icLibFuzzer: Isolated-context libFuzzer for Improving Fuzzer Comparability

Yu-Chuan (Jason) Liang and Hsu-Chun Hsiao
National Taiwan University, Taiwan

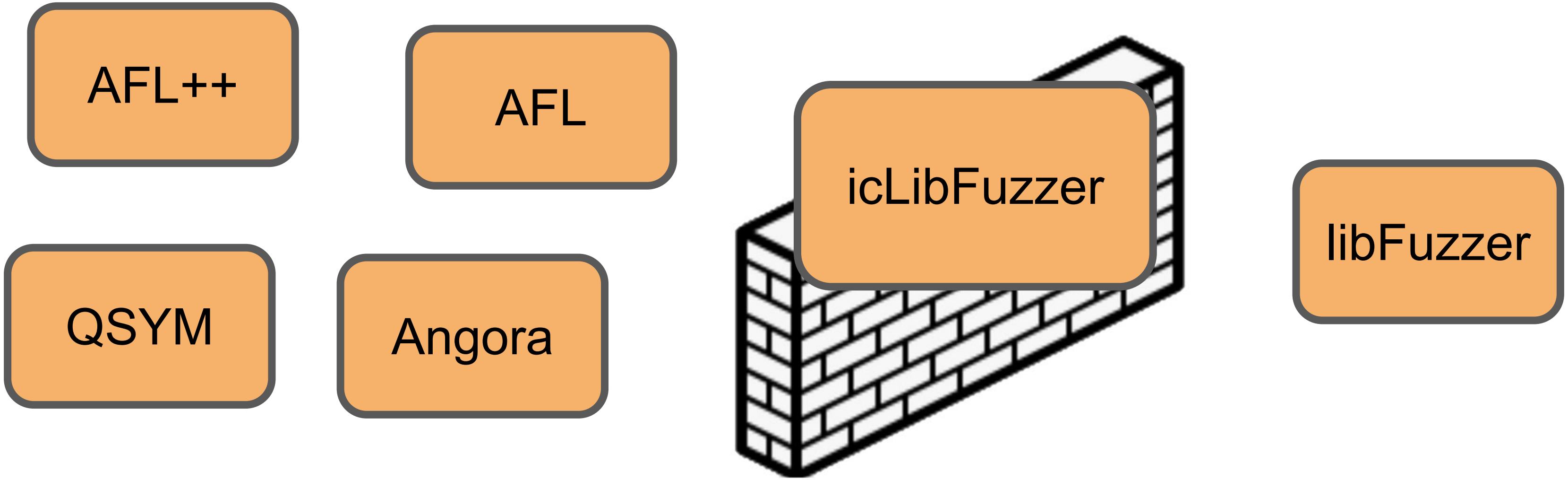
2021 NDSS BAR Workshop

國立臺灣大學
National Taiwan University

NSLAB

Motivation

libFuzzer is strong, yet it is seldom being compared with other academic papers.



Outline

- Background
- Comparability issues
- icLibFuzzer
- Evaluation
- Conclusion & future work

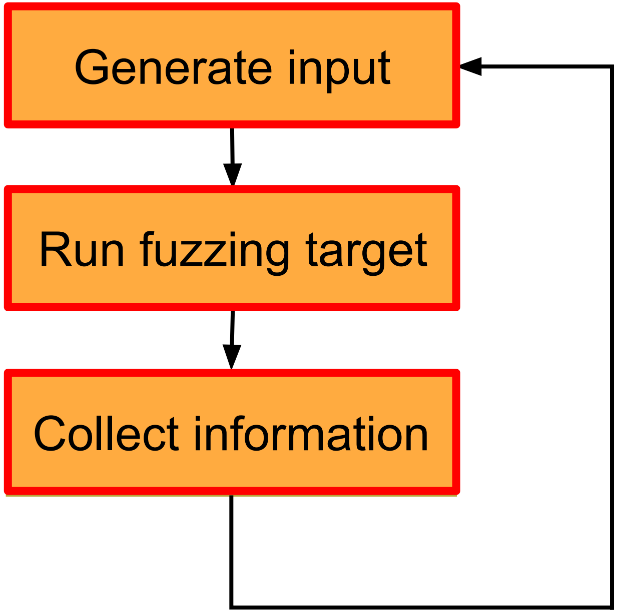
Outline

- Background
- Comparability issues
- icLibFuzzer
- Evaluation
- Conclusion & future work

