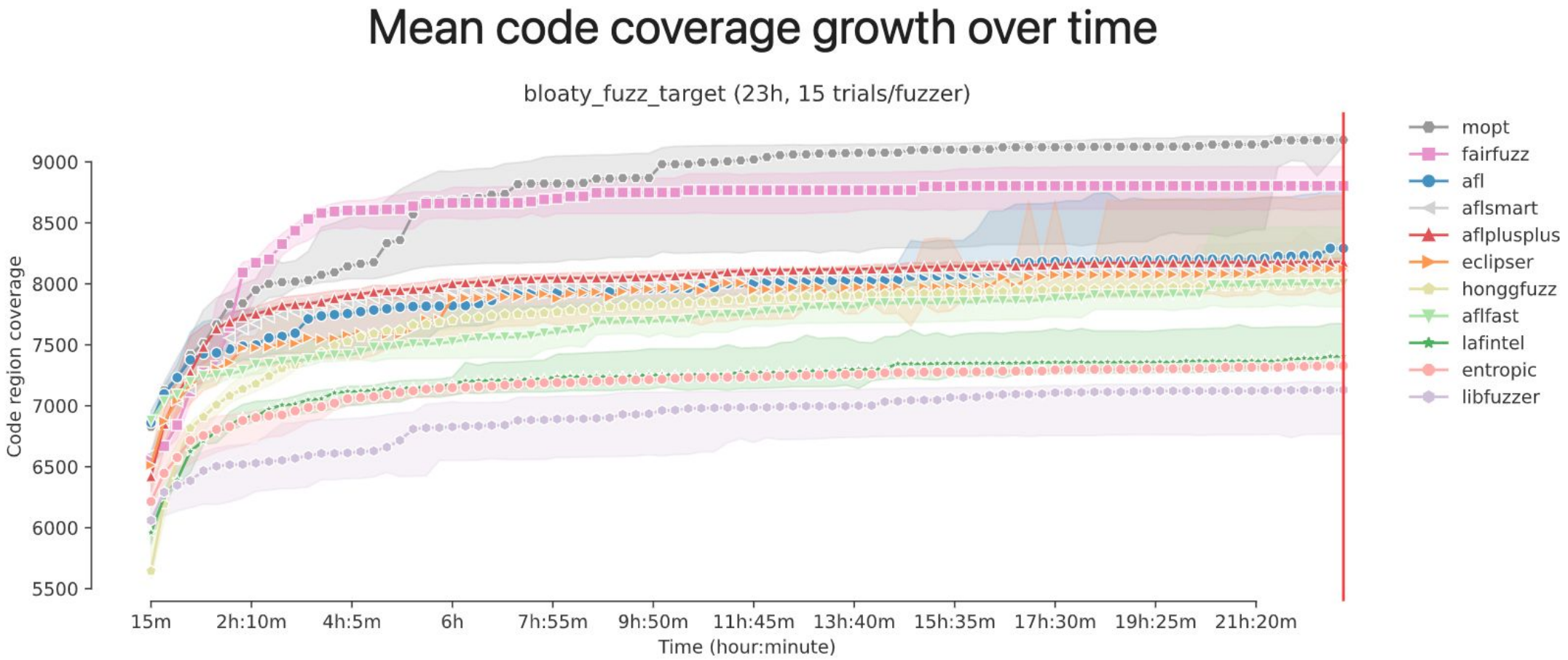


CODE COVERAGE (LINEAR)



CODE COVERAGE (LOG)

Results - continued



* The error bands show the 95% confidence interval around the mean code coverage.

https://www.fuzzbench.com/reports/2021-04-11/index.html

RevBugBench with FuzzBench

Applies FixReverter
on real world programs

Able to inject a large number of bugs

Integrated with Fuzzbench to
add Scalability and Reproducibility

FixReverter

- Identifying common bugfix patterns
  - Abort
  - Exec
  - Assign
- Defining a new framework to inject bugs
- Developing a novel fuzzing benchmark
- Evaluate the benchmark using 5 recent fuzzers

Definition of effective fuzzing benchmark:

1. Target programs must be relevant
2. Bug must be relevant
3. Bugs must be easy to identify
4. Prevent overfitting for certain programmes and bugs

FixReverter - Bug Injection

Inject 10 programs with bugs.

