Security in Software Applications
Project 3, 2020-21

Davide Quaranta
1715742

Contents

Abstract

This report documents the process of fuzzing ImageMagick with AFL[1]. Fuzzing is an automated testing technique that consists of sending random or unexpected data to a software and monitoring it for crashes or generally unwanted behavior.

For this purpose, it was chosen an earlier version of ImageMagick (6.7.7-10, released in 2012)[2] and was first used the PNG subset of the synthetic image test set provided on AFL's official site[3], then a single PNG image[4], for fuzzing the ImageMagick tool `convert`.

Since fuzzing is a resource-intensive process that can take several days before it starts producing acceptable results, the process was run on a 3-core VPS running Ubuntu, provided by Hetzner Cloud, with parallel `afl-fuzz` instances.

The process was separated into 4 runs, of which 2 to test the environment and the configuration; the 3rd run was correctly setup and found 1452 reproducible crashes, all referring to 8 different parts of the code. The 4th run was executed with a single image, and found 1008 reproducible crashes, referring to 6 more parts of the code.

Overall, the experiment took approximately 7 days, of which:

- Run 1: 4 hours.
- Run 2: 5 hours.
- **Run 3: 4 days**.
- **Run 4: 3 days**.

The report documents the process, the crash analysis strategy, includes graphs of execution provided by `afl-plot`, a categorization of the found crashes, and considerations on some flaws.

[1]`https://lcamtuf.coredump.cx/afl/`
[2]`https://sourceforge.net/projects/imagemagick/files/old-sources/6.x/6.7/`
[3]`https://lcamtuf.coredump.cx/afl/demo/`
[4]`https://github.com/google/AFL/blob/master/testcases/images/png/not_kitty.png`

1. Tools and setup

To conduct the experiment, it was needed to:

1. Compile and install AFL from source[1];
2. Compile and install ImageMagick 6.7.7-10 from source;
3. Download testcases that will be used to generate mutations.

1.1 AFL

American Fuzzy Lop is a fuzzer to brute-force programs to find the inputs that cause fatal signals (`SIGSEGV`, `SIGILL`, `SIGABRT`) or hangs, by applying subtle evolutionary mutations to a user-specified initial set of test inputs, and monitoring the behavior of the target program.

If the source code is available (as in this experiment), the target program can be compiled with the included wrappers `afl-gcc`, `afl-g++`, `afl-clang-fast`, `afl-clang-fast++`, that - at the cost of some overhead - add the *instrumentation* needed to monitor the program's execution paths and guide the algorithm that generates mutations.

The progress of the fuzzing process can be monitored from a dashboard that shows real-time statistics. Other than the main binary `afl-fuzz`, AFL comes with utilities like `afl-whatsup` to check the status of the fuzzer(s) from the output directory; `afl-plot` to plot execution statistics, `afl-tmin` to minimize testcases, and other tools.

1.2 Setup

Installing AFL straightforward, but in some systems it may require some tweaks, such as changing the CPU governor to *performance*; doing these optimizations is not difficult since they are well-documented in the included `performance_tips.txt`.

With regards to the compilation of ImageMagick, it can be done using the `afl-clang` wrapper provided by AFL, which requires some further steps documented inside the `llvm-mode` directory. As per documentation, `afl-clang` should improve the execution speed of the fuzzing process, but in this experiment didn't perform really better than `afl-gcc`.

Before compiling ImageMagick, it is important to install the development libraries needed for the image format of the input files, like PNG for this experiment; then ImageMagick can be installed and compiled with:

```
cd ImageMagick-6.7.7-10
CC=afl-gcc CXX=afl-g++ ./configure && make && make install
ldconfig /usr/local/lib
```

ImageMagick creates temporary files in `/tmp/magick-*` that are not deleted when the software crashes, so the filesystem can rapidly fill up; for this reason is also important to periodically delete them, for example with a cron job like:

```
*/3 * * * * find /tmp -maxdepth 1 -name 'magick-*' -delete >/dev/null 2>&1
```

It is important to use a tool like `find` instead of just `rm`, to avoid getting a
`/bin/rm: cannot execute [Argument list too long]` when there are too many files.

