Sydr-Fuzz

ISP RAS

Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle

---

Alexey Vishnyakov infosec.exchange/@VishnyaSweet
Daniil Kuts
Vlada Logunova
Darya Parygina
Eli Kobrin
Georgy Savidov
Andrey Fedotov infosec.exchange/@anfedotoff

December 2, 2022

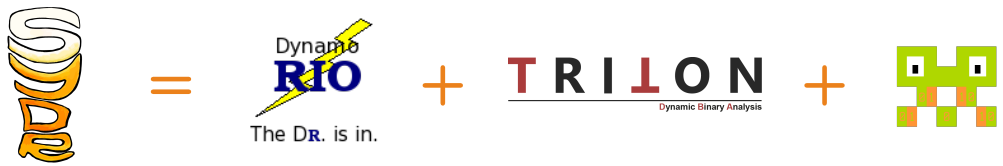
ISP RAS

arxiv.org/abs/2211.11595

Motivation

- 81% of 2400 audited commercial codebases contained at least one vulnerability (Synopsys 2022 Report)
- Security development lifecycle (SDL) is an industry standard for detecting program errors before deployment
- Hybrid fuzzing utilizes dynamic symbolic execution (DSE) and outperforms coverage-guided fuzzing
- Automated continuous dynamic analysis pipeline allows to spot bugs missed during manual fuzzing

Dynamic Symbolic Execution with Sydr



- Sydr uses DynamoRIO as a DBI framework
- Sydr uses Triton as a DSE engine
- Triton uses Bitwuzla as an SMT solver

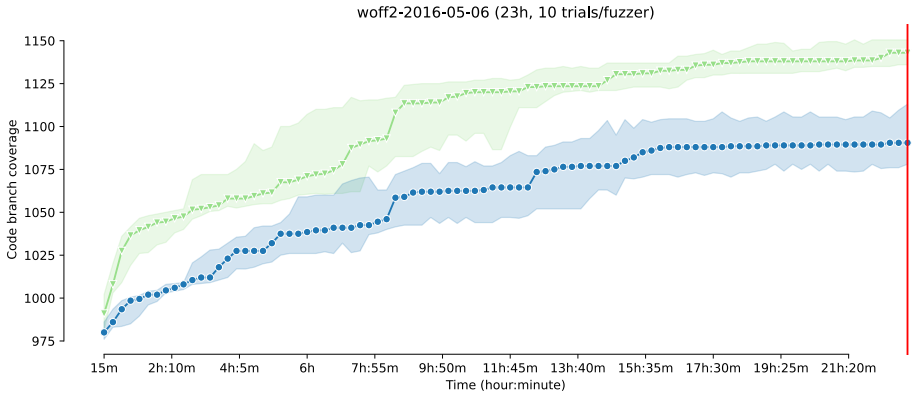
Dynamic symbolic execution:

- Each input byte is modeled by a free *symbolic variable*
- Instructions interpretation produce SMT formulas
- *Symbolic state* maps registers and memory to SMT formulas
- *Path predicate* contains taken branch constraints
- Sydr inverts branches to explore new paths and solves security predicates to detect errors (out of bounds, integer overflow, etc.)

