



LibAFL

The Advanced Fuzzing Library

Dominik Maier & Andrea Fioraldi

[@domenuk](https://twitter.com/@domenuk), [@andrea_fioraldi](https://twitter.com/@andrea_fioraldi)

dictionary : n/a, n/a, n/a

{dominik, andrea}@aflplus.plus

py/custom : 0/0, 0/0

trim : 19.25%/53.2k, n/a

overall results
cycles done : 15
total paths : 703
uniq crashes : 0
uniq hangs : 0

map coverage

map density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple

findings in depth

favored paths : 114 (16.22%)

total crashes : 0 (0 unique)

total tmouts : 0 (0 unique)

path geometry

levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%

[cpu000: 12%]



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

run time : 0 days, 0 hrs, 0 min, 43 sec

new path : 0 days, 0 hrs, 0 min, 1 sec

last uniq crash : none seen yet

last uniq hang : none seen yet

cycle progress

now processing : 261*1 (37.1%)

paths timed out : 0 (0.00%)

stage progress

now trying : splice 14

stage execs : 31/32 (96.88%)

total execs : 2.55M

exec speed : 61.2k/sec

fuzzing strategy yields

bit flips : n/a, n/a, n/a

byte flips : n/a, n/a, n/a

arithmetics : n/a, n/a, n/a

known ints : n/a, n/a, n/a

dictionary : n/a, n/a, n/a

splice/splice : 506/1.05M, 1%

custom/custom : 0/0, 0/0

trim : 19.25%/53.2k, n/a

map coverage

overall results

cycles done : 15

unique finds : 702

uniqueness : 100%

map density : 100%

count coverage : 3.30

findings : 111

favored paths : 111 (16.22%)

new edges on : 167 (23.76%)

unique edges : 167 (23.76%)

unique nodes : 167 (23.76%)

graph geometry

levels : 11

branching : 121

bad fav : 0

finds : 699

sorted : n/a

utility : 99.88%

[cpu000: 12%]

Who We Are

- Hackademics (both PhD students)





american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

run time : 0 days, 0 hrs, 0 min, 43 sec

new paths : 0 days, 0 hrs, 0 min, 1 sec

last uniq crash : none seen yet

last uniq hang : none seen yet

cycle progress

now processing : 261*1 (37.1%)

paths timed out : 0 (0.00%)

stage progress

new舞台 : splice 14

stage execs : 31/32 (96.88%)

total execs : 2.55M

exec speed : 61.2k/sec

fuzzing strategy yields

bit flips : n/a, n/a, n/a

byte flips : n/a, n/a, n/a

arithmetics : n/a, n/a, n/a

known ints : n/a, n/a, n/a

dictionary : n/a, n/a, n/a

splice/csplice : 506/1.05M, 193/1

custom : 0/0, 0/0

trim : 19.25%/53.2k, n/a

Who We Are

- Hackademics (both PhD students)

- CTFers

map coverage

map density

count coverage

findings in d

favored paths

new edges on

path length

path width

path geometry

levels : 11

pending : 121

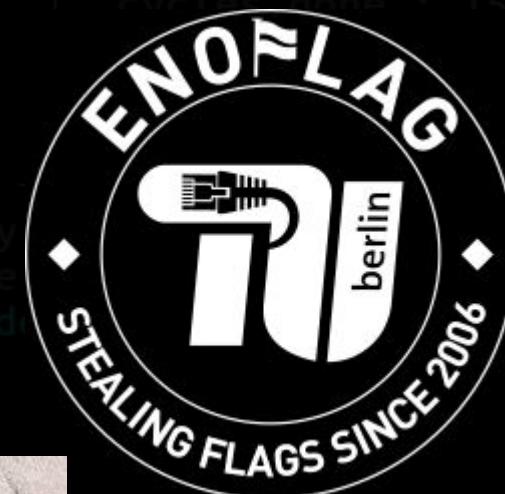
pend fav : 0

own finds : 699

imported : n/a

stability : 99.88%

[cpu000: 12%]



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
```

```
process timing
```

```
run time : 0 days, 0 hrs, 0 min, 43 sec
```

```
new paths : 0 days, 0 hrs, 0 min, 1 sec
```

```
last uniq crash : none seen yet
```

```
last uniq hang : none seen yet
```

```
cycle progress
```

```
now processing : 261*1 (37.1%)
```

```
paths timed out : 0 (0.00%)
```

```
stage progress
```

- Hackademics (both PhD students)

- CTFers

- Part of the AFL++ team



```
overall results
```

```
cycles done : 15
```

```
total paths : 703
```

```
uniq crashes : 0
```

```
uniq hangs : 0
```

```
map coverage
```

```
map density
```

```
count coverage
```

```
finding coverage
```

```
favorable coverage
```

```
new coverage
```

```
total coverage
```

```
total coverage
```

```
try
```

```
11
```

```
121
```

```
imported : 0
```

```
seconds : 699
```

```
imported : n/a
```

```
stability : 99.88%
```

```
[cpu000: 12%]
```



This Talk

We present a library for fuzzers that are

- **Fast** (low IPC, runtime overhead)

- **Scalable** (almost linearly to 200+ cores)

- **Portable** (Android, Windows, MacOS, Linux, Kernels, ...)

- **State-of-the-Art** (Hybrid-, Grammar-, Token-, Feedback-Fuzzing)

- **Multi-instrumentation** (binary-only Frida & Qemu, Clang, Python,...)

And, most importantly, **very extendable** with your own components.

We will show a simple, and a very advance example in code.



LibAFL

LibAFL, the fuzzer library.

Advanced Fuzzing Library - Slot your own fuzzers together and extend their features using Rust.

LibAFL is written and maintained by Andrea Fioraldi
andrea.fioraldi@gmail.com and Dominik Maier mail@dmnk.co.

Why LibAFL?

LibAFL gives you many of the benefits of an off-the-shelf fuzzer, while being completely customizable. Some highlight features currently include:

Contributors 26



+ 15 contributors

Languages



Unwatch 26 ★ Unstar 560 Fork 55

About

Advanced Fuzzing Library - Slot your fuzzers together in Rust! Scales across cores and machines. For Windows, Android, MacOS, Linux, no_std, ...

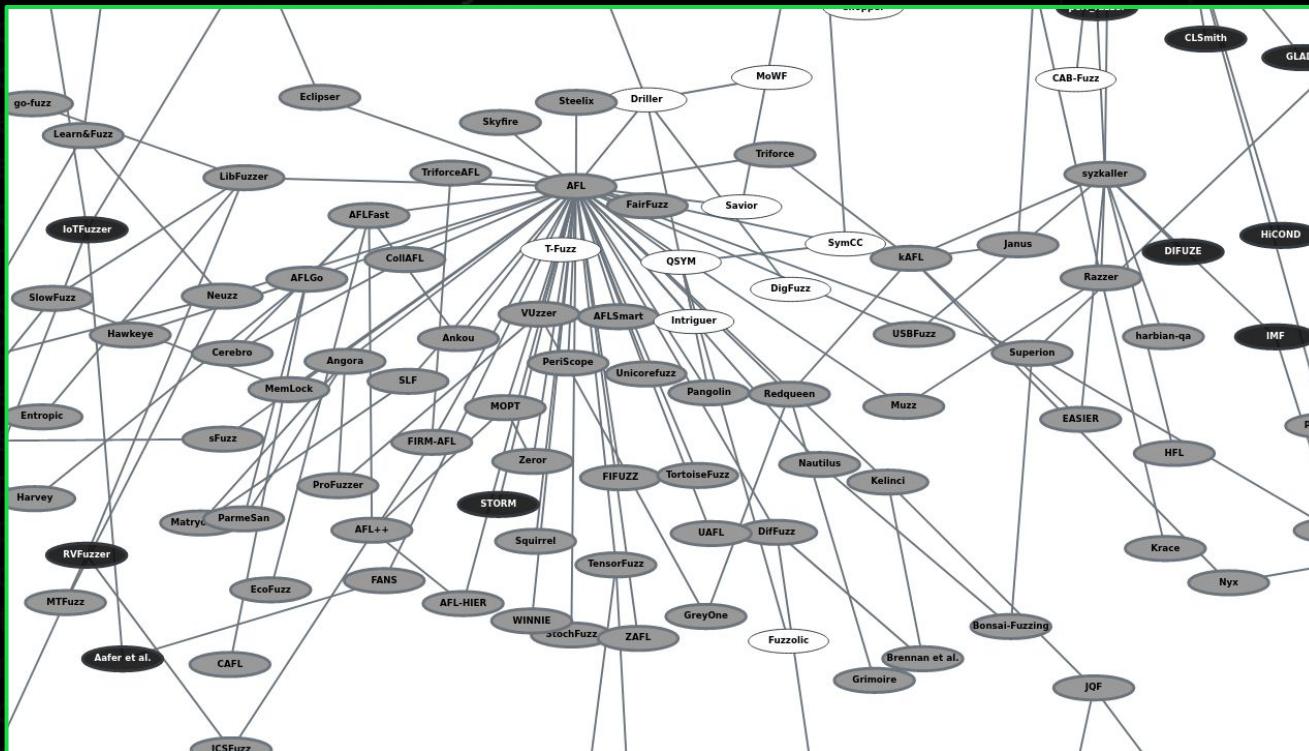
fuzzing afl afl-fuzz afplusplus frida coverage-guided

[Readme](#) [view license](#)

american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

Problem: Fuzzer Fragmentation



From <https://fuzzing-survey.org/>



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
    run time : 0 days, 0 hrs, 0 min, 43 sec
    start time: 2018-01-10 14:43:00
    last uniq crash: none seen yet
    last uniq hang: none seen yet
cycle progress
now processing : 261*1 (37.1%)
path timed out: 0 (0.00%)
stage progress
now trying : splice 14
stage progress: 20/20 (100.00%)
total execs : 2.55M
exec speed : 61.2k/sec
fuzzing strategy yields
bit flips : n/a, n/a, n/a
byte flips : n/a, n/a, n/a
array connects: n/a, n/a, n/a
block moves: n/a, n/a, n/a
dictionary : n/a, n/a, n/a
block/splice : 506/1.05M, 193/1.44M
block/custom: 0/0, 0/0
block/trim : 19.25%/53.2k, n/a
map coverage
map density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple
findings in depth
favored paths : 114 (16.22%)
edges on : 167 (23.76%)
total crashes : 0 (0 unique)
total tmouts : 0 (0 unique)
path geometry
levels : 11
pending : 121
pend fsv : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]
```