Background – Fuzzing & libFuzzer

- Fuzzing tries to trigger abnormal behavior of a target program.
- Fuzzer is the fuzzing engine.

- libFuzzer
  - a coverage-guided fuzzer
  - targets library functions
  - supports advanced instrumentation, e.g., CMP tracing & value profiling



Background – example of libFuzzer's advanced instrumentation (CMP tracing)

```
short int magic;
read(0, &magic, sizeof(short int));
if (magic == 0x1234)
save_compared_arguments();
    bug();
if (magic == 0x1234)
    bug();
```

input \xab\xcd\xff → record (0xabcd,0x1234) → input \x12\x34\xff

Background – example of libFuzzer's advanced instrumentation (value profiling)

```
short int magic;
read(0, &magic, sizeof(short int));
save_compared_distance();
if (magic == 0x1234)
    bug();
```

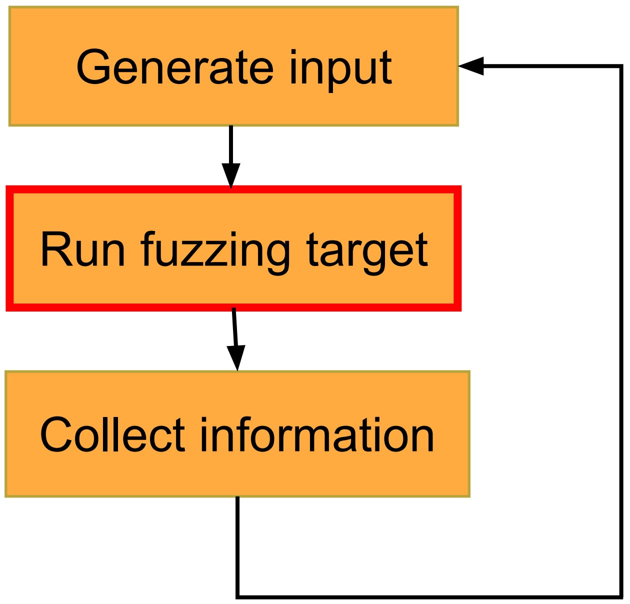
input \x12\x45 → record hamming(0x1245,0x1234)

not closer → discard

closer → used to generate next input

Background – libFuzzer's in-process infrastructure

- Treats the fuzzing target as a callback function
- The fuzzer instance and fuzzing target share the same context (e.g., virtual memory space).
  - Pro: fast
  - Con: context pollution



Outline

- Background
- Comparability issues
- icLibFuzzer
- Evaluation
- Conclusion & future work

Comparability issues – lack support of common metrics

- libFuzzer aborts immediately after the target crashes
- Cannot be evaluated using common metrics
  - E.g., code coverage, time to find all intended bugs, ...

- How prior work compares with libFuzzer
  - Uses the time-to-first-crash metric only
  - Compares on fine tuned datasets
  - Enables the ignore-crash mode

limited comparison scope

context pollution problem

Comparability issues – context pollution

- libFuzzer's ignore-crash mode restarts the fuzzer after each crash.
- It may produce wrong results due to context pollution.

- Context pollution may occur when
  a. the fuzzing target depends on global variables.
  b. memory leak exists in the fuzzing target.