- Requirements:
  - Minimum 200GB RAM
- Experiment Setup:
  - 2x Intel(R) Xeon(R) Silver 4116 CPUs

    192GB RAM

    Ubuntu 16.04
  - 2x Intel(R) Xeon(R) Gold 5218 CPUs

    384GB RAM

    Ubuntu 18.04

Steps:

1. Syntax Matcher
   a. convert grammar file into state machine
   b. traverse Clang abstract syntax tree
2. Static reachability & dependence analysis

   around 71% of injection sites dropped
3. Bug Injector & Naive Bug Filter

   drop uninteresting bugs

RevBugBench - Experiments

Setup:

- Setup FuzzBench
- Import RevBugBench into FuzzBench
- Add configuration

Experiment configuration:

- 3 trials
- 24 hours
- 5 fuzzers
- 10 benchmarks

Contributions:

- Upgraded RevBugBench to work with new version of FuzzBench

```
1
2   experiment: test-experiment
3
4   # The number of trials of a fuzzer-benchmark pair.
5   trials: 3
6
7   # The amount of time in seconds that each trial is run for.
8   # 1 day = 24 * 60 * 60 = 86400
9   max_total_time: 86400
10
11  # The location of the docker registry.
12  # FIXME: Support custom docker registry.
13  # See https://github.com/google/fuzzbench/issues/777
14  docker_registry: gcr.io/fuzzbench
15
16  # The local experiment folder that will store most of the experiment data.
17  # Please use an absolute path.
18  experiment_filestore: "/home/daniel/experiment-data"
19
20  # The local report folder where HTML reports and summary data will be stored.
21  # Please use an absolute path.
22  report_filestore: "/home/daniel/report-data"
23
24  # Flag that indicates this is a local experiment.
25  local_experiment: true
```

FixReverter - Bug Triage

Identify and categorize unique causes of crashes:

- *"individual cause"* - single bug causes a crash
- *"combination cause"* - set of bugs together cause crash

Single input can trigger multiple crashes with different causes.

Recommendations:

- CPU with more than 24 cores to speed up execution

Input: I: crashing input; ts: set of triggered injections
Output: bs: the set of I's failure causes

```
1:  bs ← ∅
2:  for i from 1 to ‖ts‖ do
3:      for each set s ∈ Powerset(ts) where ‖s‖ = i do
4:          if ∄s′ ∈ bs. s′ ⊂ s then
5:              Run I on inject(s)
6:              if I crashes then
7:                  bs ← bs ∪ {s}
8:              end if
9:          end if
10:     end for
11: end for
```

Figure 8: Bug triage algorithm.

Conclusion

- Most fuzzer benchmarks use code coverage instead of unique crashes
- Need for effective and standard way for evaluating fuzzers
- Validation FixReverter and RevBugBench implementation
- Updated RevBugBench to work with new version of FuzzBench

Thank you! Questions?

Universiteit
Leiden
The Netherlands

References

- Zhang, Z., Patterson, Z., Hicks, M. and Wei, S., 2022. {FIXREVERTER}: A Realistic Bug Injection Methodology for Benchmarking Fuzz Testing. In *31st USENIX Security Symposium (USENIX Security 22)* (pp. 3699-3715).
- Metzman, J., Szekeres, L., Simon, L., Sprabery, R. and Arya, A., 2021, August. FuzzBench: an open fuzzer benchmarking platform and service. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1393-1403).
- Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. {AFL++}: Com bining incremental steps of fuzzing research. In 14th USENIX Workshop on Offensive Technologies (WOOT 20), 2020
- Rahul Gopinath, Carlos Jensen, and Alex Groce. Code coverage for suite evaluation by developers. In Proceedings of the 36th International Conference on Software Engineering, pages 72–82, 2014.
- Laura Inozemtseva and Reid Holmes. Coverage is not strongly correlated with test suite effectiveness. In Proceedings of the 36th international conference on software engineering,
- Pavneet Singh Kochhar, Ferdian Thung, and David Lo. Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. In 2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER), pages 560–564. IEEE, 2015