American Fuzzy Lop

Introduction & technical presentation

What is “fuzz testing” ?

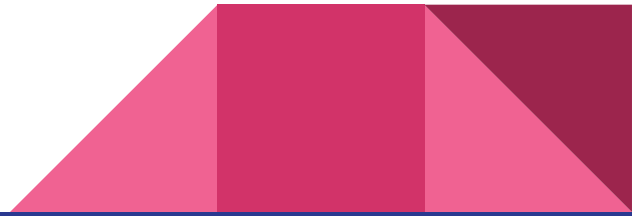
What is “fuzz testing” ?

- **A way to test software products**
  - test safety: try to find out lot of crash cases that you have not imagined yet

  - test security : in case of crash, be sure user cannot use your soft to break some things around

What is “fuzz testing” ?

- **The way :**
  - providing unexpected, invalid or random data
    - usual inputs : keyboard, mouse, API calls
    - unusual input but think to it : databases, SHM and all other ways I didn’t think of yet



Two types of fuzzing

- Generation based

- Mutation based

American Fuzzy Lop

AFL in few words

Mutation based fuzzer

Genetic algorithm

High efficiency

Code instrumentation

note : every “cf.” notifications, refer to the documentation of afl given with the package afl-2.10b from this link.

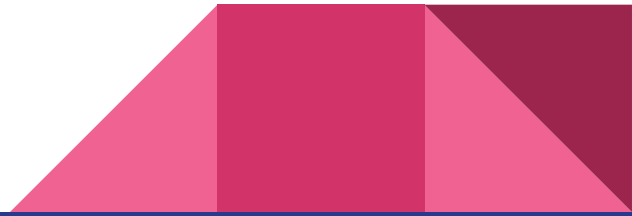
In which case can it be used ?

For white box usage

- 1st step : compile your soft with AFL’s compilators (based on gcc or clang)
- 2nd step : launch your test
- 3rd step : explore found crash cases

For black box usage

- instrumentation possible under QEMU
- 2-5x slower than classic instrumentation
- some issues with parallelism



Compilation

2 levels of code instrumentation for target binary

classic : intrumented at assembly level (near native execution speed)

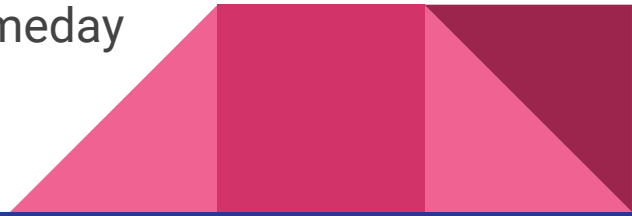
afl-gcc & afl-g++

afl-clang & afl-clang++

fast : *clang only*, intrumented at compiler level (+/-10% better than native speed)

afl-clang-fast ->should replace afl-clang someday

afl-clang-fast++



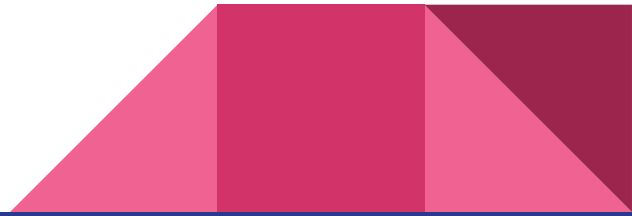
What’s needed to run some tests ?

Must provide

- One or few ***valid*** input case file(s)

Keep it simple

- ***Shortest files***, are better to get good perfs (in term of test efficiency)



What’s needed to run some tests ?

For a good efficiency

A piece of software

- to test a very specific thing
- it should be think and written to be fast

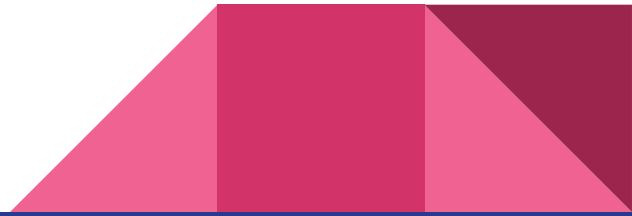
Execution

Command line : level 1

basic : give **input files** on **stdin**

afl-fuzz -i valid_inputs_folder/ **-o** afl_output_folder/ ./binary_to_test

AFL simply **send inputs** on **stdin** of your binary



Execution

Command line : level 1

basic : give **input files** as **parameter**

afl-fuzz -i valid_inputs_folder/ **-o** afl_output_folder/ ./binary_to_test @@

@@ symbolize the **place** of the **input file** in your command

Execution

Command line : level 2

Some **options** :