Starting to fuzz is simple: in this experiment 3 instances of afl-fuzz were used (one per core), started with:

[1] https://github.com/mirrorer/afl

```
# main fuzzer
afl-fuzz -M f1 -m none -i input -o output convert @@ /dev/null
# secondary fuzzers
afl-fuzz -S f2 -m none -i input -o output convert @@ /dev/null
afl-fuzz -S f3 -m none -i input -o output convert @@ /dev/null
```

Where:

- `-M` sets the instance as main; `-S` as secondary;
- `-m none` removes the memory limit;
- `-i input` sets the testcases directory to `input`;
- `-o output` sets the output directory to `output`;
- `convert @@ /dev/null` is the ImageMagick tool to fuzz: `@@` will be replaced with the name of each generated mutation, and the converted file will be discarded in `/dev/null`.

In AFL terms the `output` directory is called *sync dir*, and will be periodically checked by each instance to incorporate the results of other instances in their flow. Inside the sync directory will be created the folder structure `./instance_name/{crashes,hangs,queue}`.

2. Fuzzing

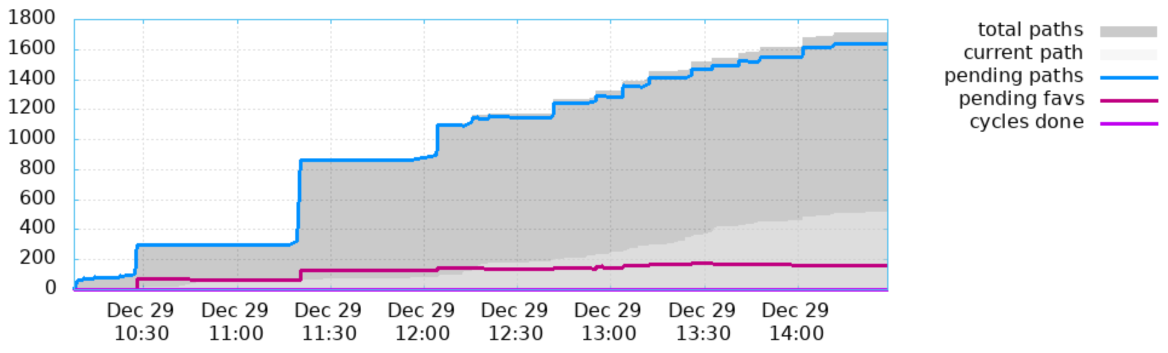
This chapter documents the fuzzing experiment, which has been split into 4 runs. The first 2 are unsuccessful and can be considered as a way to test the program and the configuration. The 3rd round had been running for 4 days and produced an acceptable set of results. A 4th round had been done with a variation, to further experiment with the tool.

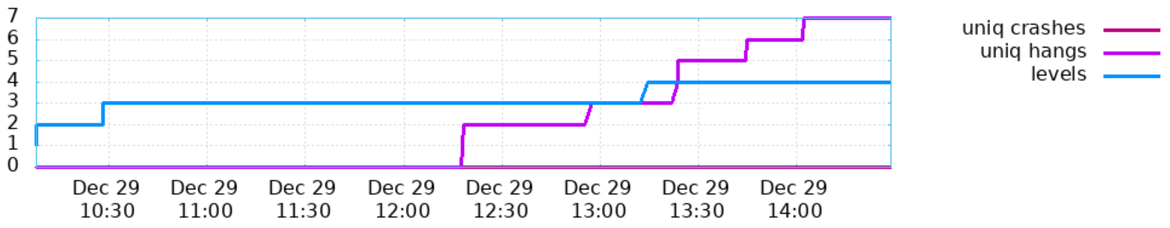
2.1 Run 1

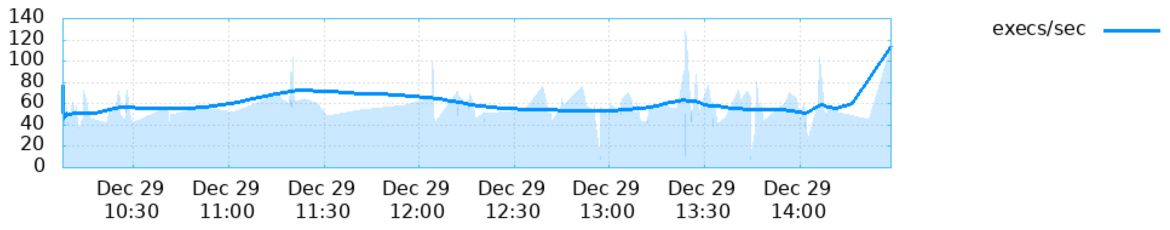
In the first execution it was used ImageMagick 7.0.7, compiled with `afl-clang-fast` and as input testcase it was used a PNG image from the Google's AFL repository[1].