Cause: Monolithic Codebases

Fuzzers are

⇒ Designed to be tools

⇒ Not designed with code reuse in mind

⇒ Hard to extend

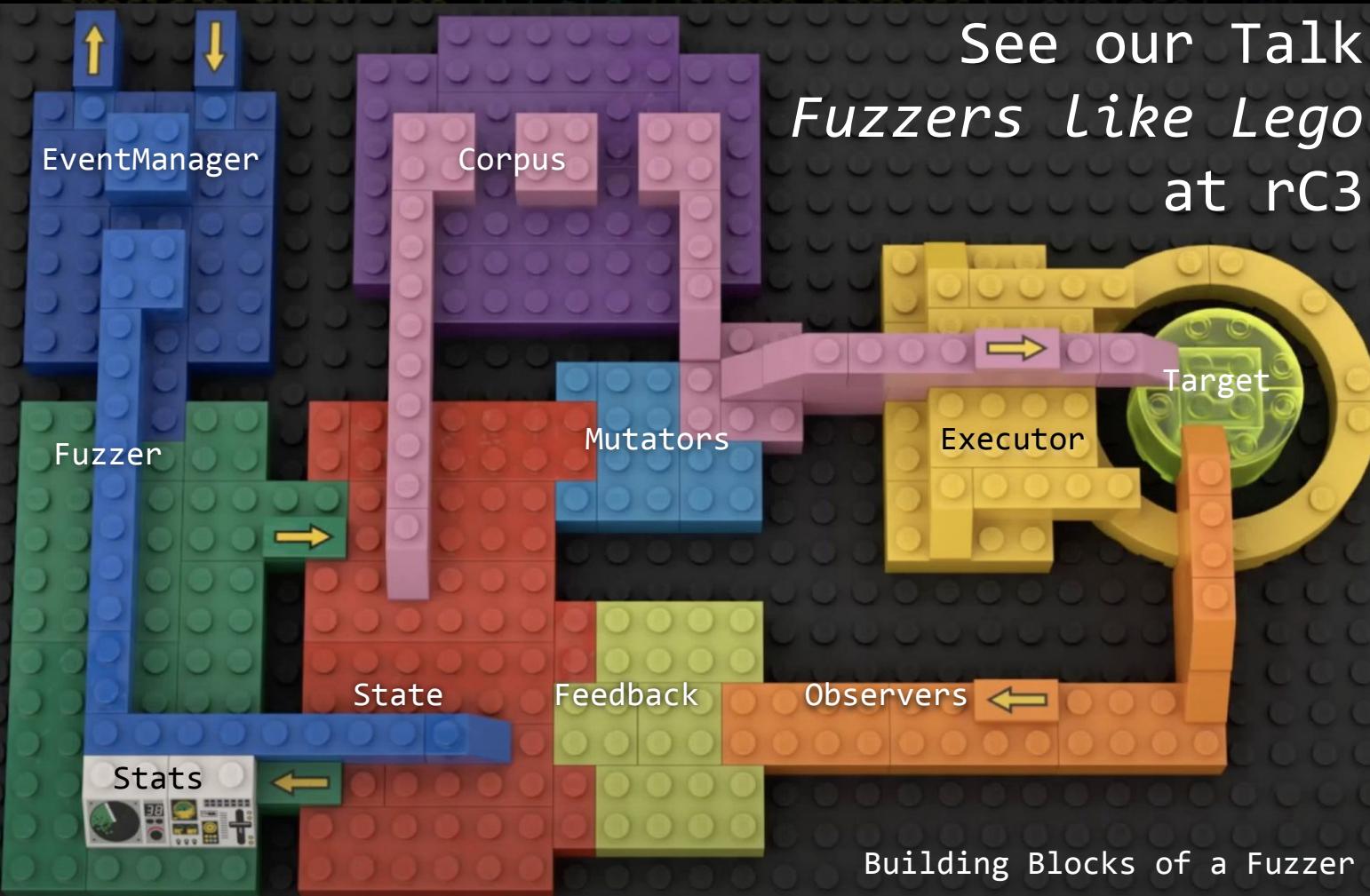
Many fuzzers are incompatible forks of others (usually AFL)

This makes them incompatible with orthogonal techniques



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}		
process timing		overall results
run time : 0 days, 0 hrs, 0 min, 43 sec		cycles done : 15
last new path : 0 days, 0 hrs, 0 min, 1 sec		total paths : 703
last uniq crash : none seen yet		uniq crashes : 0
last uniq hang : none seen yet		uniq hangs : 0
cycle progress		map coverage
now processing : 261*1 (37.1%)		map density : 5.78% / 13.98%
path coverage : 0.00%		count coverage : 3.30 bits/tuple
stage progress		findings in depth
now trying : splice 14		favored paths : 114 (16.22%)
stage execs : 3/1.2 (96.18%)		new evens : 13 (23.76%)
total execs : 2.55M		total crashes : 0 (0 unique)
exec speed : 61.2k/sec		total timeouts : 0 (0 unique)
fuzzing strategy yields		path geometry
bit flips : n/a, n/a, n/a		levels : 11
byte flips : n/a, n/a, n/a		pending : 121
arithmetics : n/a, n/a, n/a		pend fav : 0
known ints : n/a, n/a, n/a		own finds : 699
dictionary : n/a, n/a, n/a		imported : n/a
havoc/splice : 506/1.05M, 193/1.44M		stability : 99.88%
py/custom : 0/0, 0/0		
trim : 19.25%/53.2k, n/a		
	[cpu000: 12%]	

Solution: Abstract Fuzzing Concepts



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}	
process timing	overall results
run time : 0 days, 0 hrs, 0 min, 43 sec	cycles done : 15
last new path : 0 days, 0 hrs, 0 min, 1 sec	total paths : 703
last uniq crash : none seen yet	uniq crashes : 0
last uniq hang : none seen yet	uniq hangs : 0
cycle progress	map coverage
now processing : 261*1 (37.1%)	map density : 5.78% / 13.98%
paths timed out : 0 (0.00%)	count coverage : 3.30 bits/tuple
stage progress	findings in depth
now trying : splice 14	favored paths : 114 (16.22%)
stage execs : 37/1.96.88%	pending edges : 157 (23.76%)
total execs : 2.55M	total crashes : 0 (0 unique)
exec speed : 61.2k/sec	total timeouts : 0 (0 unique)
fuzzing strategy yields	path geometry
bit flips : n/a, n/a, n/a	levels : 11
byte flips : n/a, n/a, n/a	pending : 121
arithmetics : n/a, n/a, n/a	pend fav : 0
known ints : n/a, n/a, n/a	own finds : 699
dictionary : n/a, n/a, n/a	imported : n/a
havoc/splice : 506/1.05M, 193/1.44M	stability : 99.88%
py/custom : 0/0, 0/0	
trim : 19.25%/53.2k, n/a	
	[cpu000: 12%]