-x a_dictionary : put a dictionary to help fuzzer

```
# Lines starting with '#' and empty lines are ignored.

# Adds "blah" (w/o quotes) to the dictionary.
kw1="blah"
# Use \\ for backslash and \" for quotes.
kw2="\"ac\\dc\""
# Use \xAB for hex values
kw3="\xF7\xF8"
# the name of the keyword followed by '=' may be omitted:
"foo\x0Abar"
```

image from : http://llvm.org/docs/LibFuzzer.html

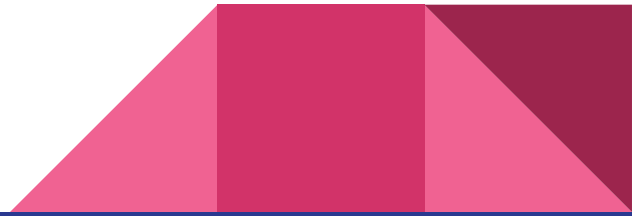
Execution

Command line : level 2

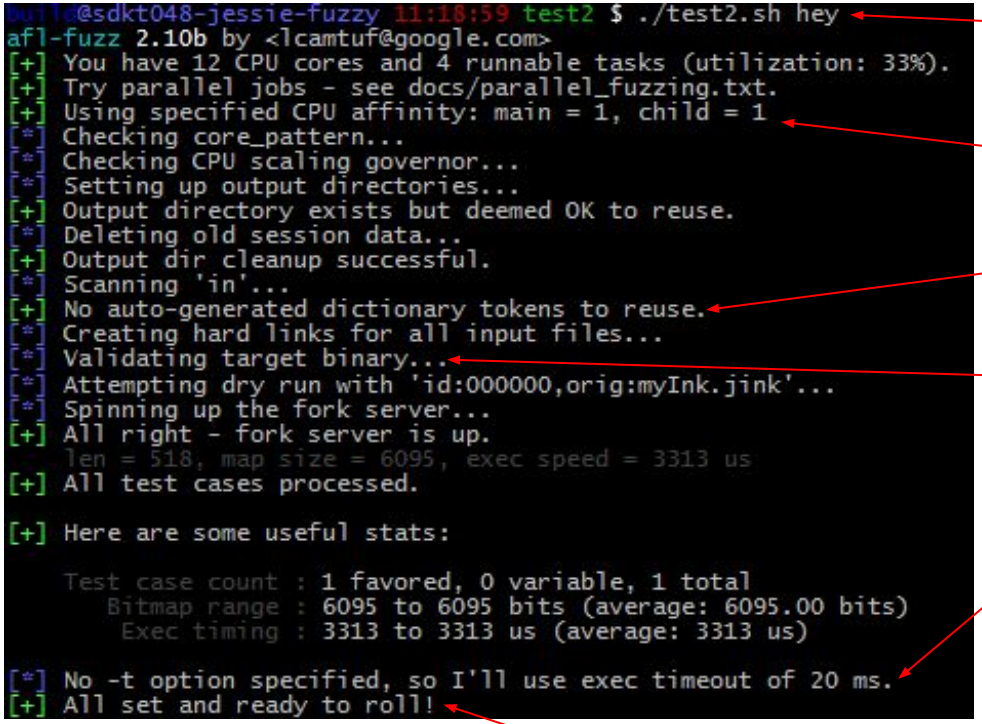
Some **options** :

-T 50 : set timeout to : 5x reference time + 50 ms

(basic timeout is : 5x reference time + 20 ms)



And then, pull the trigger...