This attempt had been running very slowly for 4 hours and didn't find any crash, but only some hangs.

f1 (main fuzzer)

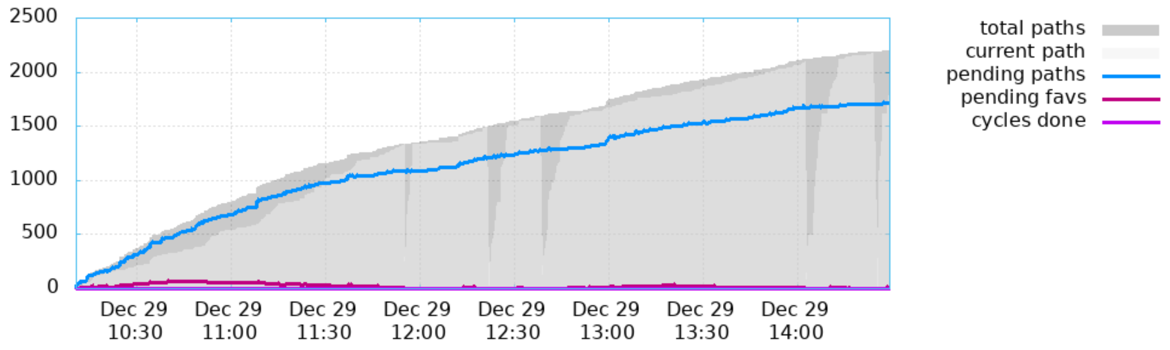


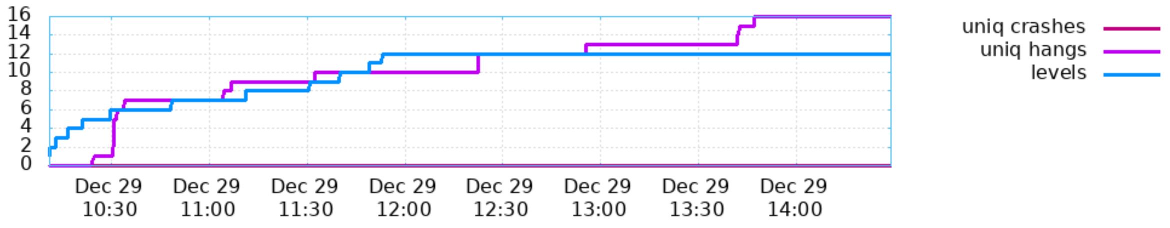


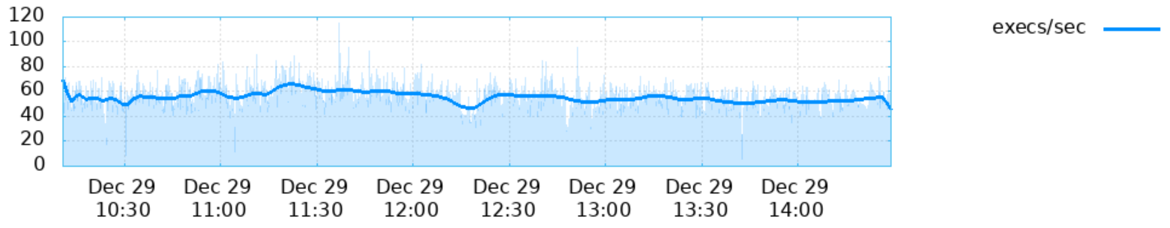