A fuzzer, step by step

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  new runs : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
  now processing : 261*1 (37.1%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice 14
  stage execs : 31/32 (96.88%)
  total execs : 2.55M
  exec speed : 61.2k/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  h/csplice : 506/1.05M, 193/1.44M
  /custom : 0/0, 0/0
  trim : 19.25%/53.2k, n/a
overall results
  cycles done : 15
  total paths : 703
  uniq crashes : 0
  uniq hangs : 0
map coverage
  map density : 5.78% / 13.98%
  count coverage : 3.30 bits/tuple
findings in depth
  favored paths : 114 (16.22%)
  new edges on : 167 (23.76%)
  total crashes : 0 (0 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 11
  pending : 121
  pend fav : 0
  own finds : 699
  imported : n/a
  stability : 99.88%
[cpu000: 12%]
```

The Function Under Test

Let's look at a simple function we want to fuzz, first.



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  new runs : 0
  last uniq crash : none seen yet
  last uniq hang : none seen yet
cycle progress
  now processing : 261 / 13,132
  paths timed out: 0
stage processes
  now trying : 14
  stage execs : 31 / 2,068 (82%)
total execs : 2.55M
exec speed : 61 {2k/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
// To test the panic:
// let input = BytesInput::new("abc".as_bytes());
// harness(&input); n/a
bit/c.splice : 506/1.05M, 193/1.44M
bit/c/custom : 0/0, 0/0
bit/c/trim : 19.25%/53.2k, n/a
overall results
  cycles done : 15
  total paths : 703
  uniq crashes : 0
  uniq hangs : 0
map coverage
  map density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple
findings in depth
  fav/total paths : 114 (16.22%)
  total crashes : 0 (0 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 11
  pending : 121
  pend fav : 0
  own finds : 699
  imported : n/a
  stability : 99.88%
[cpu000: 12%]
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
now processing : 261*1 (37.1%)
map coverage
map density : 5.78% / 13.98%
coverage : 3.30 bits/tuple
```

We will build the target and generate random input for it.

Let's see how to do this simple task in LibAFL.

```
stage progress
```

```
now trying : splice 14
stage execs : 31/32 (96.88%)
total execs : 2.55M
exec speed : 61.2k/sec
```

```
fuzzing strategy yields
```

```
bit flips : n/a, n/a, n/a
byte flips : n/a, n/a, n/a
arithmetics : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionary : n/a, n/a, n/a
splice/csplice : 506/1.05M, 193/1.44M
custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
```

```
findings in depth
```

```
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total crashes : 0 (0 unique)
total tmouts : 0 (0 unique)
```

```
path geometry
```

```
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
```

```
[cpu000: 12%]
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
now processing 261*1 (37.1%)
paths timed out : 0 (0.00%)
stage progress
now trying splice 14
stage exec 31/32 (96.8%)
total exec 2.5M
exec speed 61.2k/sec
fuzzing strategy
bit flip 1/1
byte flip : n/a, n/a, n/a
arithmetic : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionary : n/a, n/a, n/a
block/splice : 506/1.05M, 193/1.44M
custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
map coverage
map density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple
findings in depth
fuzzed paths : 114 (16.22%)
edges up : 167 (23.76%)
total crashes : 0 (0 unique)
total traces : 0 (0 unique)
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]
```

A Skeleton to Generate and Run Inputs

```
// The Stats trait define how the fuzzer stats are reported to the user
let stats = SimpleStats::new(|s| println!("{}", s));

// The event manager handle the various events generated during fuzzing
// such as the notification of the addition of a new item to the corpus
let mut mgr = SimpleEventManager::new(stats);
```



A Skeleton to Generate and Run Inputs

```
// A queue policy to get testcasess from the corpus
```

```
let scheduler = QueueCorpusScheduler::new()
```

// A fuzzer with feedbacks and a corpus scheduler

Let's start from the beginning.

let mut fuzzel = Stufuzz



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
now processing : 261*1 (37.1%)
paths timed out : 0 (0.00%)
stage processors
now trying : splice 14
stage execs : 31/32 (96.88%)
total execs : 2/55M
exec speed : 61.2k/sec
fuzzing strategy yields
bit flip : n/a, n/a, n/a
byte flips : n/a, n/a, n/a
arithmetics : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionary : n/a, n/a, n/a
splice/splice : 506/1.05M, 193/1.44M
custom/custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
map coverage
map density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple
findings in depth
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total crashes : 0 (0 unique)
total tmouts : 0 (0 unique)
path geometry
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
```

```
// Generator of printable bytearrays of max size 32
let mut generator = RandPrintablesGenerator::new(32);
```

```
stage progress
now trying splice 14
state : 31/32 (96.88%)
total exec : 2,551
exec speed : 61.2k/sec
fuzzing rate : 1.0M
bit flips : n/a, n/a, n/a
byte flips : n/a, n/a, n/a
arithmetics : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionary : n/a, n/a, n/a
splice : 506/1.05M, 193/1.44M
custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
```

map coverage

```
fan density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple
```

findings in depth

```
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total crashes : 0 (0 unique)
```

path geometry

```
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
```

[cpu000: 12%]



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
now processing : 261*1 (37.1%)
paths timed out : 0 (0.00%)
$ cargo run
Finished dev [unoptimized + debuginfo] target(s) in 0.04s
Running `target/debug/baby_fuzzer`
[LOG Debug]: Loaded 0 over 8 initial testcases
exec speed : 61.2k/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  boc/splice : 506/1.05M, 193/1.44M
  /custom : 0/0, 0/0
  trim : 19.25%/53.2k, n/a
map coverage
  map density : 5.78% / 13.98%
  count coverage : 3.30 bits/tuple
findings in depth
  edges on : 167 (23.76%)
  total : 0 (0 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 11
  pending : 121
  pend fav : 0
  own finds : 699
  imported : n/a
  stability : 99.88%
[cpu000: 12%]
```



A Skeleton to Generate and Run Inputs

Recap, we created a Fuzzer out of:

- a **State**: everything LibAFL keeps, including Corpus, Metadata, RNG state, ... - it get serialized and re-read on crash

- a **Stats** instance that simply prints the current stats. Advanced stats may print a complete afl-like screen.

- an **EventManager**: fires events, for example new Testcases and, depending on implementation, propagates them to other nodes.

- the **Executor**: this calls the Function under Test

- a **Generator**: Generators emit bytes, according to rules.



Evolving the Corpus With Feedbacks

We now have everything in place to call the function with random input.

However, modern fuzzing uses Feedback.

By observing the target, the fuzzer learns about interesting behavior.

It sees which Testcases are interesting, and can focus on them.

While most fuzzers instrument the target to get coverage as metric, for this simple fuzzer, we build it manually.





Evolving the Corpus With Feedbacks

american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

last uniq crash : none seen yet

last uniq hangs : none seen yet

// Coverage map with explicit assignments due to the lack of instrumentation

```
static mut SIGNALS: [u8; 16] = [0; 16];
fn signals_set(idx: usize) {
    unsafe { SIGNALS[idx] = 1 };
}
```

stage progress

now trying harnesses: 14 / 14 (100%)

stage execs: 312 / 1212 (25.6%)

total execs: 312 / 1212 (25.6%)

exec speed: 1.00M/s (0.00M/s)

fuzzing strategy: bit flips: n/a, byte flips: n/a, arithmetic: n/a, known ints: n/a, dictionary: n/a, doc/splice: 506/1.05M, 193/1.44M

ExitKind::Ok

overall results

cycles done : 15

days, hrs, min, sec : 0 days, 0 hrs, 0 min, 43 sec

uniq crashes : 0

uniq hangs : 0

map coverage

map density : 5.78% / 13.98%

count coverage : 3.30 bits/tuple

findings in depth

avored paths : 114 (16.22%)

new edges on : 167 (23.76%)

total crashes : 0 (0 unique)

total tmouts : 0 (0 unique)

path geometry

levels : 11

pending : 121

pend fav : 0

own finds : 699

imported : n/a

stability : 99.88%

[cpu000: 12%]



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

last uniq crash : none seen yet

last uniq hang : none seen yet

cycle progress

now processing : 261*1 (37.1%)

paths timer out : 0/0

stage progress

now trying : splice 14

stage passed : 31/32 (96.88%)

total exec : 2.5M

exec speed : 61.2k/sec

fuzzing rate

bit flips : n/a, n/a, n/a

byte flips : n/a, n/a, n/a

arithmetics : n/a, n/a, n/a

known ints : n/a, n/a, n/a

dictionary : n/a, n/a, n/a

splice/splice : 506/1.05M, 193/1.44M

custom/custom : 0/0, 0/0

trim : 19.25%/53.2k, n/a

map coverage

map density

findings in depth

focused paths

new edges on

total paths

total nodes

levels

pending

pend fav

own finds

imported

stability

[cpu000: 12%]