(my script just want name as param do not pay attention)

one core selected

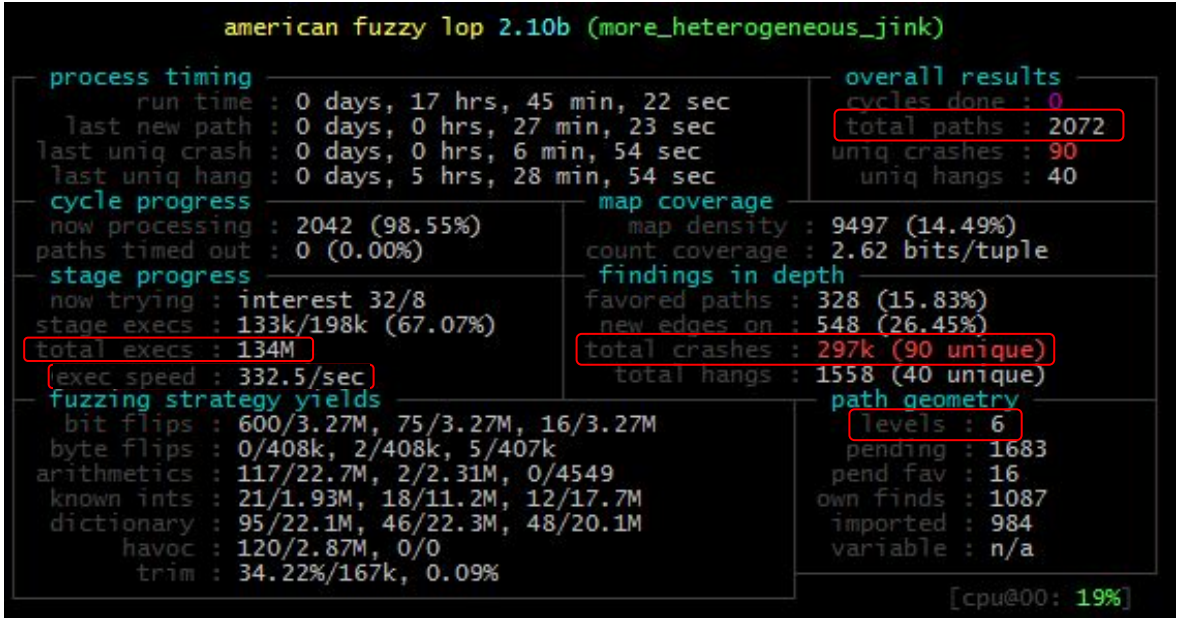
no **-x** given

test binary with given input case(s)

what it's written

Everything is ok !

And then, pull the trigger...



total path : unique call stacks found

total execs : number of tests done

“90 unique” : means 90 differents ???

levels : deepness of the mutation

exec speed : number of tests / sec

cf. : status_screen.txt

AFL output format

The output/ folder

```
build@sdkt048-jessie-fuzzy 11:20:45 math_1_simple_file_no_persistent $ ll
total 1056
drwx------ 2 build build  24576 Jul 11 05:24 crashes
-rw------- 1 build build  65536 Jul 11 09:49 fuzz_bitmap
-rw------- 1 build build    873 Jul 11 11:20 fuzzer_stats
drwx------ 2 build build  24576 Jul 11 11:08 hangs
-rw------- 1 build build 605909 Jul 11 11:18 plot_data
drwx------ 3 build build 266240 Jul 11 09:49 queue
```