[1]https://github.com/google/AFL/blob/master/testcases/images/png/not_kitty.png

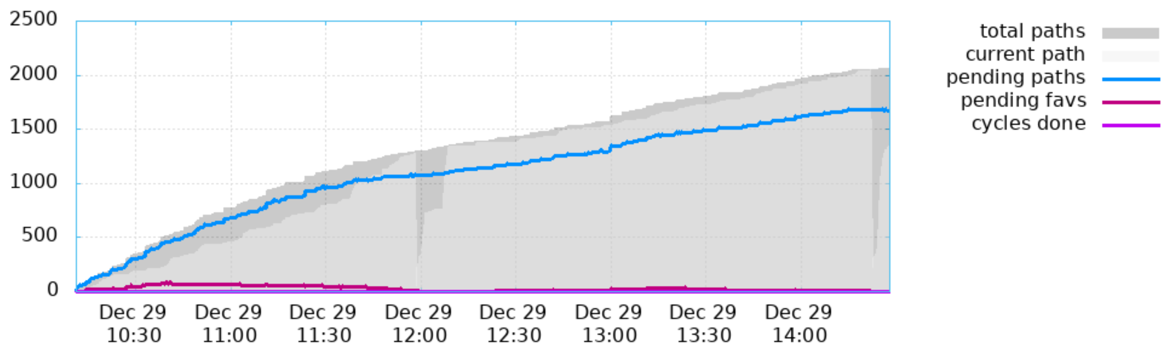
f2 (secondary fuzzer)

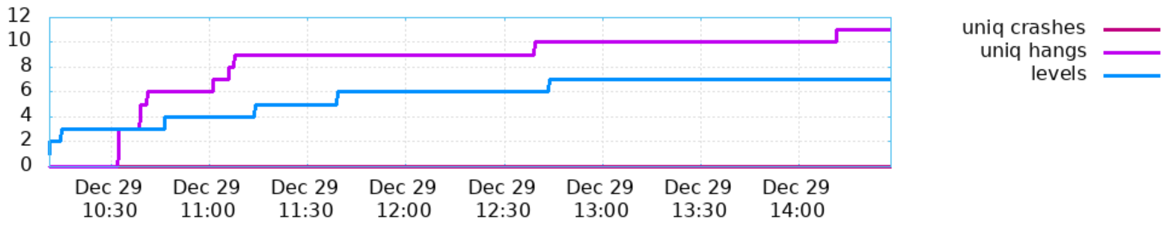


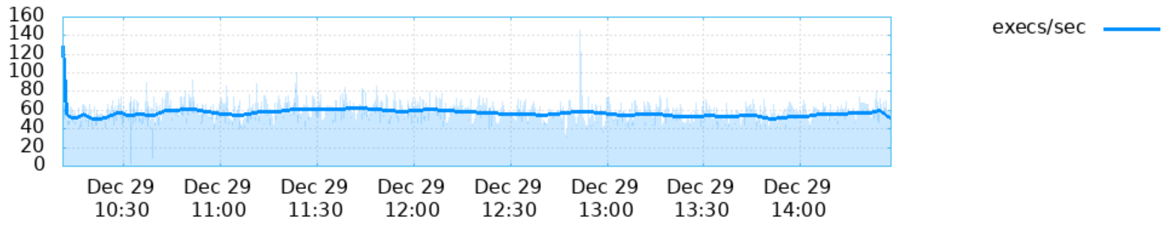




f3 (secondary fuzzer)