Comparability issues – context pollution

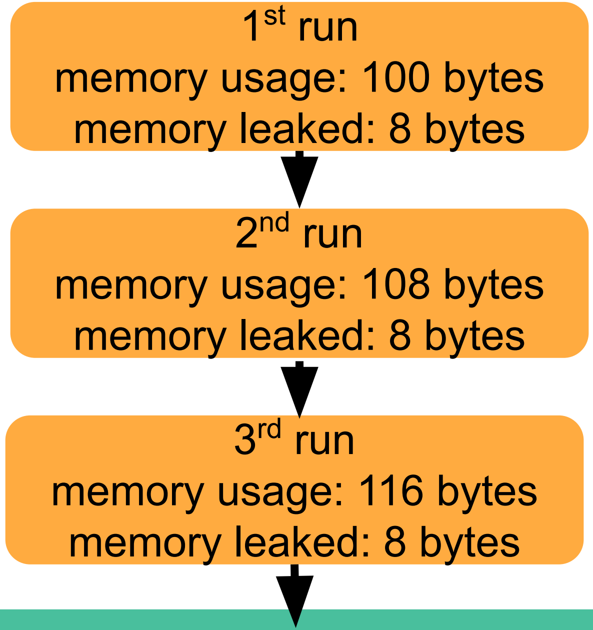
a. When the fuzzing target depends on global variables: C/C++ programs may assume that global variables will be initialized before the main function starts.

```
int getopt(int argc, char *const argv[],
  const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

./fuzzing_target -f /tmp/log → Before 1st run: optind == 1 → Before 2nd run: optind == 3 → Unexpected non-bug behavior

Comparability issues – context pollution

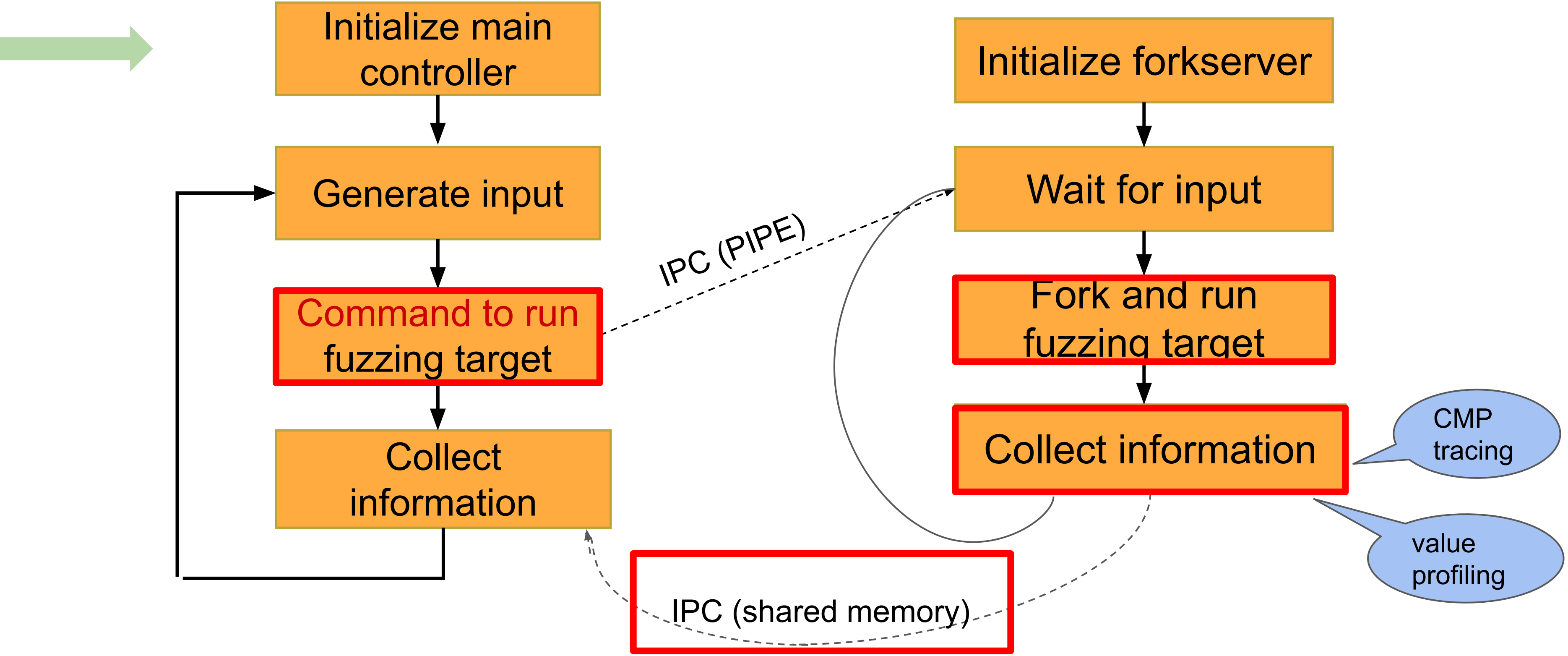
b. When memory leak exists in the fuzzing target: Memory leak will persist and keep consuming the memory until libFuzzer crashes.



Outline

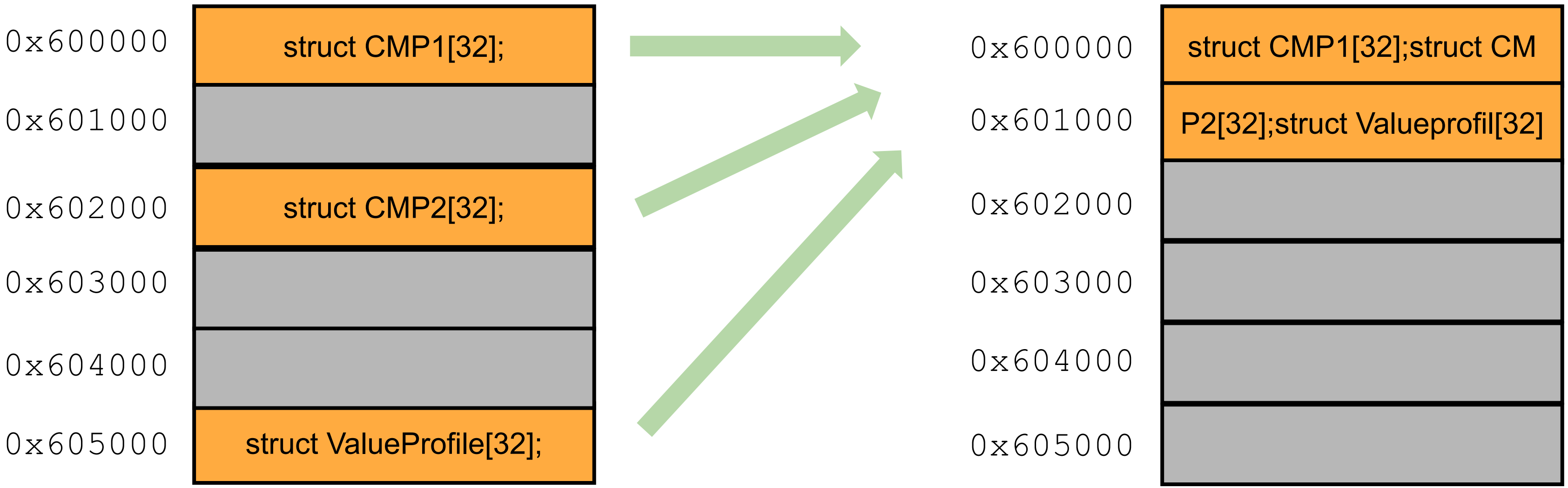
- Background
- Comparability issues