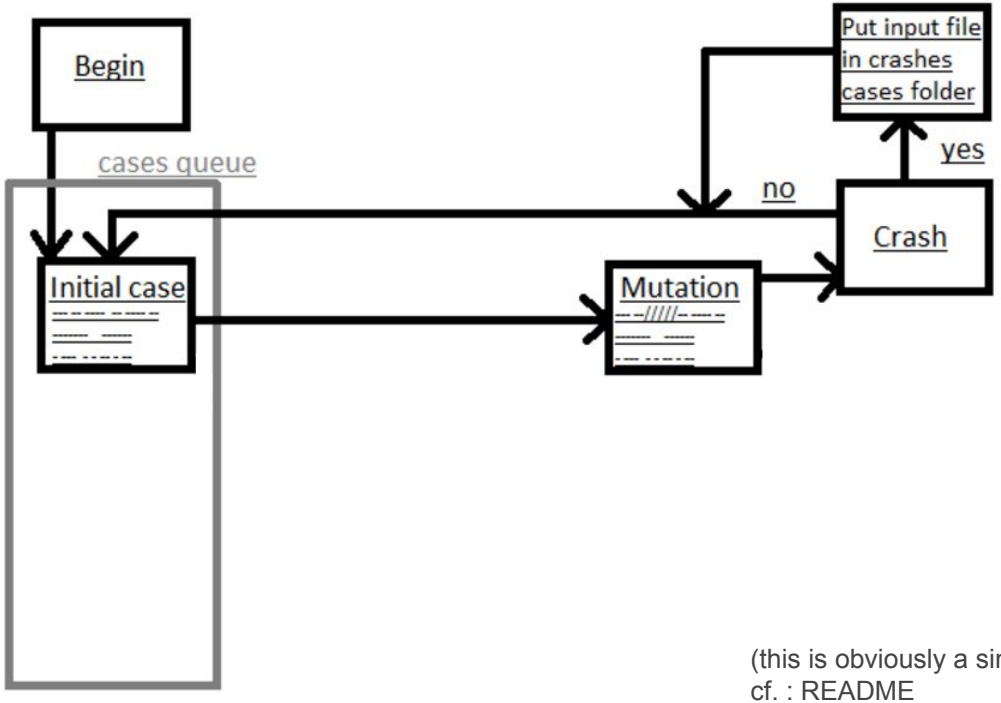
AFL output format

Folder content

```
-rw------- 1 build build 192 Jul  5 17:37 id:000347,sig:11,src:004768,op:havoc,rep:2
-rw------- 1 build build 195 Jul  5 17:38 id:000348,sig:11,src:004768,op:havoc,rep:2
-rw------- 1 build build 189 Jul  5 17:56 id:000349,sig:11,src:004797,op:flip1,pos:31
-rw------- 1 build build 189 Jul  5 17:57 id:000350,sig:11,src:004797,op:flip4,pos:5
-rw------- 1 build build 189 Jul  5 17:57 id:000351,sig:11,src:004797,op:flip4,pos:12
-rw------- 1 build build 189 Jul  5 17:57 id:000352,sig:11,src:004797,op:flip4,pos:77
-rw------- 1 build build 189 Jul  5 17:57 id:000353,sig:11,src:004797,op:flip4,pos:108
-rw------- 1 build build 189 Jul  5 17:57 id:000354,sig:11,src:004797,op:arith8,pos:14,val:+6
-rw------- 1 build build 189 Jul  5 17:57 id:000355,sig:11,src:004797,op:arith8,pos:27,val:+13
-rw------- 1 build build 189 Jul  5 17:58 id:000356,sig:11,src:004797,op:arith8,pos:43,val:+14
-rw------- 1 build build 189 Jul  5 17:58 id:000357,sig:11,src:004797,op:arith8,pos:101,val:-5
-rw------- 1 build build 189 Jul  5 17:58 id:000358,sig:11,src:004797,op:arith8,pos:108,val:-7
-rw------- 1 build build 189 Jul  5 18:00 id:000359,sig:11,src:004797,op:int16,pos:187,val:be:-128
-rw------- 1 build build 189 Jul  5 18:01 id:000360,sig:11,src:004797,op:ext_AO,pos:12
-rw------- 1 build build 189 Jul  5 18:01 id:000361,sig:11,src:004797,op:ext_AO,pos:108
-rw------- 1 build build 192 Jul  5 18:05 id:000362,sig:11,src:004797,op:havoc,rep:2
-rw------- 1 build build 192 Jul  5 18:05 id:000363,sig:11,src:004797,op:havoc,rep:2
```

