



# AFL++

## Combining Incremental Steps of Fuzzing Research

Andrea Fioraldi, Dominik Maier, Heiko Eifeldt, Marc Heuse

[@andrea](#), [@domenuk](#)

{andrea, dominik}@aflplusplus.com

[cpu000: 12%]

# American Fuzzy Lop

## american fuzzy lop 0.47b (readpng)

### process timing

run time : 0 days, 0 hrs, 4 min, 43 sec  
last new path : 0 days, 0 hrs, 0 min, 26 sec  
last uniq crash : none seen yet  
last uniq hang : 0 days, 0 hrs, 1 min, 51 sec

### cycle progress

now processing : 38 (19.49%)  
paths timed out : 0 (0.00%)

### stage progress

now trying : interest 32/8  
stage execs : 0/9990 (0.00%)  
total execs : 654k  
exec speed : 2306/sec

### fuzzing strategy yields

bit flips : 88/14.4k, 6/14.4k, 6/14.4k  
byte flips : 0/1804, 0/1786, 1/1750  
arithmetics : 31/126k, 3/45.6k, 1/17.8k  
known ints : 1/15.8k, 4/65.8k, 6/78.2k  
havoc : 34/254k, 0/0  
trim : 2876 B/931 (61.45% gain)

### overall results

cycles done : 0  
total paths : 195  
uniq crashes : 0  
uniq hangs : 1

### map coverage

map density : 1217 (7.43%)  
count coverage : 2.55 bits/tuple

### findings in depth

favorable paths : 128 (65.64%)  
new edges on : 85 (43.59%)  
total crashes : 0 (0 unique)  
total hangs : 1 (1 unique)

### path geometry

levels : 3  
pending : 178  
pend fav : 114  
imported : 0  
variable : 0  
latent : 0

8%  
ple

7  
1  
21  
99  
/a  
9.88%

trim : 19.25%/53.2k, n/a

[cpu000: 12%]



# American Fuzzy Lop

- A legendary tool that proved its effectiveness
- A baseline for a wide range of academic and industrial research
- No new features after 2017



# American Fuzzy Lop

- A legendary tool that proved its effectiveness
- A baseline for a wide range of academic and industrial research
- No new features after 2017

Fork it!



```
american_fuzzy_lop ++2.65d (1
process timing
  run time : 0 days, 0 hrs, 0 mi
  last new path : 0 days, 0 hrs, 0 mi
  last uniq crash : none seen yet
  last uniq hang : none seen yet
  cycle progress
    now progressing: 61*1 (97.1%)
  paths timed out : 0 (0.00%)
  stage progress
    now time: 54
  stage execs : 31/32 (96.88%)
  total execs : 1.55M
  total time: 61.20 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
    mc/splice : 506/1.05M, 193/1.44M
    /custom : 0/0, 0/0
    trim : 19.25%/53.2k, n/a
```

# A lot of Research Based on AFL

- AFLFast
  - AFLSmart
  - AFL LAF-Intel
  - AFL MOpt
  - kAFL
  - ...
  - Whatever-AFL
- 



Works on

# Fuzzer Scheduling

- Seed scheduling [AFLFast]

⇒ How much time should we fuzz a test case?

- Mutation scheduling [MOpt]

⇒ Probability for each mutational operator



# Works On Bypassing Roadblocks

- Feedback for comparisons [LAF-Intel]

⇒ Split multi-byte comparisons

- Input-to-state replacement [Redqueen (kAFL)]

⇒ Guess the input bytes that affect a comparison and replace it with the extracted token



# Structured Mutators

- Take input structure into account [AFLSmart]
  - Avoid to generate almost always invalid inputs
  - Stress more deep paths





# Speed Enhancements

- Reduce the number of instrumented program points while maintaining the same coverage [Instrim]
- Get rid of fork() and fuzz with snapshots [Opt-AFL]
- Inline instrumentation and re-enable TB linking in QEMU mode [abiondo-AFL]



# What if I Want to Use X AND Y?

- Orthogonal techniques not easy to combine
- Research fuzzers often unmaintained
- Some techniques are not implemented on top of the original AFL



# I created Z AND I want X

- If you peak one of the derived fuzzers as baseline you may be incompatible with other orthogonal techniques
- Hard to evaluate techniques without the relation with others (e.g. a new type of coverage without having a roadblock bypassing technique)



Here comes



# The AFL++ Project

- Integrates and reimplements fuzzing techniques in a single framework, AFL++
- Ongoing research and new insights about fuzzing using such framework
- We improve the state of the art combining techniques and tuning the implementations



# Usability

- All techniques are integrated in afl-fuzz
- Best-effort defaults
- Users familiar with AFL benefit from cutting-edge research without pain