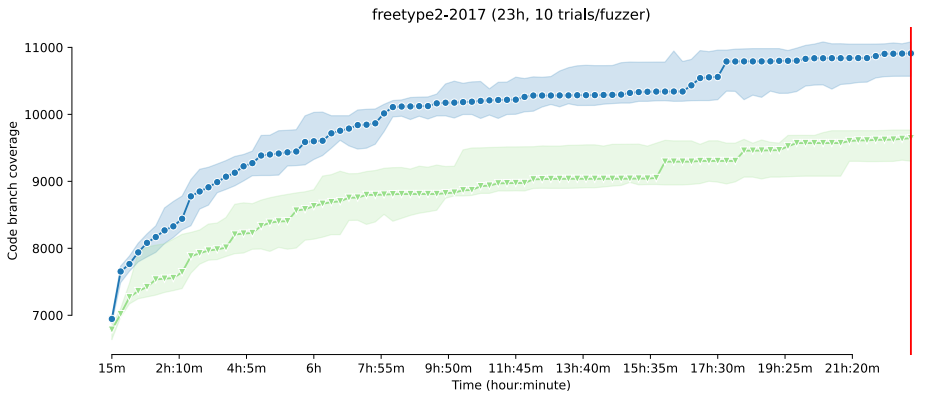
Contributions

- Hybrid fuzzer Sydr-Fuzz: Sydr & libFuzzer/AFL++
- First integration between a DSE-tool and libFuzzer
- Symbolic pointers reasoning helps hybrid fuzzing
- Dynamic analysis pipeline: hybrid fuzzing, corpus minimization, symbolic security predicates, coverage collection, and crash triaging
- Continuous hybrid fuzzing infrastructure
- Sydr-Fuzz outperforms coverage-guided fuzzers and proves to be comparable to hybrid fuzzers on Google FuzzBench
- Crash triaging tool Casr is open-sourced: github.com/ispras/casr

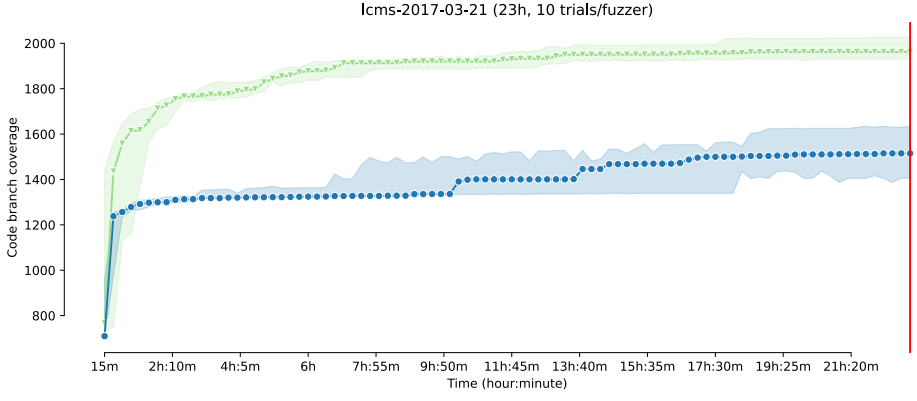
1. Sydr-Fuzz achieved higher coverage than other fuzzers
2. Sydr-Fuzz outperformed existing fuzzers on most benchmarks



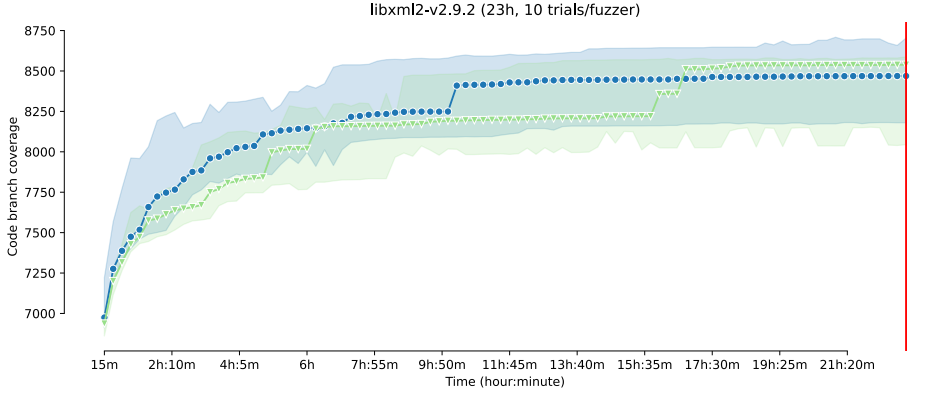
Sydr+libFizzer vs 2xlibFuzzer



Sydr+AFL++ vs SymQEMU+AFL++



Sydr+AFL++ vs 2xAFL++



Sydr+AFL++ vs FUZZOLIC+AFL++

sydr-fuzz.github.io/fuzzbench

Sydr-Fuzz: Dynamic Analysis Pipeline

- Hybrid fuzzing with Sydr and libFuzzer/AFL++: `sydr-fuzz run`
- Corpus minimization: `sydr-fuzz cmin`
- Error detection (out of bounds, integer overflow, etc.) via symbolic security predicates: `sydr-fuzz security`
- Crash triaging (deduplication, clustering, severity estimation) with Casr: `sydr-fuzz casr`
- Collecting coverage: `sydr-fuzz cov-report`