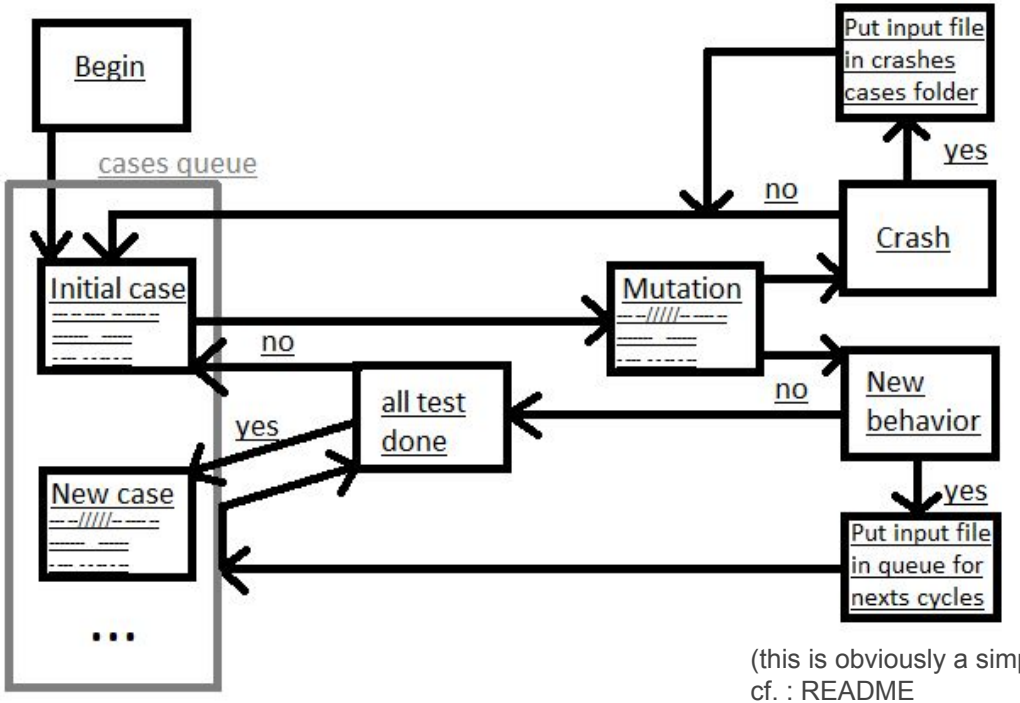
Dive more into AFL & its ecosystem

Nice ! It works :)
But… what's really happening ?



(this is obviously a simplistically draw)
cf. : README

Nice ! It works :)
But… what's really happening ?



(this is obviously a simplistically draw)
cf. : README

Mutations ? What’s mutation ?

- **bitflip L/S**
- **arith L/8**
- **interest L/8**
- **havoc**
- **slice**
- **extras**

cf. : status_screen.txt

Multi thread tests

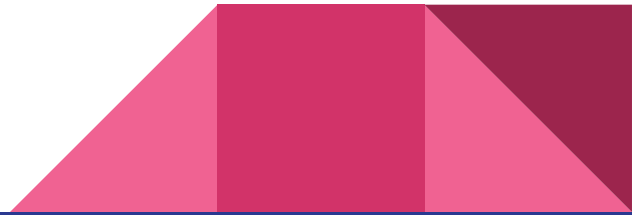
For **multi thread** test :

-M a_name : specify ***master*** fuzzer (deterministic)

-S an_other_name : specify ***slave*** fuzzer (random)

Only one master and as many slaves as you want

All fuzzers must **share** the same **input_folder** as well as the **output_folder**.



One more thing ...

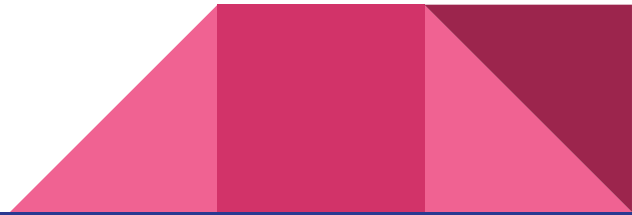
Software with **heavy initial load** can **get huge performance boost** with :

AFL persistent mode

For me, it make the fuzzer increase by **1000%** the average speed execution.

From 300 to 3000 executions / second…

It’s easy to set up, just go read README.llvm.