Evolving the Corpus With Feedbacks



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

last uniq crash : none seen yet

last uniq hang : none seen yet

cycle progress

now processing : 261*1 (37.1%)

paths timer out : 0/0

stage progress

now trying : splice 14

stage passed : 31/32 (96.88%)

total exec : 2.5M

exec speed : 61.2k/sec

fuzzing rate

bit flips : n/a, n/a, n/a

byte flips : n/a, n/a, n/a

arithmetics : n/a, n/a, n/a

known ints : n/a, n/a, n/a

dictionary : n/a, n/a, n/a

splice/splice : 506/1.05M, 193/1.44M

custom/custom : 0/0, 0/0

trim : 19.25%/53.2k, n/a

map coverage

findings in depth

focused paths : 114 (16.22%)

new edges on : 167 (23.76%)

total paths : 0 (0 unique)

path geometry

levels : 11

pending : 121

pend fav : 0

own finds : 699

imported : n/a

stability : 99.88%

[cpu000: 12%]

```
// Create an observation channel using the signals map
let observer = StdMapObserver::new("signals", unsafe { &mut SIGNALS });

// Create the executor for an in-process function with just one observer
let mut executor =
    InProcessExecutor::new(&mut harness, tuple_list!(observer),
                          &mut state, &mut mgr)
    .expect("Failed to create the Executor".into());
```



Evolving the Corpus With Feedbacks

```
// The state of the map feedback.  
let feedback_state = MapFeedbackState::with_observer(&observer);  
  
// Feedback to rate the interestingness of an input  
let feedback = MaxMapFeedback::new(&feedback_state, &observer);  
  
// A feedback to choose if an input is a solution or not  
let objective = CrashFeedback::new();
```



Evolving the Corpus With Feedbacks

```
// create a State from scratch
let mut state = StdState::new(
    // RNG
    StdRand::with_seed(current_nanos()),
    // The Corpus that will be evolved
    InMemoryCorpus::new(),
    // The Corpus in which we store solutions (crashes in this example), on
    // disk so the user can get them after stopping the fuzzer
    OnDiskCorpus::new(PathBuf::from("./crashes")).unwrap(),
    // States of the feedbacks, the data related to the feedbacks that you
    // want to persist in the State.
    tuple_list!(feedback_state),
);
// A fuzzer with feedbacks and a corpus scheduler
let mut fuzzer = StdFuzzer::new(scheduler, feedback, objective);
```



Evolving the Corpus With Feedbacks

```
// create a State from scratch
let mut state = StdState::new(
    // RNG
    StdRand::with_seed(current_nanos()),
    // The Corpus that will be evolved
    InMemoryCorpus::new(),
    // The Corpus in which we store solutions (crashes in this example), on
    // disk so the user can get them after stopping the fuzzer
    OnDiskCorpus::new(PathBuf::from("./crashes")).unwrap(),
    // States of the feedbacks, the data related to the feedbacks that you
    // want to persist in the State.
    tuple_list!(feedback_state),
);
// A fuzzer with feedbacks and a corpus scheduler
let mut fuzzer = StdFuzzer::new(scheduler, feedback, objective);
```



Evolving the Corpus With Feedbacks

Recap, we

- Manually added a coverage map to the target
- Added a *panic!* (rust for "crash") to the target
- Use a **MapObserver** that points LibAFL to this map
- Use a **MapMaxFeedback**: with it, the fuzzer tries to increase the map
 - Use a **CrashFeedback** that will tell the fuzzer when it has our crash
 - Added a **Corpus** for new interesting Testcases and one for Solutions
 - We store the MapFeedback's state in the fuzzer State

Observers and Feedbacks are decoupled, so we can use the same feedback for multiple targets, and even have feedbacks over multiple observers.



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}	
process timing	overall results
run time : 0 days, 0 hrs, 0 min, 43 sec	cycles done : 15
new paths : 0	total paths : 703
last uniq crash : none seen yet	uniq crashes : 0
last uniq hang : none seen yet	uniq hangs : 0
cycles progress	map coverage
now processing : 261*1 (37.1%)	map density : 5.78% / 13.98%
path coverage	st coverage : 3.30 bits/tuple
Loop the fuzzer until it finds a crash.	findings in depth
stage progress	favored paths : 114 (16.22%)
now trying : splice 14	new edges on : 167 (23.76%)
stage execs : 31/32 (96.88%)	total crashes : 0 (0 unique)
total execs : 2.55M	total tmouts : 0 (0 unique)
exec speed : 61.2k/sec	path geometry
fuzzing strategy yields	levels : 11
bit flips : n/a, n/a, n/a	pending : 121
byte flips : n/a, n/a, n/a	pend fav : 0
arithmetics : n/a, n/a, n/a	own finds : 699
known ints : n/a, n/a, n/a	imported : n/a
dictionary : n/a, n/a, n/a	stability : 99.88%
splice/csplice : 506/1.05M, 193/1.44M	
custom : 0/0, 0/0	[cpu000: 12%]
trim : 19.25%/53.2k, n/a	



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

run time : 0 days, 0 hrs, 0 min, 43 sec

min, 1 sec

last uniq crash : none seen yet

last uniq hang : none seen yet

cycle progress

now processing : 261*1 (37.1%)

paths timed out : 0 (0.00%)

// Setup a mutational stage with a basic bytes mutator

```
let mutator = StdScheduledMutator::new(havoc_mutations());
let mut stages = tuple_list!(StdMutationalStage::new(mutator));
fuzzer
    .fuzz_loop(&mut stages, &mut executor, &mut state, &mut mgr)
    .expect("Error in the fuzzing loop");
bit flips : n/a, n/a, n/a
byte flips : n/a, n/a, n/a
arithmetics : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionary : n/a, n/a, n/a
havoc/splice : 506/1.05M, 193/1.44M
havoc/custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
```

overall results

cycles done : 15

total paths : 703

uniq crashes : 0

uniq hangs : 0

map coverage

map density : 5.78% / 13.98%

count coverage : 3.30 bits/tuple

new edges on : 167 (23.76%)

total crashes : 0 (0 unique)

total timeouts : 0 (0 unique)

path geometry

levels : 11

pending : 121

pend fav : 0

own finds : 699

imported : n/a

stability : 99.88%

[cpu000: 12%]



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time: 0 days, 0 hrs, 0 min, 43 sec
min, 1 sec
last uniq crash: none seen yet
last uniq hang: none seen yet
cycle progress
$ cargo run --release
Compiling baby_fuzzer v0.1.0 (/home/andrea/Desktop/baby_fuzzer)
Finished release [optimized] target(s) in 1.56s
Running `target/release/baby_fuzzer` map coverage
[New Testcase] clients: 1, corpus: 2, objectives: 0, executions: 1, exec/sec: 0
[LOG Debug]: Loaded 1 over 8 initial testcases
[New Testcase] clients: 1, corpus: 3, objectives: 0, executions: 804, exec/sec: 0
[New Testcase] clients: 1, corpus: 4, objectives: 0, executions: 1408, exec/sec: 0
thread 'main' panicked at '==', src/main.rs:35:21
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
Crashed with SIGABRT
Child crashed!
[Objective] clients: 1, corpus: 4, objectives: 1, executions: 1408, exec/sec: 0
Waiting for broker...
Bye!
```



The Actual Fuzzing

```
$ cargo run --release
Compiling baby_fuzzer v0.1.0 (/home/andrea/Desktop/baby_fuzzer)
Finished release [optimized] target(s) in 1.56s
Running `target/release/baby_fuzzer`
[New Testcase] clients: 1, corpus: 2, objectives: 0, executions: 1, exec/sec: 0
[LOG Debug]: Loaded 1 over 8 initial testcases
[New Testcase] clients: 1, corpus: 3, objectives: 0, executions: 804, exec/sec: 0
[New Testcase] clients: 1, corpus: 4, objectives: 0, executions: 1408, exec/sec: 0
thread 'main' panicked at '='), src/main.rs:35:21
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
Crashed with SIGABRT
Child crashed!
[Objective] clients: 1, corpus: 4, objectives: 1, executions: 1408, exec/sec: 0
Waiting for broker...
Bye!
```

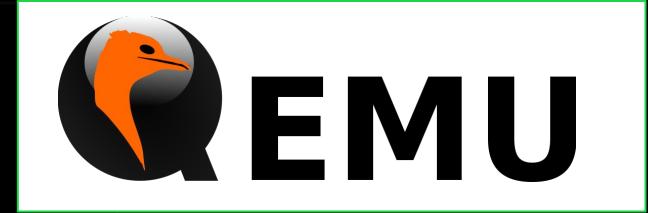
It crashes too fast to even calculate the execs/sec stat



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}		
process timing		overall results
run time : 0 days, 0 hrs, 0 min, 43 sec		cycles done : 15
last new path : 0 days, 0 hrs, 0 min, 1 sec		total paths : 703
last uniq crash : none seen yet		uniq crashes : 0
last uniq hang : none seen yet		uniq hangs : 0
cycle progress		map coverage
now processing : 261*1 (37.1%)		map density : 5.78% / 13.98%
paths timed out : 0 (0.00%)		count coverage : 3.30 bits/tuple
stage progress		findings in depth
now trying : splice 14		failed paths : 114 (16.22%)
stage execs : 3/32 (0%)		new paths : 16 (33.7%)
total execs : 2.55M		total crashes : 0 (0 unique)
exec speed : 61.2k/sec		total timeouts : 0 (0 unique)
fuzzing strategy yields		path geometry
bit flips : n/a, n/a, n/a		levels : 11
byte flips : n/a, n/a, n/a		pending : 121
arithmetics : n/a, n/a, n/a		pend fav : 0
known ints : n/a, n/a, n/a		own finds : 699
dictionary : n/a, n/a, n/a		imported : n/a
havoc/splice : 506/1.05M, 193/1.44M		stability : 99.88%
py/custom : 0/0, 0/0		
trim : 19.25%/53.2k, n/a		
		[cpu000: 12%]

A less useless (QEMU) fuzzer

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
    run time : 0 days, 0 hrs, 0 min, 43 sec
    new path : 0 days, 0 hrs, 0 min, 1 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
now processing : 261*1 (37.1%)
path coverage
now trying : splice 14
stage execs : 31/32 (96.88%)
total execs : 2.35M
fuzzing strategy yields
    bit flips : n/a, n/a, n/a
    byte flips : n/a, n/a, n/a
    arithmetics : n/a, n/a, n/a
    known ints : n/a, n/a, n/a
    dictionary : n/a, n/a, n/a
block/splice : 506/1.05M, 193/1.44M
custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
map coverage
map density : 5.78% / 13.98%
findings in depth
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total crashed : 0 (0 unique)
total tmouts : 0 (0 unique)
path geometry
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]
```