- icLibFuzzer
- Evaluation
- Conclusion & future work

icLibFuzzer – forkserver infrastructure to avoid context pullution



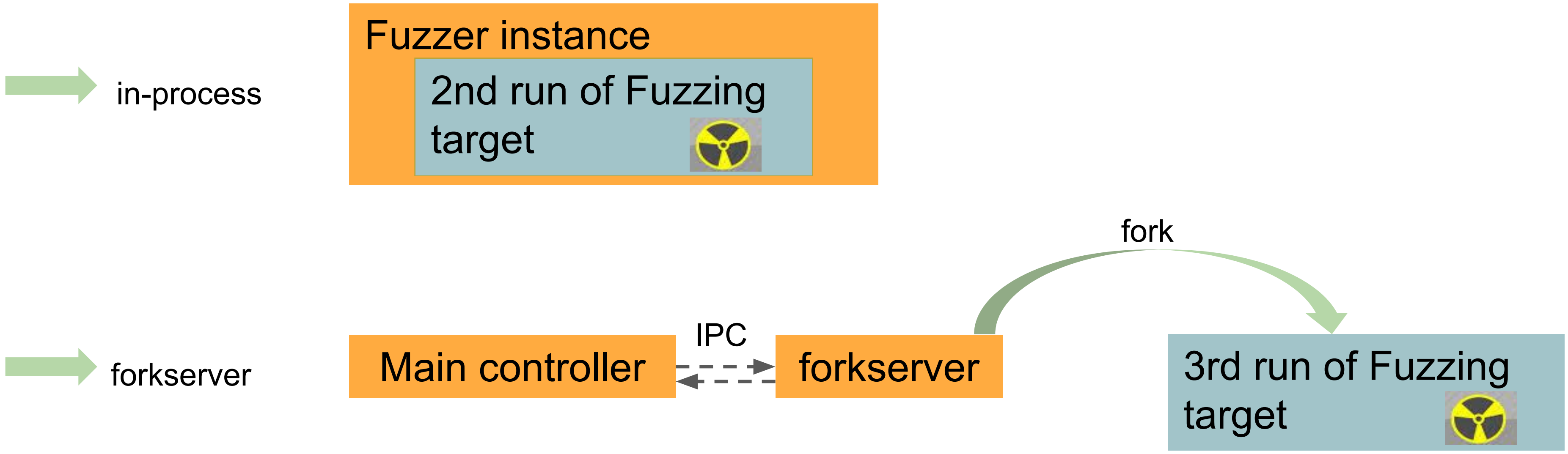
icLibFuzzer – structure packing for faster forking

Pack multiple data structure all-in-one to minimize memory size.



icLibFuzzer – in-process vs. forkserver

- in-process is more fragile to context pollution
- in-process is faster than forkserver



Outline

- Background
- Comparability issues
- icLibFuzzer
- Evaluation
- Conclusion & future work

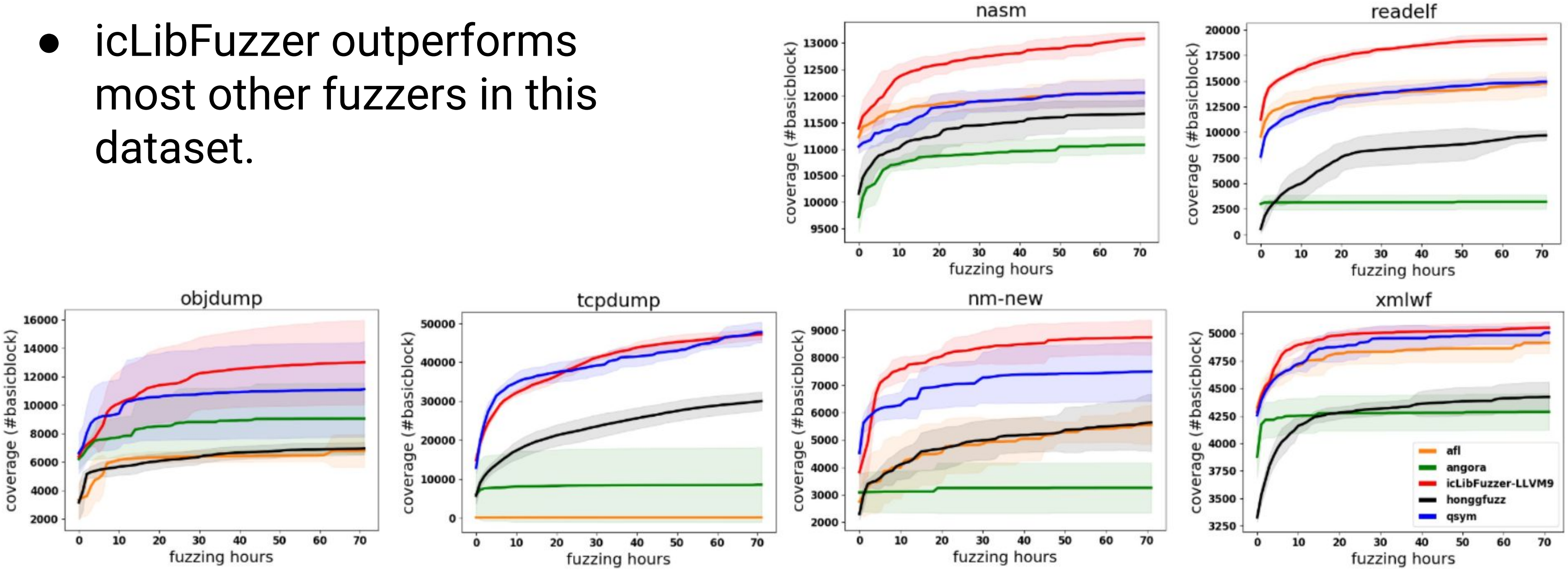
Evaluation – setup

- AMD Ryzen Threadripper 2990WX 32-Core processor, 64 GB memory, Ubuntu 18.04.
- Fuzzers
  - AFL (afl-clang-fast)
  - Honggfuzz
  - QSYM
  - Angora