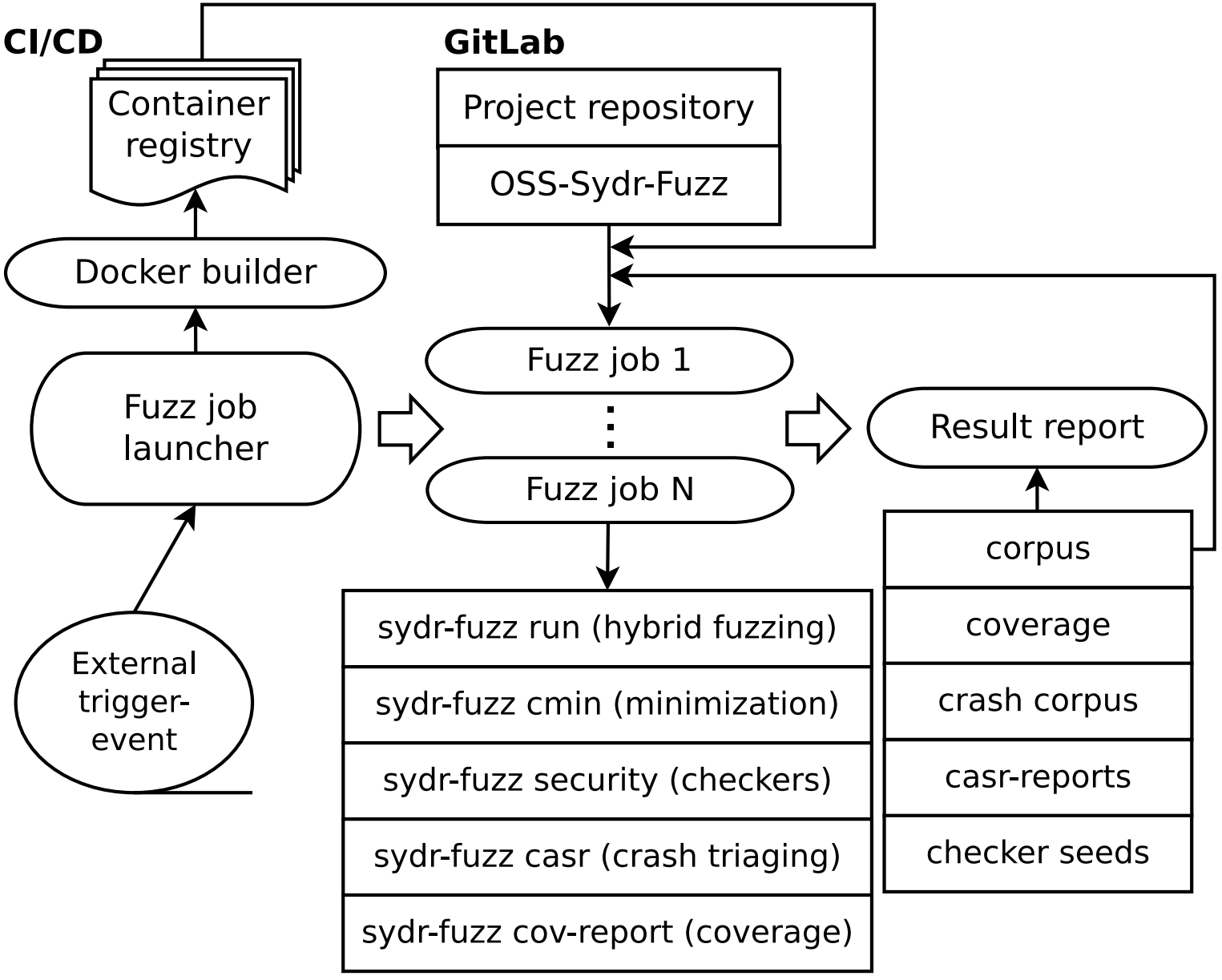
Sydr-Fuzz DEMO

OSS-Sydr-Fuzz: Hybrid Fuzzing for Open Source

github.com/ispras/oss-sydr-fuzz — fork of OSS-Fuzz for hybrid fuzzing with Sydr-Fuzz