QEMU

QEMU is the Quick EMULATOR

We built an API around QEMU to set hooks (e.g. on each block execution or memory operation) and control the guest state.

You can load ELF's in libafel_qemu and fuzz binary-only with coverage, CmpLog, snapshots and more.



QEMU setup

```
// Initialize QEMU
let args: Vec<String> = env::args().collect();
let env: Vec<(String, String)> = env::vars().collect();
emu::init(&args, &env);

let mut elf_buffer = Vec::new();
let elf = EasyElf::from_file(emu::binary_path(), &mut elf_buffer).unwrap();

let test_one_input_ptr = elf
    .resolve_symbol("LLVMFuzzerTestOneInput", emu::load_addr())
    .expect("Symbol LLVMFuzzerTestOneInput not found");
println!("LLVMFuzzerTestOneInput @ {:#x}", test_one_input_ptr);

emu::set_breakpoint(test_one_input_ptr); // LLVMFuzzerTestOneInput
emu::run();
```



QEMU setup

```
// Initialize QEMU
let args: Vec<String> = env::args().collect();
let env: Vec<(String, String)> = env::vars().collect();
emu::init(&args, &env);

let mut elf_buffer = Vec::new();
let elf = EasyElf::from_file(emu::binary_path(), &mut elf_buffer).unwrap();
```

```
let test_one_input_ptr = elf
    .resolve_symbol("LLVMFuzzerTestOneInput", emu::load_addr())
    .expect("Symbol LLVMFuzzerTestOneInput not found");
println!("LLVMFuzzerTestOneInput @ {:#x}", test_one_input_ptr);
```

```
emu::set_breakpoint(test_one_input_ptr); // LLVMFuzzerTestOneInput
emu::run();
```



QEMU setup

```
// Get the return address
let stack_ptr: u64 = emu::read_reg(Amd64Regs::Rsp).unwrap();
let mut ret_addr = [0u64];
emu::read_mem(stack_ptr, &mut ret_addr);
let ret_addr = ret_addr[0];

println!("Stack pointer = {:#x}", stack_ptr);
println!("Return address = {:#x}", ret_addr);

emu::remove_breakpoint(test_one_input_ptr); // LLVMFuzzerTestOneInput
emu::set_breakpoint(ret_addr); // LLVMFuzzerTestOneInput ret addr
```

```
let input_addr = emu::map_private(0, 4096, MmapPerms::ReadWrite).unwrap();
```

```
println!("Placing input at {:#x}", input_addr);
```



[cpu000: 12%]

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
    run time : 0 days, 0 hrs, 0 min, 43 sec
    new runs : 0 days, 0 hrs, 0 min, 1 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycles coverage
now processing : 261113 / 13113
map density : 5.78% / 13.98%
paths timed out : 0 / 0
count coverage : 3.30 bits/tuple
stage processes
now trying : split 14 / 32 (43.75%)
stage execs : 31/32 (96.88%)
total execs : 2.55M
exec speed : 61.3k/sec
fuzzing structures
bit flip
byte flip
arithmetic
known ints : n/a, n/a, n/a
dictionary
hoc/split
custom : 0/0, 0/0
trim : 19.25%/53.2k, n/a
ExitKind: 0k 05M, 193/1.44M
};

// The wrapped harness function, calling out to the LLVM-style harness
let mut harness = |input: &BytesInput| {
    let target = input.target_bytes();
    let mut buf = target.as_slice();
    if buf.len() > 4096 {
        buf = &buf[0..4096];
    }
    emu::write_mem(input_addr, buf);
    emu::write_reg(Amd64Regs::Rdi, input_addr).unwrap();
    emu::write_reg(Amd64Regs::Rsi, len).unwrap();
    emu::write_reg(Amd64Regs::Rip, test_one_input_ptr).unwrap();
    emu::write_reg(Amd64Regs::Rsp, stack_ptr).unwrap();
    known ints : n/a, n/a, n/a
    dictionary
    hoc/split
    custom : 0/0, 0/0
    trim : 19.25%/53.2k, n/a
};

overall results
cycles done : 15
total paths : 703
uniq crashes : 0
uniq hangs : 0
findings in depth
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total crashes : 0 (0 unique)
total tmouts : 0 (0 unique)
path geometry
levels : 11
end fav : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
new paths, 0 hrs, 0 min, 1 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycles done : 15
now trying : 14
paths : 703
stage progress : 14 / 14
total execs : 313
exec speed : 14k/sec
fuzzing
bit flip : n/a, n/a, n/a
byte flip : n/a, n/a, n/a
arithmetic : n/a, n/a, n/a
known ints : n/a, n/a, n/a
dictionaries : n/a, n/a, n/a
doc/split : 0/0, 0/0
custom trim : 19.25%/53.2k, n/a
[cpu000: 12%]
overall results
cycles done : 15
total paths : 703
uniq crashes : 0
uniq hangs : 0
map density : 5.78% / 13.98%
average : 3.30 bits/tuple
findings in depth
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
crashes : 0 (0 unique)
total tmouts : 0 (0 unique)
path geometry
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]

// The shared memory allocator
let shmem_provider = StdShMemProvider::new().expect("Failed to init shared memory");
// The stats reporter for the broker
let stats = MultiStats::new(|s| println!("{}", s));
// Build and run a Launcher
match Launcher::builder()
    .shmem_provider(shmem_provider)
    .broker_port(broker_port)
    .configuration(EventConfig::from_build_id())
    .stats(stats)
    .run_client(&mut run_client)
    .cores(&cores)
    .stdout_file(Some("/dev/null"))
    .build()
    .launch()
{
    Ok(()) => (),
    Err(Error::ShuttingDown) => println!("Fuzzing stopped by user. Good bye."),
    Err(err) => panic!("Failed to run launcher: {:?}", err),
}
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last 1000 runs, 0 hrs, 0 min, 1 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycles done : 15
overall results
total paths : 703
uniq crashes : 0
uniq hangs : 0
map density : 5.78% / 13.98%
average : 3.30 bits/tuple
findings in depth
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total crashes : 0 (0 unique)
total thouts : 0 (0 unique)
path geometry
levels : 11
pending : 121
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
[cpu000: 12%]