- Run each binary eight times, without initial seed [1], each for 72 hours.
- Code coverage is calculated using llvm-cov.

[1] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, “Evaluating fuzz testing,” in ACM CCS, 2018.

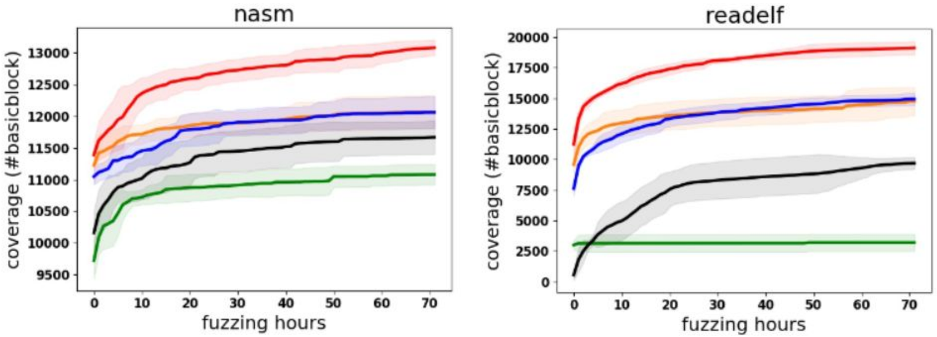
Evaluation – code coverage (1/3)

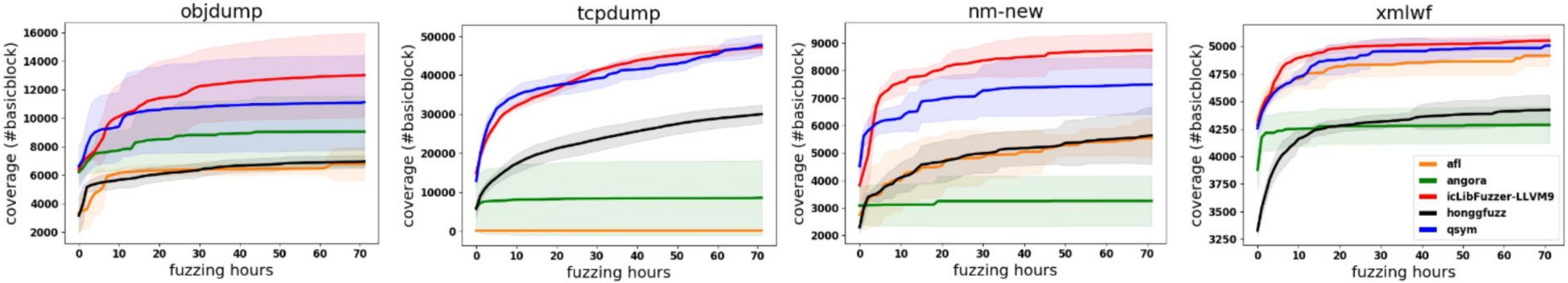
- icLibFuzzer outperforms most other fuzzers in this dataset.



Evaluation – code coverage (2/3)

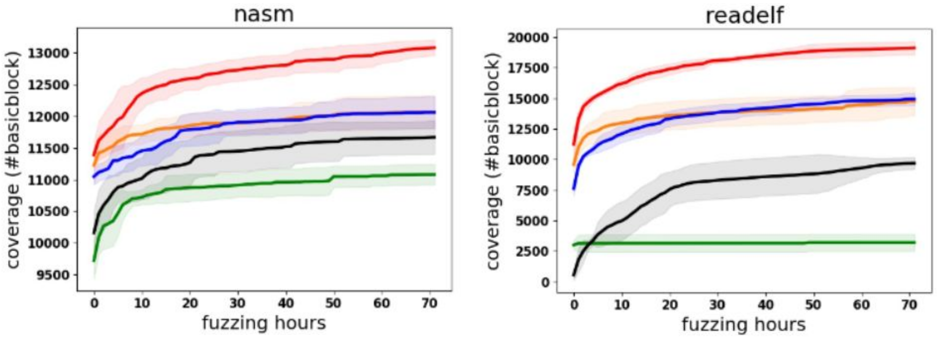
- Coverage keep growing stably after 24 hours
  - 24 hours is not enough

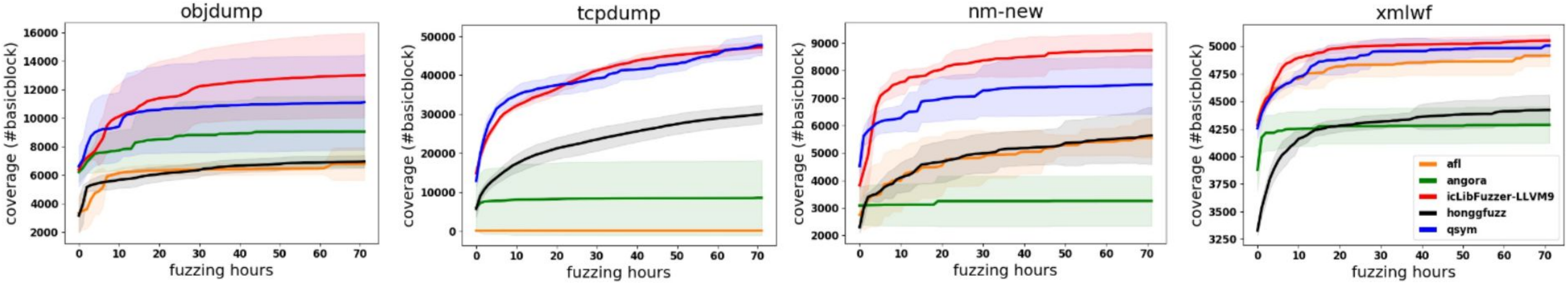




Evaluation – code coverage (3/3)

- Angora does not perform as well as its original paper
    - possible reason: no initial seed





Evaluation - speed difference