- **45** projects and **300+** fuzz targets
- During one year sydr-fuzz discovered **85** new bugs in 22 projects: TensorFlow, PyTorch, Cairo (GTK), OpenJPEG, Poppler, ICU, Tarantool, Torchvision, etc. All trophies on GitHub
- 13 issues were found by Sydr symbolic security predicates

Continuous Hybrid Fuzzing Infrastructure



Sydr & libFuzzer

- libFuzzer workers use shared corpus directory
- Sydr takes seeds to modify and puts generated seeds to the same directory
- libFuzzer immediately loads seeds generated by Sydr
- Reloaded files are logged by libFuzzer: reviews.llvm.org/D100303000
- Sydr-Fuzz removes not reloaded seeds from corpus
- Scheduling seeds for Sydr:
  - whether seed discovered new function
  - whether seed brought new coverage
  - whether seed increased libFuzzer features
  - *creation time/size*

Sydr & AFL++

- Sydr is launched as a fake secondary AFL worker
- Sydr is executed on seeds from AFL main worker queue
- Sydr-Fuzz uses afl-showmap to minimize seeds generated by Sydr before putting them in Sydr worker queue
- AFL main worker scans Sydr queue and imports useful seeds
- Seeds for Sydr are scheduled: new coverage, initial corpus seed, file size, novelty
- Running AFL++ in parallel mode with automatically assigned options (schedulers, MOpt, etc.)

Symbolic Security Predicates

- Out of bounds, integer overflow, etc.
- Security predicates are checked on minimized corpus after fuzzing
- Generated seeds are verified on sanitizers
- Deduplication of detected errors

Integer Overflow to Buffer Overflow in Rizin

```
symbols_size = (symbols_count + 1) * 2 * sizeof(struct symbol_t);
if (symbols_size < 1) {
    ht_pp_free(hash);
    return NULL;
}
if (!(symbols = calloc(1, symbols_size))) {
    ht_pp_free(hash);
    return NULL;
}
...
symbols[j].last = true;
```

CASR: Crash Triaging

- `casr-san` runs crashes on sanitized binary and creates reports
- Crash report contains stack trace, crash line, crash severity, assembly, source, etc.
- `casr-cluster -d` deduplicates crashes based on stack trace hash
- `casr-cluster -c` performs hierarchical clustering of Casr reports
- `casr-gdb` generates crash reports for non-instrumented binaries

Average Number of Imported Seeds from Symbolic Engines

Application	Sydr	SymQEMU	FUZZOLIC
freetype2	**307.8**	90.8	241.9
harfbuzz	**58.8**	34.8	21.3
lcms	139.3	192.5	**203.5**
libpng	**30.4**	25.7	23.9
libjpeg-turbo	**17.5**	13.5	14.6
libxml2	34.8	**41.9**	26.9
mbedtls	17.1	18.0	**31.0**
openthread	59.3	38.1	**72.9**
re2	**2.5**	2.0	0.1
sqlite3	59.7	88.2	**96.6**
vorbis	3.1	**3.5**	2.2
woff2	**24.9**	13.0	—
zlib_uncompress	1.2	**4.7**	3.8

FuzzBench Configuration

- Mean code coverage (23 hours, 10 trials per fuzzer)
- 1 fuzzer + 1 Sydr
- Two binaries: sanitizers for fuzzer and no instrumentation for Sydr
- Invert branches in direct order for 2 minutes
- 10 second limit for single query and 60 seconds total solving limit
- Asynchronous solving during path predicate construction
- Path predicate construction is suspended when solving queue contains 300 queries
- Explored paths cache for Sydr (QSYM-like)
- Symbolic addresses fuzzing by default, switched to full symbolic pointers reasoning every 25th Sydr launch

FuzzBench Summary

- Sydr-Fuzz outperformed libFuzzer on **9** out of 14 benchmarks
- Sydr-Fuzz outperformed AFL++ on **9** out of 14 benchmarks
- Sydr-Fuzz outperformed SymQEMU on **7** out of 13 benchmarks
- Sydr-Fuzz outperformed FUZZOLIC on **6** out of 12 benchmarks
- Sydr-Fuzz reached higher coverage normalized score in all experiments

Future Work

- Hybrid fuzzing for AARCH64 (Baikal-M) / RISC-V
- Dynamic analysis pipeline for Python via Atheris and Casr
- Security predicates for integer truncation, command injection, and format string errors

Questions?

Telegram: @sydr_fuzz