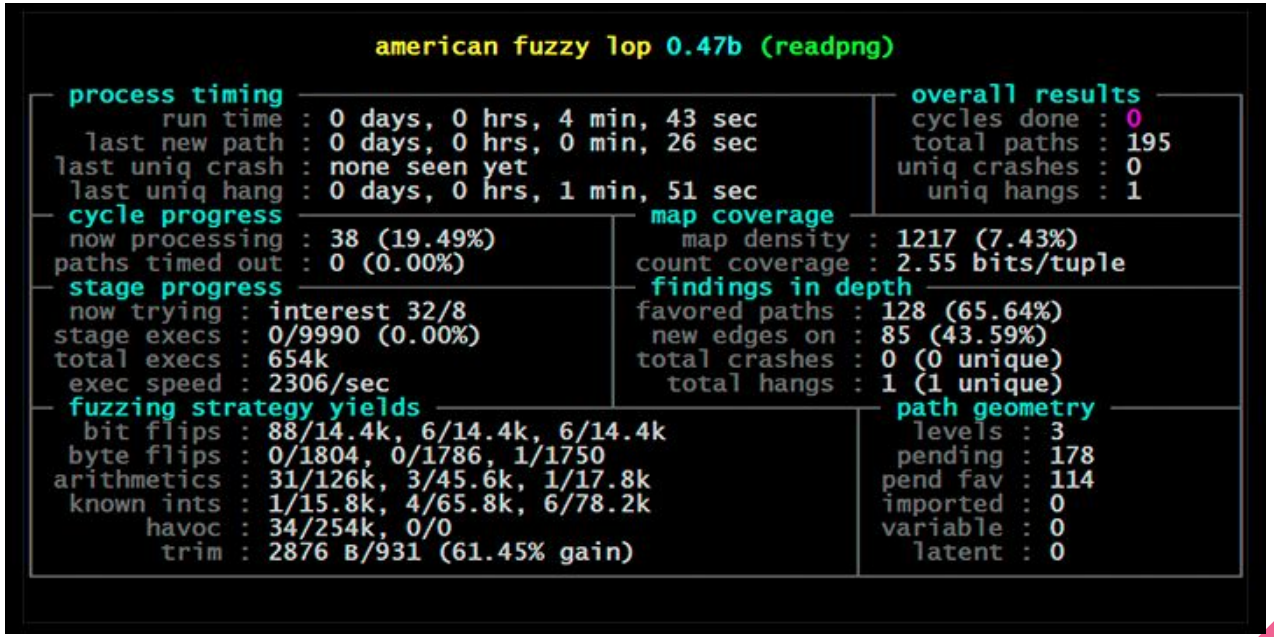
Persistent mode

```
int main(int argc, char** argv)
{
        while (__AFL_LOOP(1000))
        {
                * Reset state. */
                memset(buf, 0, 100);

                /* Read input data. */
                read(0, buf, 100);

                /* Parse it in some vulnerable way. You'd normally call a library here.
*/
                if (buf[0] != 'p') puts("error 1"); else
                 if (buf[1] != 'w') puts("error 2"); else
                if (buf[2] != 'n') puts("error 3"); else
                        abort();
        }
}
```

exemple from : https://lcamtuf.blogspot.se/2015/06/new-in-afl-persistent-mode.html