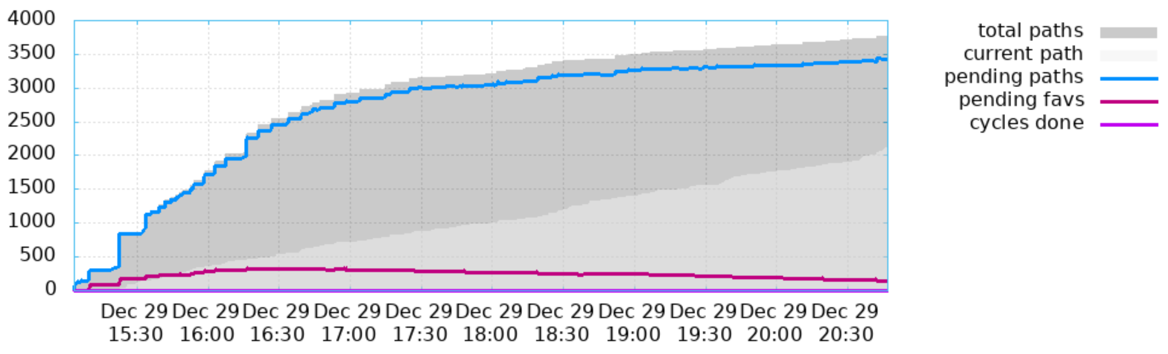


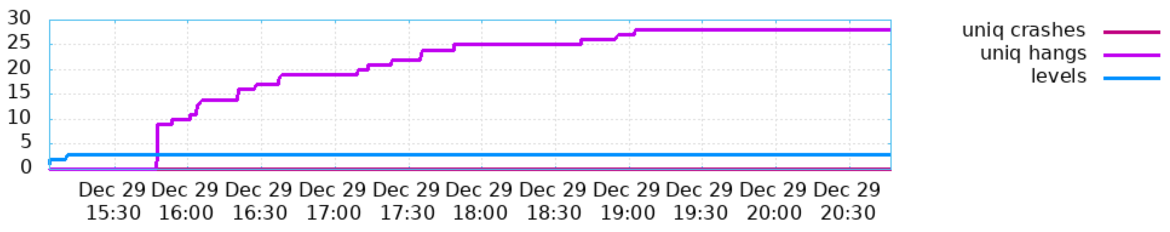


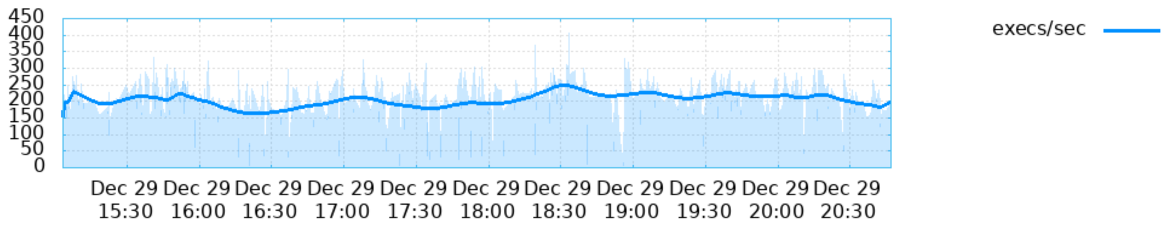
2.2 Run 2

For the second attempt, ImageMagick was compiled with `afl-gcc`, that resulted in a faster execution speed, in contrast to the expectations. The same testcase was used and the results were similar to the previous run.

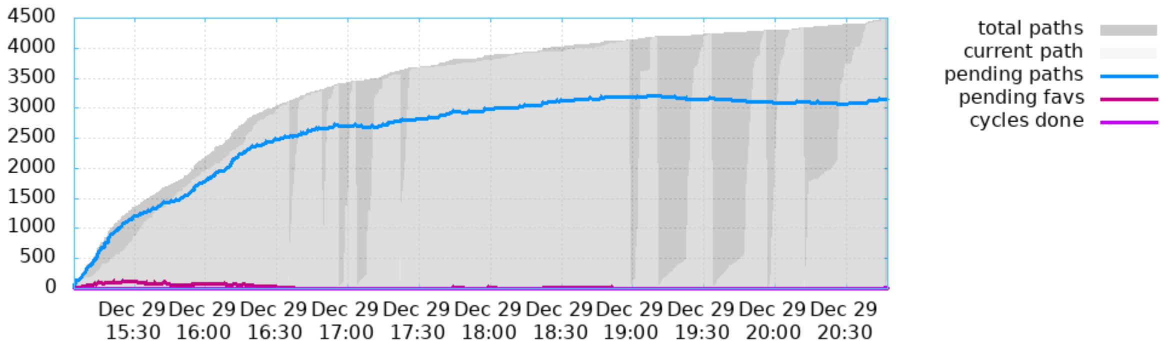
f1 (main fuzzer)