// The shared memory allocator
let shmem_provider = StdShMemProvider::new().expect("Failed to init shared memory");
// The stats reporter for the broker
let stats = MultiStats::new(|s| println!("{}", s));
// Build and run a Launcher
match Launcher::builder()
    .shmem_provider(shmem_provider)
    .broker_port(broker_port)
    .configuration(EventConfig::from_build_id())
    .stats(stats)
    .run_client(&mut run_client)
    .cores(&cores)
    .stdout_file(Some("/dev/null"))
    .build()
    .launch()
{
    Ok(() ) => (),
    Err(Error::ShuttingDown) => println!("Fuzzing stopped by user. Good bye."),
    Err(err) => panic!("Failed to run launcher: {:?}", err),
}
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
run time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycles done : 15
overall results
total paths : 703
uniq crashes : 0
uniq hangs : 0
// The shared memory allocator
let shmem_provider = StdShMemProvider::new().expect("Failed to init shared memory");
// The stats reporter for the broker
let stats = MultiStats::new(|s| println!("{}", s));
// Build and run a Launcher
match Launcher::builder()
    .shmem_provider(shmem_provider)
    .broker_port(broker_port)
    .configuration(EventConfig::from_build_id())
    .stats(stats)
    .run_client(&mut run_client)
    .cores(&cores)
    .stdout_file(Some("/dev/null"))
    .build()
    .launch()
{
    Ok(()) => (),
    Err(Error::ShuttingDown) => println!("Fuzzing stopped by user. Good bye."),
    Err(err) => panic!("Failed to run launcher: {:?}", err),
}
```



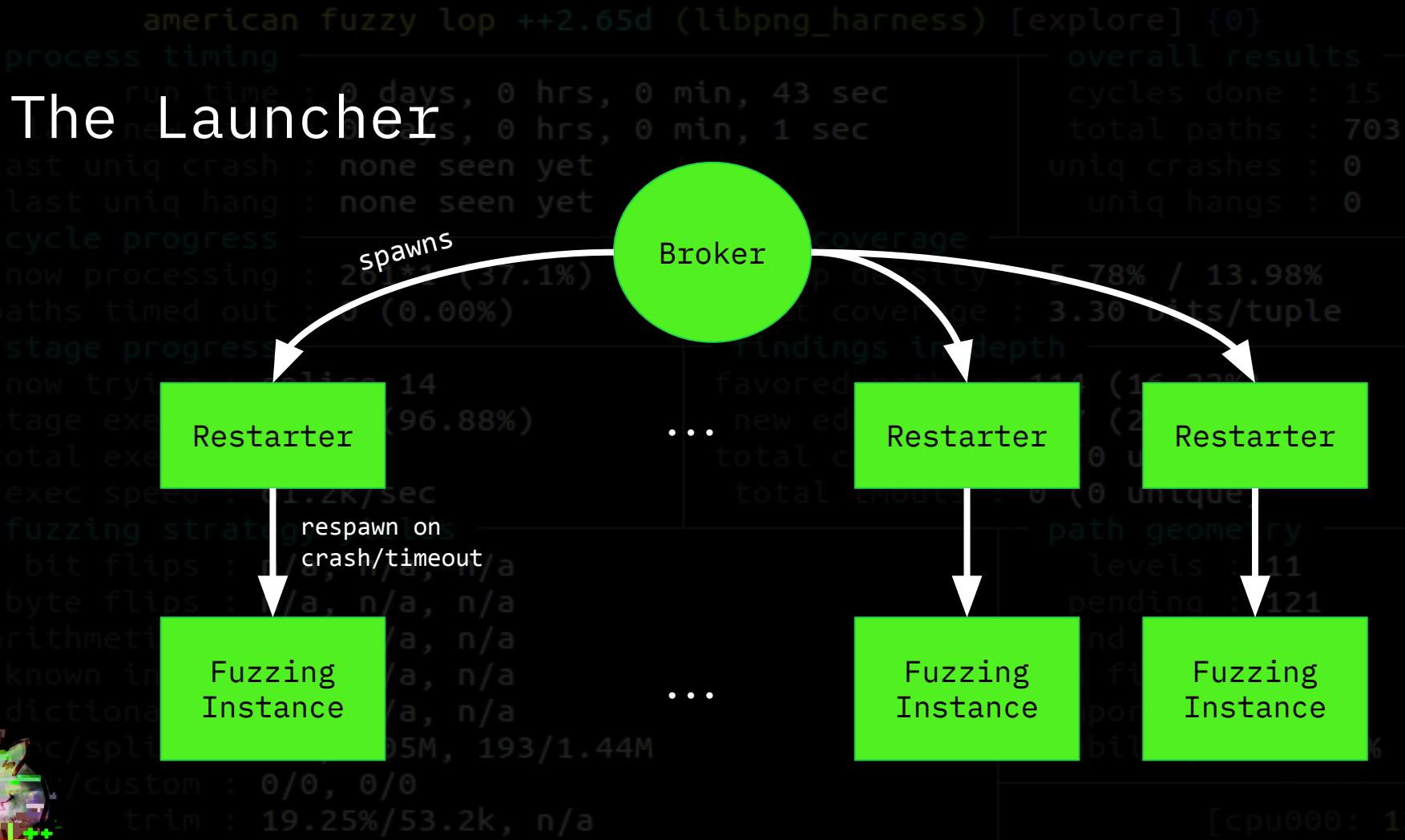
[cpu000: 12%]

The Launcher

```
// The shared memory allocator
let shmem_provider = StdShMemProvider::new().expect("Failed to init shared memory");
// The stats reporter for the broker
let stats = MultiStats::new(|s| println!("{}", s));
// Build and run a Launcher
match Launcher::builder()
    .shmem_provider(shmem_provider)
    .broker_port(broker_port) .configuration(EventConfig::from_build_id())
    .stats(stats)
    .run_client(&mut run_client)
    .cores(&cores)
    .stdout_file(Some("/dev/null"))
    .build()
    .launch()
{
    Ok(()) => (),
    Err(Error::ShuttingDown) => println!("Fuzzing stopped by user. Good bye."),
    Err(err) => panic!("Failed to run launcher: {:?}", err),
}
```



[cpu000: 12%]



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
```

```
process timing
```

```
run time : 0 days, 0 hrs, 0 min, 43 sec  
new paths, 0 hrs, 0 min, 1 sec
```

```
last uniq crash : none seen yet
```

```
last uniq hang : none seen yet
```

```
cycle progress
```

```
dataflow processing : 201*1 (37.1%)
```

```
paths timed out : 0 (0.00%)
```

```
stage progress
```

```
now trying collision 14 (96.88%)
```

```
stage exec : 0 (0.00%)
```

```
total exec : 0 (0.00%)
```

```
exec speed : 11.2k/sec
```

```
fuzzing strategy : 111us
```

```
bit flips : n/a, n/a
```

```
byte flips : n/a, n/a
```

```
arithmetic : n/a, n/a
```

```
known invariants : n/a, n/a
```

```
dictionary : n/a, n/a
```

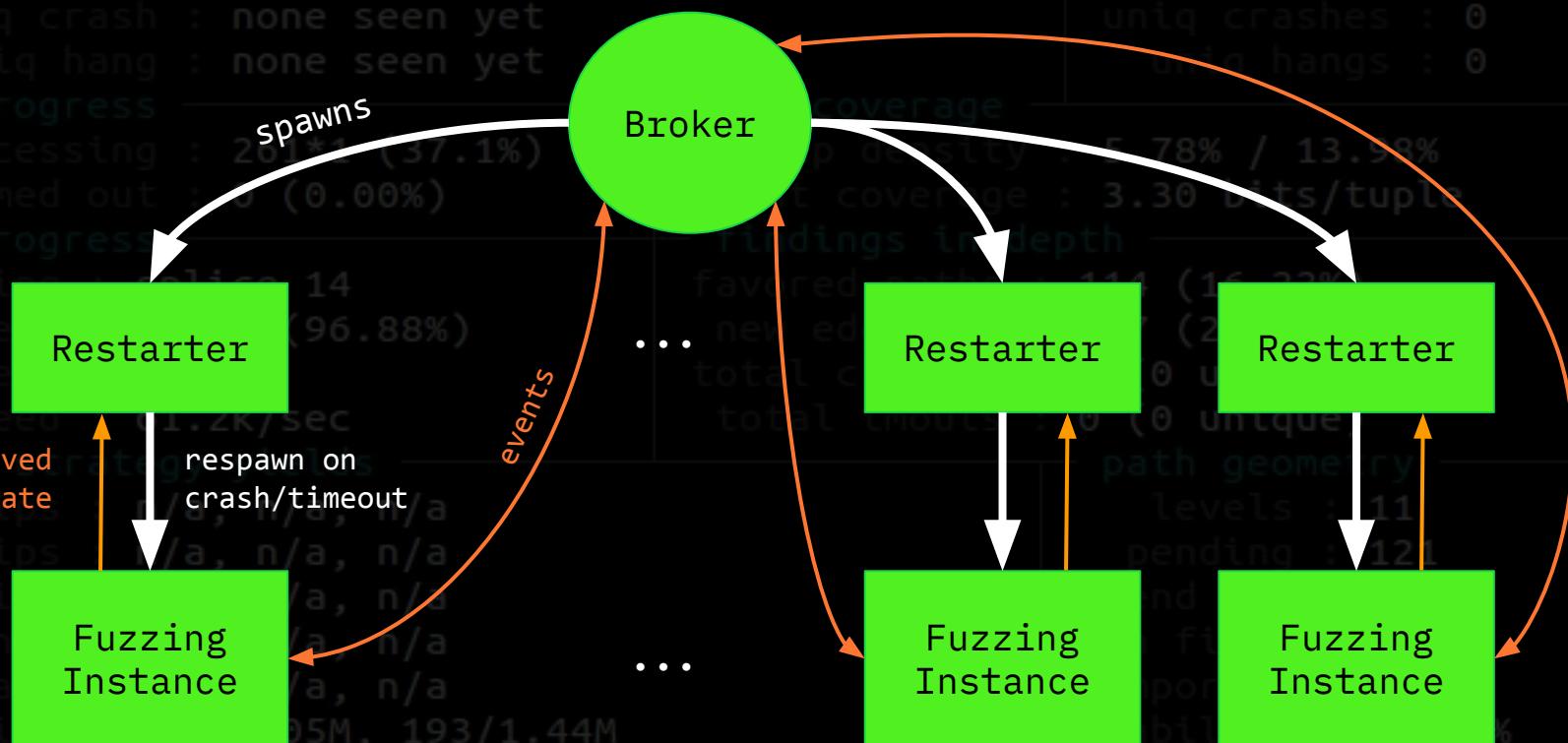
```
bufc/split : 0/0, 0/0
```

```
mem/custom : 0/0, 0/0
```

```
mem/trim : 19.25%/53.2k, n/a
```

```
[cpu000: 12%]
```