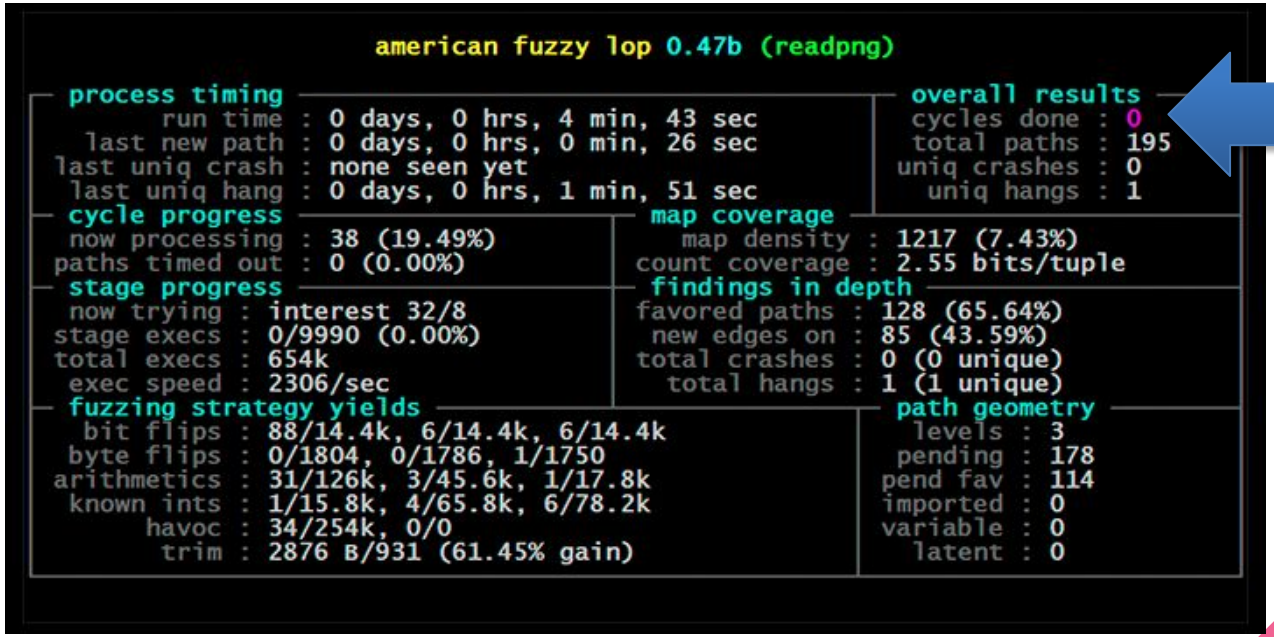
AFL : fuzzing process

- Launch a **first run**,



AFL : fuzzing process

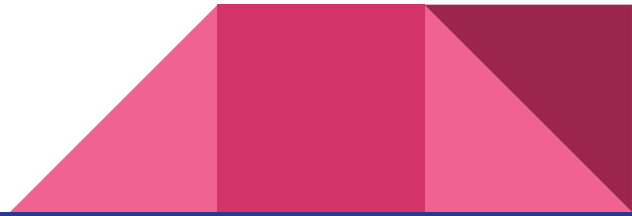
- Launch a **first run**,



cycle counter

Interesting env variables

- **AFL_DONT_OPTIMIZE**
- **AFL_SKIP_CPUFREQ**
- **AFL_EXIT_WHEN_DONE**
- **AFL_HARDEN**



“*Sanitize, Fuzz, and Harden Your C++ Code*” conference by **Kostya Serebryany**

Kostya Serebryany

- Software engineer and software systems at google research department

- Kostya’s team develops dynamic testing tools, of “The LLVM Compiler Infrastructure” : the **sanitizers**
  And also, another fuzzing tool little bit different from AFL, which is **LibFuzzer**

cf. the conference : https://www.youtube.com/watch?v=FP8zFhB_cOo

“*Sanitize, Fuzz, and Harden Your C++ Code*” conference by **Kostya Serebryany**

About fuzzing

- At google, **several thousand of cpu cores** are **fuzzing** something every minutes.

- While the **development of chromium**, they found more than **5000 bugs with fuzzing**

cf. the conference : https://www.youtube.com/watch?v=FP8zFhB_cOo

Sanitizers : description

Sanitizers

- ASan: Address Sanitizer detects use-after-free, buffer-overflow, and leaks.
- TSAN: Thread Sanitizer detects data races, deadlocks.
- MSAN: Memory Sanitizer detects uses of uninitialized memory.
- UBSan: Undefined Behavior Sanitizer detects… that.

New tools, based on **compiler** instrumentation.
Available in LLVM and GCC (both open-source)

Image from the conference of the previous slide :
https://www.youtube.com/watch?v=FP8zFhB_cOo

My experience

What did i found with fuzz testing :

- forgotten checks
- floating point exceptions
- conception problems
- I found some cases that could never happen, but they did !

Lot of other things

Just go to **http://lcamtuf.coredump.cx/afl/** get the last release and read all the documentation available.

Participate to the AFL project

Be part of the AFL project by asking questions and proposing ideas for this project, on the google group next here :

https://groups.google.com/forum/#!forum/afl-users

Thank you for your attention !

I hope you have learnt some interesting things.

This is a completely subjective “documentation”.
It is absolutely not the official one or the unique and right way to use AFL.

HOUDU Loïc
loic.houdu@gmail.com