American Fuzzy Lop

Introduction & technical demo

What is “fuzzy testing” ?

What is “fuzzy testing” ?

- **A way to test sofware products**
  - test sureness : 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 “fuzzy testing” ?

- **The manner :**
  - providing unexpected, invalid or random data
    - usual inputs : keyboard, mouse, API calls
    - unusual input but think to it : databases, SHM and all other ways that 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 parallelizism

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 do

- set CPU **scale governor** to **“performance”** (won’t works if not)

Must provide

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

Keep it simple

- More ***various*** and ***shortest files***, are better to get good perfs (in term of test efficiency)

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_dictionnary : put a dictionnary to help fuzzer

-Z 1 : force fuzzer to use specified processor core

(little performance gain)

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

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

Sample of JINK dictionnary

```
full_basic_jink.dict
"tags"
"name"
"segment"
"data"
"label"
"format"
"formats"
"channels"
"metadata"
"type"
"sample-rate"
"uniform-sampling"
"unit"
"resolution"
"quantization"
"min"
```

Execution

Command line : level 2

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**.

And then, pull the trigger...

```
@sdkt048-jessie-fuzzy 11:18:59 test2 $ ./test2.sh hey
afl-fuzz 2.10b by <lcamtuf@google.com>
[+] You have 12 CPU cores and 4 runnable tasks (utilization: 33%).
[+] Try parallel jobs - see docs/parallel_fuzzing.txt.
[+] Using specified CPU affinity: main = 1, child = 1
[*] Checking core_pattern...
[*] Checking CPU scaling governor...
[*] Setting up output directories...
[+] Output directory exists but deemed OK to reuse.
[*] Deleting old session data...
[+] Output dir cleanup successful.
[*] Scanning 'in'...
[+] No auto-generated dictionary tokens to reuse.
[*] Creating hard links for all input files...
[*] Validating target binary...
[*] Attempting dry run with 'id:000000,orig:myInk.jink'...
[*] Spinning up the fork server...
[+] All right - fork server is up.
    len = 518, map size = 6095, exec speed = 3313 us
[+] All test cases processed.

[+] Here are some useful stats:

    Test case count : 1 favored, 0 variable, 1 total
       Bitmap range : 6095 to 6095 bits (average: 6095.00 bits)
        Exec timing : 3313 to 3313 us (average: 3313 us)

[*] No -t option specified, so I'll use exec timeout of 20 ms.
[+] All set and ready to roll!
```

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

-Z 1

no **-x** given

test binary with input cases

what is written

Everything is ok !

And then, pull the trigger...

```
                american fuzzy lop 2.10b (more_heterogeneous_jink)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 17 hrs, 45 min, 22 sec     │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 27 min, 23 sec      │  total paths : 2072   │
│ last uniq crash : 0 days, 0 hrs, 6 min, 54 sec       │ uniq crashes : 90     │
│  last uniq hang : 0 days, 5 hrs, 28 min, 54 sec      │   uniq hangs : 40     │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 2042 (98.55%)    │    map density : 9497 (14.49%)         │
│ paths timed out : 0 (0.00%)        │ count coverage : 2.62 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : interest 32/8        │ favored paths : 328 (15.83%)           │
│ stage execs : 133k/198k (67.07%)   │  new edges on : 548 (26.45%)           │
│ total execs : 134M                 │ total crashes : 297k (90 unique)       │
│  exec speed : 332.5/sec            │   total hangs : 1558 (40 unique)       │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 600/3.27M, 75/3.27M, 16/3.27M         │    levels : 6          │
│  byte flips : 0/408k, 2/408k, 5/407k                │   pending : 1683       │
│ arithmetics : 117/22.7M, 2/2.31M, 0/4549            │  pend fav : 16         │
│  known ints : 21/1.93M, 18/11.2M, 12/17.7M          │ own finds : 1087       │
│  dictionary : 95/22.1M, 46/22.3M, 48/20.1M          │  imported : 984        │
│       havoc : 120/2.87M, 0/0                        │  variable : n/a        │
│        trim : 34.22%/167k, 0.09%                    ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu@00: 19%]
```

total path : unique call stacks found

total execs : number of tests done

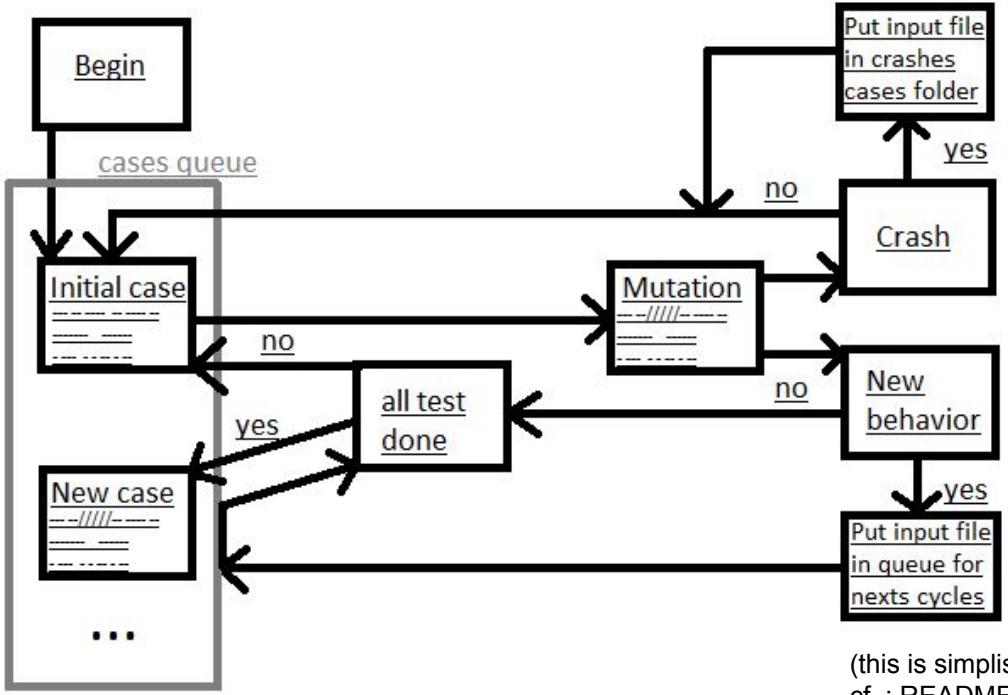
“90 unique” : means 90 differents call stacks that crash

levels : deepness of the mutation

exec speed : number of tests / sec

cf. : status_screen.txt

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



(this is simplistically draw)
cf. : README

Mutations ? What’s mutation ?

- **bitflip L/S** : L bits are flipped every S bits, variants are 1/1, 2/1, 4/1, 8/8, 16/8, 32/8
- **arith L/8** : try to substract or add smalls integer to 8, 16 or 32 bits
- **interest L/8** : same as arith but with “interesting” values
- **extras** : in case of given dictionnary it will overwrite or insert values
- **havoc** : fixed length cycle that randomly combine every precedent technics, and add : bloc deletion and/or bloc duplication
- **splice** : last resort strategy, in case that no new path were found for an entire cycle : do the same as havoc, but first splices together two random inputs from the queue at some arbitrarily selected midpoint

cf. : status_screen.txt

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…

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

Lot of other things

There is a lot af other tools & functionnality provided by AFL, i can't list them all.

There is a lot a way to use them for many reasons.

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

Also, 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 reading

I hope you learnt some interesting things.

This is a completely subjective documentation.
It’s absolutly not the official one or the unique. And it’s no more the only way to use AFL.

HOUDU Loïc
loic.houdu@gmail.com