The Launcher



The Observers

```
let mut run_client = |state: Option<_>, mut mgr, _core_id| {
    // Create an observation channel using the coverage map
    let edges = unsafe { &mut hooks::EDGES_MAP };
    let edges_counter = unsafe { &mut hooks::MAX_EDGES_NUM };
    let edges_observer =
        HitcountsMapObserver::new(
            VariableMapObserver::new("edges", edges, edges_counter)
        );
    // Create an observation channel to keep track of the execution time
    let time_observer = TimeObserver::new("time");
}
```



[cpu000: 12%]

The Feedbacks

```
// The state of the edges feedback, in this case the coverage seen so far.  
let feedback_state = MapFeedbackState::with_observer(&edges_observer);  
  
// Feedback to rate the interestingness of an input  
// This one is composed by two Feedbacks in OR  
let feedback = feedback_or!(  
    // New maximization map feedback linked to the observer and the feedback state  
    MaxMapFeedback::new_tracking(&feedback_state, &edges_observer, true, false),  
    // Time feedback, this one does not need a feedback state  
    TimeFeedback::new_with_observer(&time_observer)  
);  
  
// A feedback to choose an input as solution if it is a crash or a timeout  
let objective = feedback_or_fast!(CrashFeedback::new(), TimeoutFeedback::new());
```



The State on restart

```
let mut run_client = |state: Option<_>, mut mgr, _core_id| {
    // ...
    // If not restarting, create a State from scratch
    let mut state = state.unwrap_or_else(|| {
        StdState::new(
            StdRand::with_seed(current_nanos()),
            InMemoryCorpus::new(),
            OnDiskCorpus::new(objective_dir.clone()).unwrap(),
            // The feedback state persist in the State
            bit flips: n/a,
            byte flips: n/a, n/a, n/a
        );
        arithmetic: n/a, n/a, n/a
        known ints: n/a, n/a, n/a
        dictionary: ..., n/a, n/a
    });
    bocsplice: 506/1.05M, 193/1.44M
    /custom: 0/0, 0/0
    trim: 19.25%/53.2k, n/a
}
```

process timing
run time: 0 days, 0 hrs, 0 min, 43 sec
last uniq crash: none seen yet
last uniq hang: none seen yet
cycles done: 15
total paths: 703
uniq crashes: 0
uniq hangs: 0

Map coverage
map density: 5.78% / 13.98%
count coverage: 3.30 bits/tuple

findings in depth

favorited paths: 114 (16.22%)

total edges: 167 (23.76%)

total crashes: 0 (0 unique)

total finds: 0 (0 unique)

path geometry

levels: 11

pending: 121

pend fav: 0

own finds: 699

imported: n/a

stability: 99.88%

[cpu000: 12%]



american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

run time : 6 days, 0 hrs, 0 min, 43 sec

last uniq crash : none seen yet

last uniq hang : none seen yet

cycle progress

now processing : 261*1 (37.1%)

paths timed out : 0 (0.00%)

// A minimization+queue policy to get testcases from the corpus

let scheduler = IndexesLenTimeMinimizerCorpusScheduler::new(QueueCorpusScheduler::new());

// A fuzzer with feedbacks and a corpus scheduler

let mut fuzzer = StdFuzzer::new(scheduler, feedback, objective);

fuzzing strategy yields

bit flips : n/a, n/a, n/a

byte flips : n/a, n/a, n/a

arithmetics : n/a, n/a, n/a

known ints : n/a, n/a, n/a

dictionary : n/a, n/a, n/a

splice : 506/1.05M, 193/1.44M

custom : 0/0, 0/0

trim : 19.25%/53.2k, n/a

map coverage

map density : 5.78% / 13.98%

count coverage : 3.30 bits/tuple

findings in depth

Favored paths : 114 (16.22%)

New edges on : 167 (23.76%)

Total edges : 0 (0 unique)

Total nodes : 0 (0 unique)

path geometry

levels : 11

Pending : 121

Pend Fav : 0

Own finds : 699

Imported : n/a

Stability : 99.88%

[cpu000: 12%]



The QEMU Executor

```
// Create a QEMU in-process executor
let executor = QemuExecutor::new(
    &mut harness,
    // The QEMU helpers define common hooks like coverage tracking hooks
    // In the specific, QemuEdgeCoverageHelper hooks every executed edge
    // for binary-only collision free edge coverage
    tuple_list!(QemuEdgeCoverageHelper::new()),
    tuple_list!(edges_observer, time_observer),
    &mut fuzzer,
    &mut state,
    &mut mgr,
)
.expect("Failed to create QemuExecutor");

// Wrap the executor to keep track of the timeout
let mut executor = TimeoutExecutor::new(executor, timeout);
```



The QEMU Executor

```
// Create a QEMU in-process executor
let executor = QemuExecutor::new(
    &mut harness,
    // The QEMU helpers define common hooks like coverage tracking hooks
    // In the specific, QemuEdgeCoverageHelper hooks every executed edge
    // for binary-only collision free edge coverage
    tuple_list!(QemuEdgeCoverageHelper::new()),
    tuple_list!(edges_observer, time_observer),
    &mut fuzzer,
    &mut state,
    &mut mgr,
)
.expect("Failed to create QemuExecutor");

// Wrap the executor to keep track of the timeout
let mut executor = TimeoutExecutor::new(executor, timeout);
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  new time : 0 days, 0 hrs, 0 min, 1 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
//...
cycle progress
now progress : 26111 (27.1%) paths timed : 0 (0.00%)
stage progress .load_initial_inputs(&mut fuzzer, &mut executor, &mut mgr, &corpus_dirs)
now trying : speedup : 1.00x
stage execs : 31/32 (96.8%) total execs : 2.55M
exec speed : 6.1k/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  // Setup an havoc mutator with a mutational stage
  let mutator = StdScheduledMutator::new(havoc_mutations());
  let mut stages = tuple_list!(StdMutationalStage::new(mutator));
fuzzer.fuzz_loop(&mut stages, &mut executor, &mut state, &mut mgr)?;
Ok(())
}
```

The last bits...

overall results
cycles done : 15
total paths : 703
uniq crashes : 0
uniq hangs : 0

map coverage
map density : 5.78% / 13.98%
count coverage : 3.30 bits/tuple

favored paths
new edges : 107 (25.70%)
total crashes : 0 (0 unique)
total tmoouts : 0 (0 unique)

path geometry
levels : 11
pending : 121
end fav : 0
end ends : 699
imported : n/a
stability : 99.88%

[cpu000: 12%]



```
$ cargo run --release ./libpng_harness
warning: CPU_TARGET is not set, default to x86_64
warning: Qemu not found, cloning with git (dd66ee9d0ec90221eb6476c63800f2671ad2a0c8)...
Compiling qemu_launcher v0.6.1 (/home/andrea/Desktop/LibAFL/fuzzers/qemu_launcher)
Finished release [optimized + debuginfo] target(s) in 27.84s
Running `target/release/qemu_launcher ./libpng_harness`
```

LLVMFuzzerTestOneInput @ 0x400000277b
Break at 0x400000277b
Stack pointer = 0x4001832908
Return address = 0x400000314f
Placing input at 0x4001de2000
spawning on cores: [0, 1, 2, 3]
...
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] "New connection" = "New connection"
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] addr = 127.0.0.1:59192
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] stream.peer_addr().unwrap() = 127.0.0.1:59192
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] "New connection" = "New connection"
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] addr = 127.0.0.1:59194
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] stream.peer_addr().unwrap() = 127.0.0.1:59194
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] "New connection" = "New connection"
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] addr = 127.0.0.1:59196
[/home/andrea/Desktop/LibAFL/libafl/src/bolts/llmp.rs:2191] stream.peer_addr().unwrap() = 127.0.0.1:59196
[Stats #3] (GLOBAL) clients: 4, corpus: 0, objectives: 0, executions: 0, exec/sec: 0
 (CLIENT) corpus: 0, objectives: 0, executions: 0, exec/sec: 0, edges: 752/752 (100%)
[Testcase #3] (GLOBAL) clients: 4, corpus: 1, objectives: 0, executions: 1, exec/sec: 0
 (CLIENT) corpus: 1, objectives: 0, executions: 1, exec/sec: 0, edges: 752/752 (100%)
[Stats #3] (GLOBAL) clients: 4, corpus: 1, objectives: 0, executions: 1, exec/sec: 0
 (CLIENT) corpus: 1, objectives: 0, executions: 1, exec/sec: 0, edges: 821/821 (100%)
[Testcase #3] (GLOBAL) clients: 4, corpus: 2, objectives: 0, executions: 2, exec/sec: 0
 (CLIENT) corpus: 2, objectives: 0, executions: 2, exec/sec: 0, edges: 821/821 (100%)