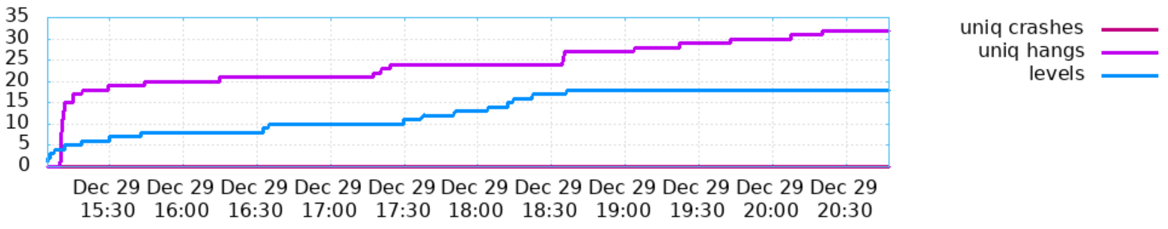


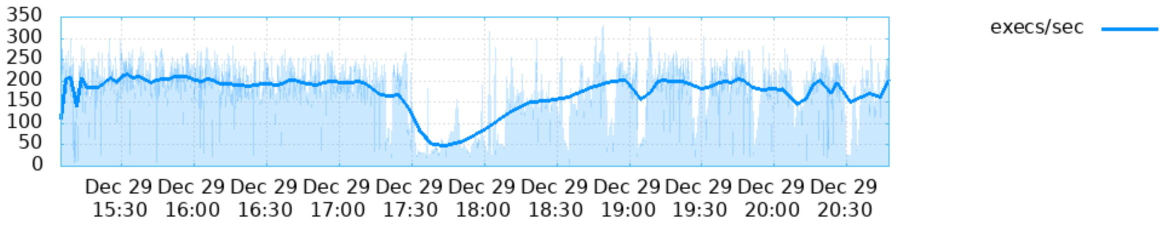




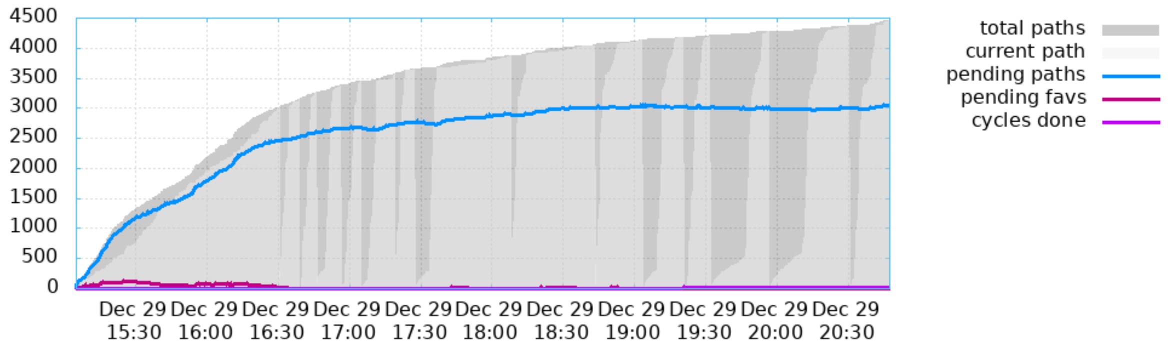
f2 (secondary fuzzer)

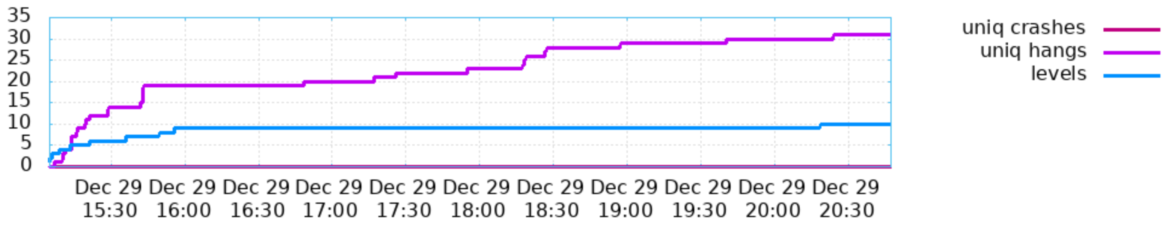


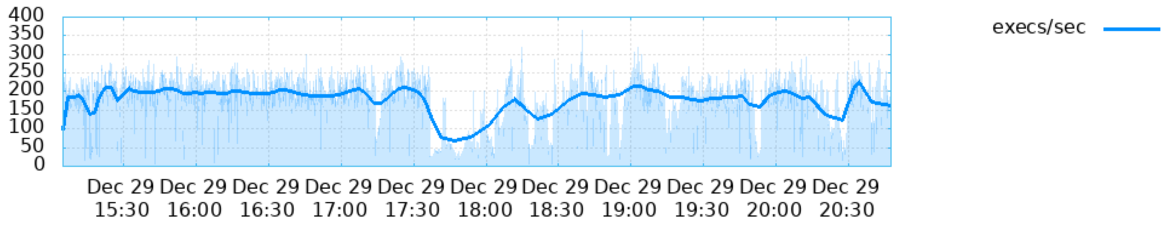




f3 (secondary fuzzer)