# Extensibility

- To enable further research to do cross-comparisons with a reduced effort, we defined a set of API to extend AFL++, the *Custom Mutator API*



```
american_fuzzy_lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  total time : 0 days, 0 hrs, 0 min, 1 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice 14
stage execs : 31/32 (96.88%)
total execs : 1.5M
exec speed : 61.2k/sec
fuzzing fields
  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
  crc/splice : 506/1.05M, 193/1.44M
  /custom : 0/0, 0/0
  trim : 19.25%/53.2k, n/a
map coverage
  map coverage : 5.78% / 13.98%
count coverage : 3.30 bits/tuple
findings
  favored paths : 114 (16.22%)
  new edges on : 167 (23.76%)
  total tmouts : 0 (0 unique)
path geometry
  levels : 11
  pending : 121
  pend fav : 0
  own finds : 699
  imported : n/a
  stability : 99.88%
overall results
  cycles done : 15
  total paths : 703
  uniq crashes : 0
  uniq hangs : 0
[cpu000: 12%]
```

# Custom Mutator API

afl\_custom\_fuzz 51\*1 (37.1%)

afl\_custom\_post\_process

afl\_custom\_trim 14

afl\_custom\_havoc\_mutation

afl\_custom\_havoc\_mutation\_probability

afl\_custom\_queue\_get /a

afl\_custom\_queue\_new\_entry

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 hangs : 0 (0 unique)

path geometry

levels : 11

pending : 121

pend fav : 0

own finds : 699

imported : n/a

stability : 99.88%

[cpu000: 12%]





# INSTRUMENT ALL THE THINGS

- We extended techniques to work with other instrumentation backends.
- For Example: QEMU & Unicorn modes can split comparisons in a similar way to LLVM LAF-Intel
- Currently supported instrumentations are LLVM, QEMU, Unicorn, QBDI, GCC plugin, afl-gcc



# Runs on Everything

- AFL++ builds and runs on GNU/Linux, Android, iOS, macOS, FreeBSD, OpenBSD, NetBSD, IllumOS, Haiku, Solaris
- It is packaged in popular distributions like Debian, Ubuntu, NixOS, Arch Linux, FreeBSD, Kali Linux, ...



# Cross Evaluations

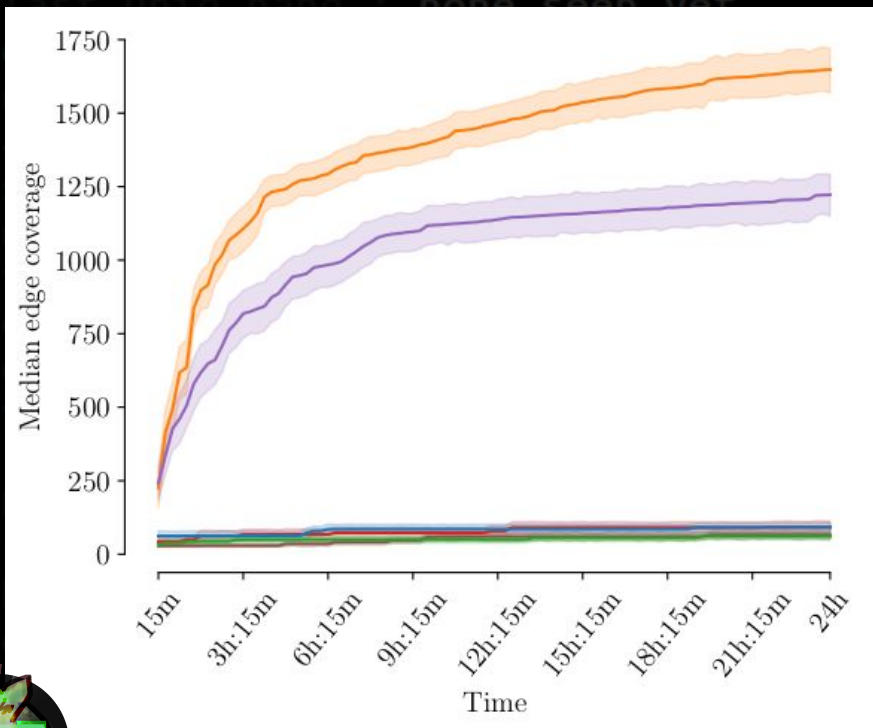
Using AFL++ as baseline gives you immediate access to cross evaluation of your technique combined with pre-existing works

Examples:

- [Default]
- Ngram4
- M0pt
- Redqueen



# Cross-Evaluations (libpcap)



Redqueen

Redqueen+MOpt

MOpt

Ngram4

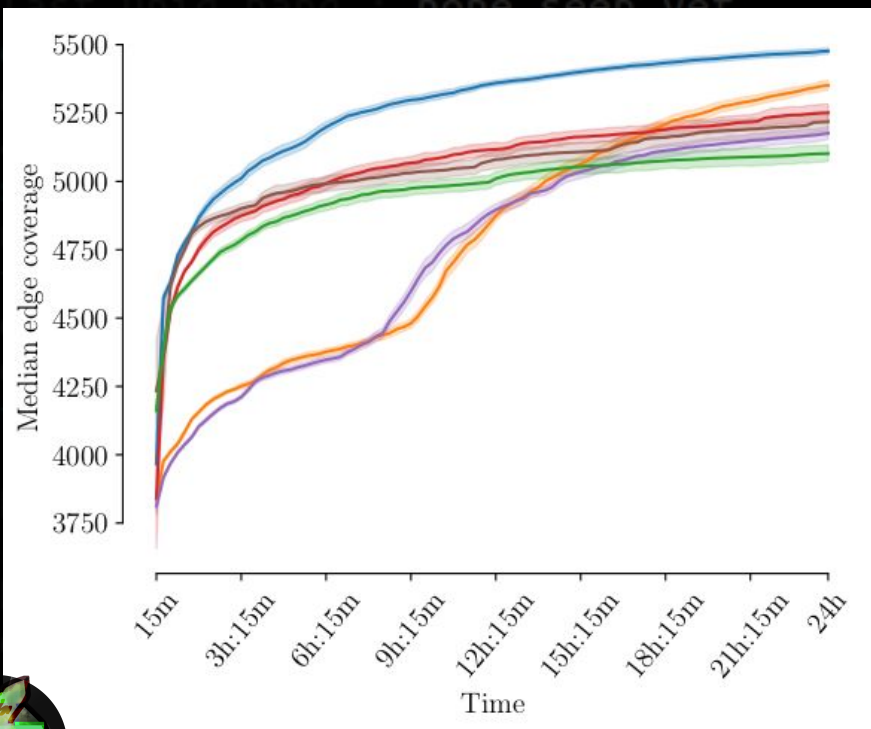
Ngram4+Rare

[Default]

```
map coverage
count coverage : 3.30 bits/tuple
favored paths : 114 (16.22%)
new edges on : 167 (23.76%)
total throuths : 0 (0 unique)
path geometry
levels : 11
pend fav : 0
own finds : 699
imported : n/a
stability : 99.88%
```



# Cross-Evaluations (bloaty)



● Redqueen

● Redqueen+MOpt

● MOpt

● Ngram4

● Ngram4+Rare

● [Default]



# Optimal Configuration

- Observe several runs of AFL++ in different configuration on the same target for a while
- Try to catch blind spots and select the best combination of features
- Profit



```
american_fuzzy_lop ++2.65d (libpng_harness) [explore] {0}
process timing
  run time : 0 days, 0 hrs, 0 min, 43 sec
  total time : 0 days, 0 hrs, 0 min, 43 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
cycle progress
  paths timed out : 0 (0.00%)
stage progress
  now trying : splice_14
stage execs : 31/32 (96.88%)
total execs : 2000000
exec speed : 61.2k/sec
fuzzing options
  bit flips : n/a, n/a, 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, n/a
  crc/splice : 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
  count coverage : 3.30 bits/tuple
  favored paths : 114 (16.22%)
  new edges on : 167 (23.76%)
  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%]
```

# Future work

- Static analysis for optimal fuzz settings
- Multicore linear scaling
- Plugin system (executors, queues, feedbacks, ...)
- Collision-free instrumentation



```
american_fuzzy_lop ++2.65d (libpng_harness) [explore] {0}
```

```
process timing
```

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

```
total time : 0 days, 0 hrs, 0 min, 1 sec
```

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

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

```
cycle progress
```

```
now trying : splice 14
```

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

```
stage execs :
```

```
now trying : splice 14
```

```
stage execs : 31/32 (96.88%)
```

```
total time :
```

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

```
fuzzing options
```

```
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
```

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

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

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

```
overall results
```

```
cycles done : 15
```

```
total paths : 703
```

```
uniq crashes : 0
```

```
uniq hangs : 0
```

```
map coverage
```

```
count coverage : 13.98%
```

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

```
findings in depth
```

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

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

```
total ttrouts : 0 (0 unique)
```

```
total ttrouts : 0 (0 unique)
```

```
path geometry
```

```
levels : 11
```

```
pending : 121
```

```
pend fav : 0
```

```
own finds : 699
```

```
imported : n/a
```

```
stability : 99.88%
```

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

# Conclusion

- AFL++ enhances comparability of research
- We further improve the state-of-the-art with speed, usability, new features
- AFL++'s custom mutator API can be used to implement novel research in a maintainable way





AFL++ is FOSS!

american\_fuzzy\_lop ++2.65d (libpng\_harness) [explore] {0}

process timing

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

time left : 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 <https://aflplus.plus/> 114 (16.22%)

stage execs : 31/32 (96.88%)

total execs : 2.55M

exec speed : 61.24k/sec <https://github.com/AFLplusplus> 0 (0 unique)

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

libc/splice : 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

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%]



Thank you for  
your attention.