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
```

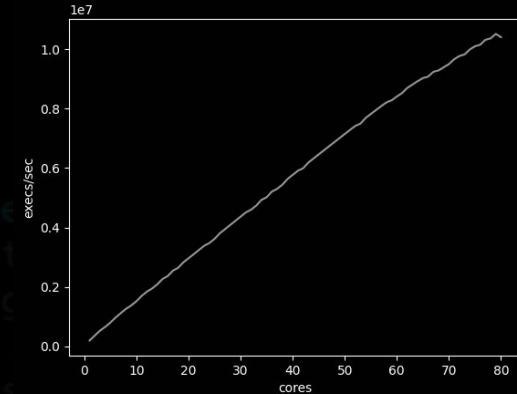
process timing

Scaling

- Little Kernel Load

- Fast message passing

- libpng: > 10mio execs/s on 80 cores



```
now trying : splice 14  
stage execs : 31/32 (96.88%)
```



```
Tasks: 268, 171 thr; 80 running  
Load average: 57.42 26.66 16.60  
Uptime: 6 days, 19:02:49
```



```
/custom : 0/0, 0/0
```

```
trim : 19.25%/53.2k, n/a
```

```
[cpu000: 12%]
```

american fuzzy lop ++2.65d (libpng_harness) [explore] {0}		
process timing		overall results
run time : 0 days, 0 hrs, 0 min, 43 sec		cycles done : 15
last new path : 0 days, 0 hrs, 0 min, 1 sec		total paths : 703
last uniq crash : none seen yet		uniq crashes : 0
last uniq hang : none seen yet		uniq hangs : 0
cycle progress	map coverage	
now processing : 261*1 (37.1%)	map density : 5.78% / 13.98%	
paths timed out : 0 (0.00%)	count coverage : 3.30 bits/tuple	
stage progress	findings in depth	
now trying : splice 14	fav pending : 114 (16.22%)	
stage execs : 3/32 9.06%	new pending : 16 (23.7%)	
total execs : 2.55M	total crashes : 0 (0 unique)	
exec speed : 61.2k/sec	total timeouts : 0 (0 unique)	
fuzzing strategy yields	path geometry	
bit flips : n/a, n/a, n/a	levels : 11	
byte flips : n/a, n/a, n/a	pending : 121	
arithmetics : n/a, n/a, n/a	pend fav : 0	
known ints : n/a, n/a, n/a	own finds : 699	
dictionary : n/a, n/a, n/a	imported : n/a	
havoc/splice : 506/1.05M, 193/1.44M	stability : 99.88%	
py/custom : 0/0, 0/0		
trim : 19.25%/53.2k, n/a		
	[cpu000: 12%]	

A less verbose (QEMU) fuzzer

american fuzzy lop ++2.65d (libpng_harness) [explore] {0}

process timing

last new paths seen yet

last unique paths seen yet

cycle progress

now processing: 25/32 (78.125%)

paths timed in: 25/32 (78.125%)

stage progress

now trying : splicer 14/32 (43.75%)

stage execs : 31/32 (96.88%)

total execs : 2.55M

exec speed : 61.21/sec

fuzzing strategy

bit flips : n/a

byte flips : n/a

arithmetics : n/a, n/a, n/a

known ints : n/a, n/a, n/a

dictionary : n/a, n/a, n/a

build() : 506/1.05M, 193/1.44M

run() : 0/0, 0/0

trim : 19.25%/53.2k, n/a

overall results

cycles done : 15

total paths : 703

uniq crashes : 0

uniq hangs : 0

map coverage

map density : 5.78% / 13.98%

count coverage : 3.30 bits/tuple

findings in depth

favored paths : 114 (16.22%)

new edges on : 167 (23.76%)

total crashes : 0 (0 unique)

total tncounts : 0 (0 unique)

path geometry

level : 11

pending fav : 121

pend fav : 0

own finds : 699

imported : n/a

stability : 99.88%

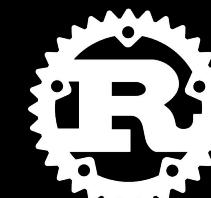
[cpu000: 12%]

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
```

```
process timing
```

Wait, so only Rust?

```
from pylibafl import sugar, qemu
# ...
def harness(b):
    if len(b) > MAX_SIZE:
        b = b[:MAX_SIZE]
    qemu.write_mem(inp, b)
    qemu.write_reg(qemu.amd64.Rsi, len(b))
    qemu.write_reg(qemu.amd64.Rdi, inp)
    qemu.write_reg(qemu.amd64.Rsp, sp)
    qemu.write_reg(qemu.amd64.Rip, test_one_input)
    qemu.run()
known ints : n/a, n/a, n/a
dict       : n/a, n/a, n/a
fuzz = sugar.QemuBytesCoverageSugar(['./corpus'], './crashes',
                                    1337, [0,1,2,3])
fuzz.run(harness)
```



```
overall results
cycles done : 15
crash found : 703
uniq hang    : 0
last uniq hang: none seen yet
last uniq hang: none seen yet
cycle progress
now processing : 261*1 (37.1%)
paths timed out : 0 (0.00%)
stage process
now trying to exec stage 14
stage execs : 31/32(96.875%)
total execs : 2,556
exec speed   : 61.2k/sec
fuzzing
bit flip     : 14
byte flip    : 14
arithmetic  : n/a, n/a
known ints  : n/a, n/a, n/a
dict        : n/a, n/a, n/a
fuzz = sugar.QemuBytesCoverageSugar(['./corpus'], './crashes',
                                    1337, [0,1,2,3])
fuzz.run(harness)
```

[cpu000: 12%]



Wait, so only Rust?

You can already use LibAFL from python!

(And yes, you can also use it to fuzz python ;))



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
```

```
process timing
```

LibAFL is FOSS!

LibAFL is young but the community is growing

```
now processing : 261*1 (37.1%)
```

```
paths : 2 GSOC students this summer
```

```
stages : 1 process
```

```
now trying : splice 14
```

```
stage execs : 31/32 (96.83%)
```

```
total execs : 2.55M
```

```
exec speed : 61.2k/sec
```

```
fuzzing strategy : yes/no
```

```
bit flips : n/a, n/a, n/a
```

```
byte flips : n/a, n/a, n/a
```

```
arithmetics : n/a, n/a, n/a
```

```
known ints : n/a, n/a, n/a
```

```
dictionary : n/a, n/a, n/a
```

```
splice/splice : 506/1.05M, 193/1.44M
```

```
custom/custom : 0/0, 0/0
```

```
trim : 19.25%/53.2k, n/a
```

```
overall results
```

```
cycles done : 15
```

```
total paths : 703
```

```
uniq crashes : 0
```

```
uniq hangs : 0
```

```
map coverage
```

```
map density : 5.78% / 13.98%
```

```
count coverage : 3.30 bits/tuple
```

```
findings in depth
```

```
favored paths : 114 (16.22%)
```

```
new edges on : 167 (23.76%)
```

```
total crashes : 0 (0 unique)
```

```
total timeouts : 0 (0 unique)
```

```
path geometry
```

```
levels : 11
```

```
pending : 121
```

```
pend fav : 0
```

```
own finds : 699
```

```
imported : n/a
```

```
stability : 99.88%
```

```
[cpu000: 12%]
```



LibAFL is FOSS!

LibAFL is young but the community is growing

- 2 GSOC students this summer

- 26 contributors

- >500 stars on GitHub

<https://github.com/AFLplusplus/LibAFL>

Fuzzers written in it already exist, for example <https://github.com/tlspuffin/tlspuffin>



[cpu000: 12%]

Other reasons to use LibAFL

- Easy-to-use Mutators

Platform independent Shared Map implementation (incl. Android)

- Serializable "AnyMap" implementation

Low-level Message Passing: Scalable Many-to-many communication

- QEMU bindings



[cpu000: 12%]

Conclusion

LibAFL is a library to build fuzzers

Example fuzzers outperform State-of-the-Art fuzzers in scalability and execution speed

Modern features like CmpLog, Grammar Fuzzing, Hybrid Fuzzing, Frida-Mode, Binary-Only ASAN

It's FOSS, with a growing community



[cpu000: 12%]

Thanks y'all

Have a nice FuzzCon Europe

