



AFLrusttrust^A and LibAFL_libFuzzer^B

^AAndrea Fioraldi¹, Dominik Maier², Dongjia Zhang³, Addison Crump⁴

^BAddison Crump⁴, Andrea Fioraldi¹, Dominik Maier², Dongjia Zhang³

¹EURECOM, ²Google Inc., ³The University of Tokyo, ⁴CISPA Helmholtz Center for Information Security



LibAFL

- Fuzzer framework
- Written in Rust
- Standardised high-performance components
- Highly configurable for creating **custom runtimes**



Motivation and Design

- AFL++ and LibAFL consistently top Fuzzbench results
- We want to demonstrate LibAFL's flexibility
- We want to make LibAFL more widely used
- Write runtimes for popular fuzzers in LibAFL

AFLrustrust





AFLrustrust: a shim for AFL++

- AFL++-compiled binaries export data for AFL++
- LibAFL can observe this feedback
- Use LibAFL's components to speed up fuzzing
- User does not need to modify fuzzing infrastructure



AFLrustrust design

- Instrumentation provided by AFL++'s LLVM pass
- Edge coverage + cmplog via shared memory
- AFL-style forkserver implemented in LibAFL
- Corpus scheduling with the EXPLORE power schedule
- Effectively: AFL++, but written with LibAFL components



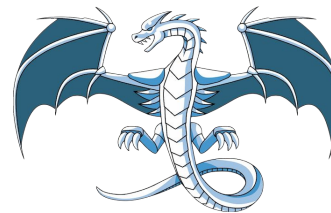
AFLrustrust's differences

- Redqueen disabled (experimental support in LibAFL)
- Coverage map acceleration with SIMD
- MOpt enabled by default
- AFL++ implementation in <400 lines with LibAFL

LibAFL_libFuzzer

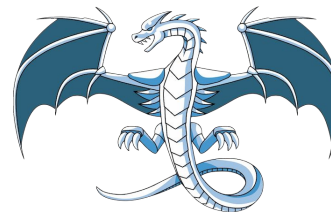


libFuzzer



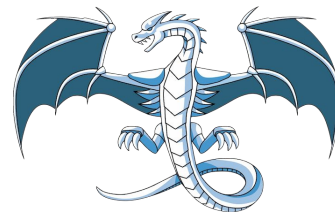
- libFuzzer by LLVM is the de facto standard for in-process fuzzing
- Shipped with LLVM's compiler-rt
- Depends on default LLVM instrumentation (-fsanitize-coverage...)
- Compatible with most LLVM-based compilers
- Entered maintenance mode in 2022

LibAFL_libFuzzer: a shim for libFuzzer



- Ongoing project to build a full replacement for libFuzzer
- Intentionally constrained to libFuzzer instrumentation
- Fully compatible with libFuzzer flags and support
- Utilises LibAFL's components to improve fuzzing performance

LibAFL_libFuzzer's differences



- Power scheduling/minimising algorithm from AFL++
- GRIMOIRE-style structured analysis and mutation
- AFL-style cmplog
 - Some libFuzzer comparison interceptors not implemented
 - Mutations optimised for string inputs not implemented

Concluding Thoughts





LibAFL, and why we use it

- Frequent updates and community fixes
- Fast implementations of bleeding edge fuzzing techniques
- Common baseline for comparing and combining strategies
 - Ask us about how we use LibAFL to evaluate!



So, what's next from us?

- Continued development of LibAFL_libFuzzer
 - Windows/macOS/etc.
 - Comparison interceptors
 - Plug-and-play Rust fuzzer support
- RedQueen stabilisation

Questions?