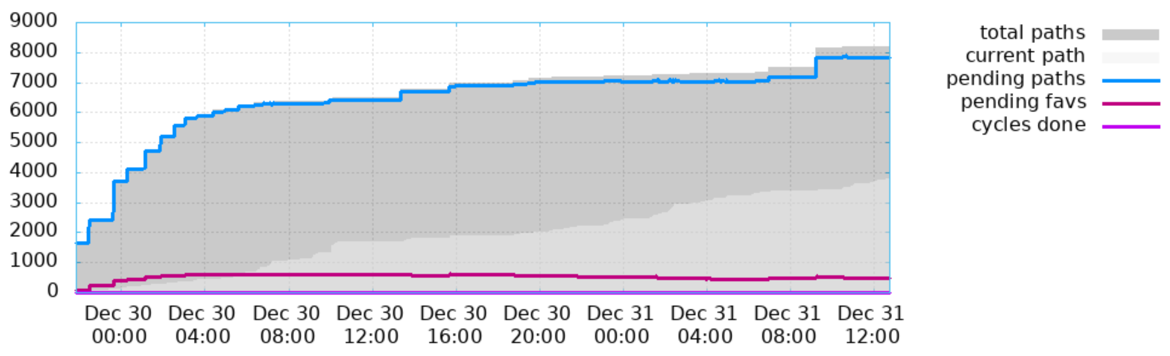


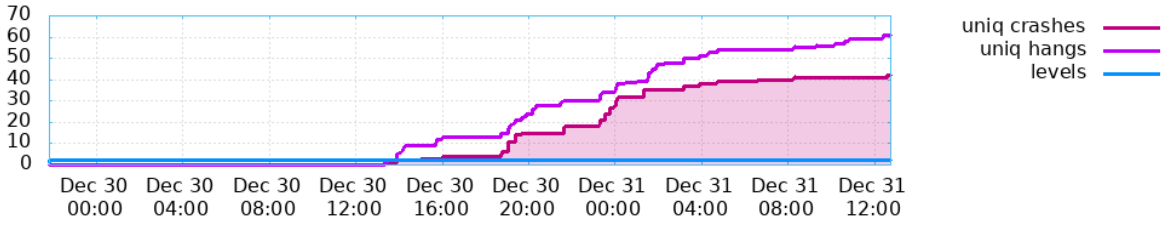
2.3 Run 3

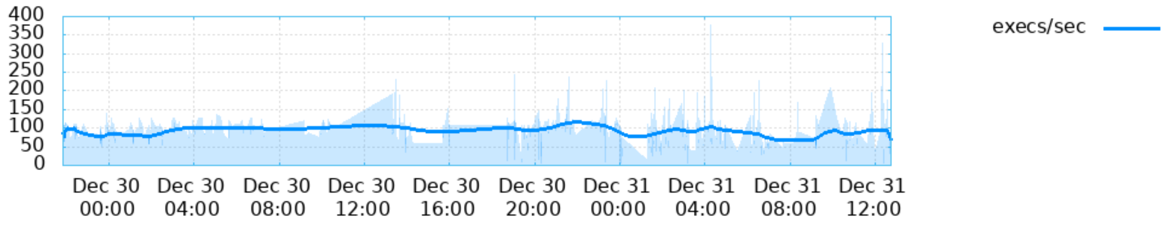
In the 3rd attempt, the following changes were done to address specific issues of the previous runs:

- It was used ImageMagick 6.7.7-10: a further earlier version, to increase the chances of finding flaws.
- It was used the PNG subset of the minimized testcase corpora provided by AFL (1652 files), which in theory acts as a stronger starting point.
- The PNG development delegates were installed before compiling ImageMagick.