- icLibFuzzer is 4x to 50x slower than original libFuzzer on fuzzer-test-suite.

	libFuzzer	icLibFuzzer	AFL
boringssl-2016-02-12	26707	1722	1201
freetype2-2017	9210	927	1038
libpng-1.2.56	8707	2556	3340
openssl-1.1.0c	22460	512	560
sqlite-2016-11-14	20587	1112	881

executions per second on fuzzer-test-suite dataset

- icLibFuzzer runs almost as fast as AFL.

	readelf	nm-new	objdump	tcpdump	nasm	xmlwf
icLibFuzzer	1701	1667	1030	1603	359	2349
AFL	1708	1906	1032	2438	337	2983
qsym	1380	2990	1174	3087	430	4917
angora	1300	681	684	1068	419	1157
honggfuzz	86	96	116	108	76	126

executions per second on real-world programs

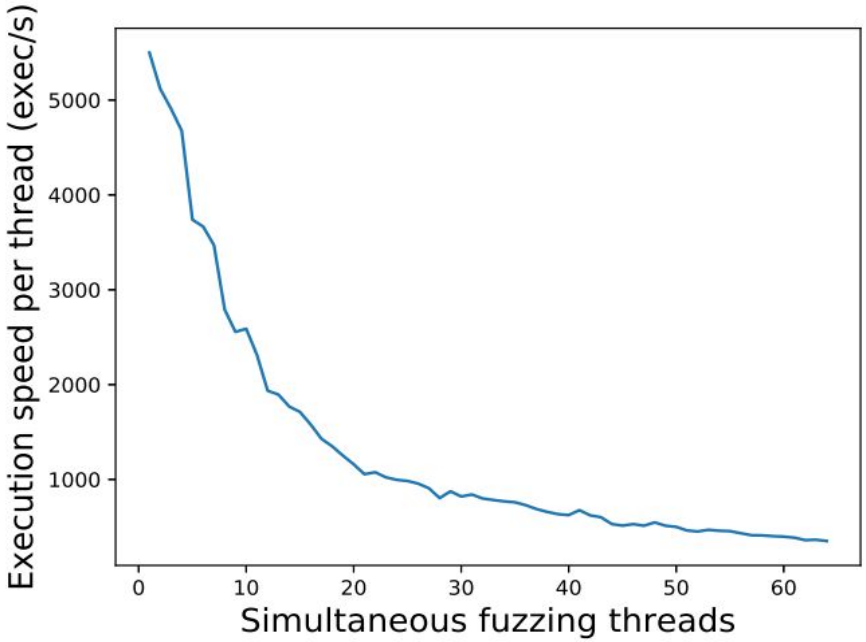
Evaluation - structure packing

Structure packing halves the memory usage and double the speed.

	exec/s		RSS (Bytes)	
	no pack	pack	no pack	pack
objdump	1992	3740	7260K	3636K
readelf	2404	4426	7672K	3356K
nm-new	2083	3991	7544K	3324K
tcpdump	2168	4103	7504K	3492K

Evaluation - interesting phenomenon

Execution speed is highly related to the number of simultaneously fuzzing threads. [2]



[2] W. Xu, S. Kashyap, C. Min, and T. Kim, “Designing new operating primitives to improve fuzzing performance,” in ACM CCS, 2017.

Outline

- Background
- Comparability issues
- icLibFuzzer
- Evaluation
- Conclusion & future work

Conclusion

- libFuzzer lacks support of common metrics, and suffer from context pollution.
- We propose icLibFuzzer, to improve comparability of libFuzzer
- icLibFuzzer may serve as another baseline in fuzzing research

Future work

- The impact of initial seeds and how to choose them wisely
- How to speed up icLibFuzzer? Infrastructural or implementation wise?
- Cache-aware structure packing?

Questions?
NDSS Slack: @jason liang
Email: jasonliang30115@gmail.com
https://github.com/csienslab/icLibFuzzer

Reference

[1] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, “Evaluating fuzz testing,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2123–2138. [Online]. Available: https://doi.org/10.1145/3243734.3243804

[2] W. Xu, S. Kashyap, C. Min, and T. Kim, “Designing new operating primitives to improve fuzzing performance,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS ’17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2313–2328. [Online]. Available: https://doi.org/10.1145/3133956.3134046