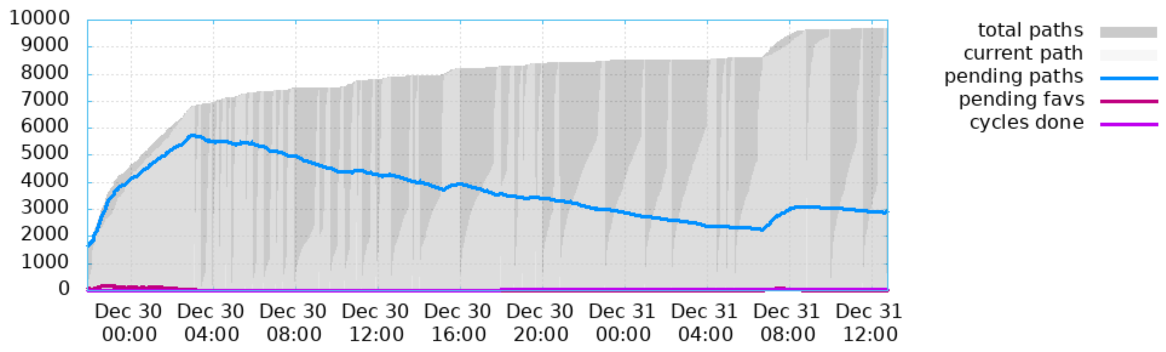
f1 (main fuzzer)

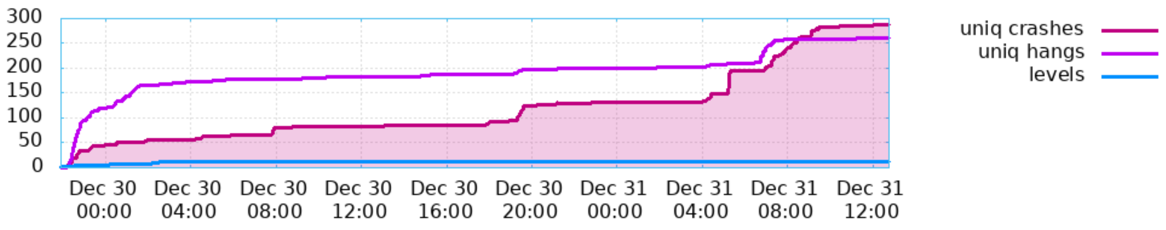


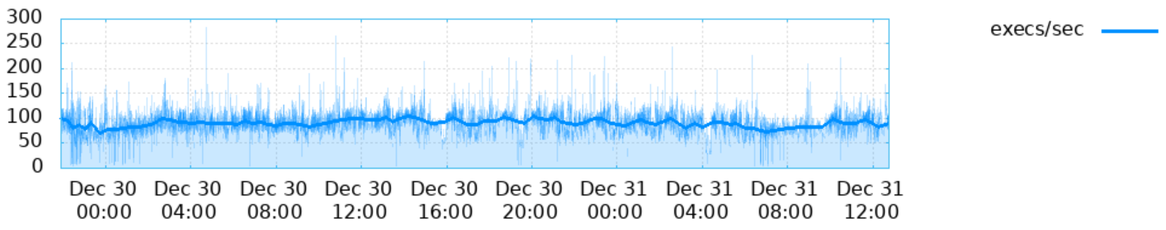




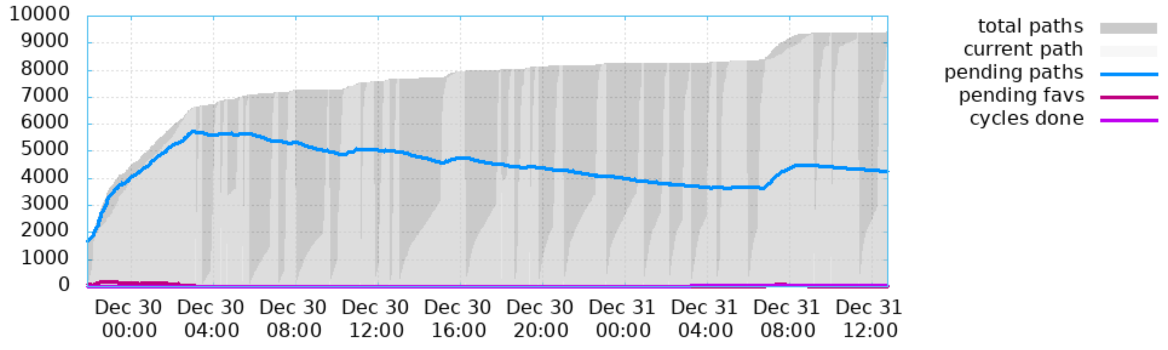
f2 (secondary fuzzer)

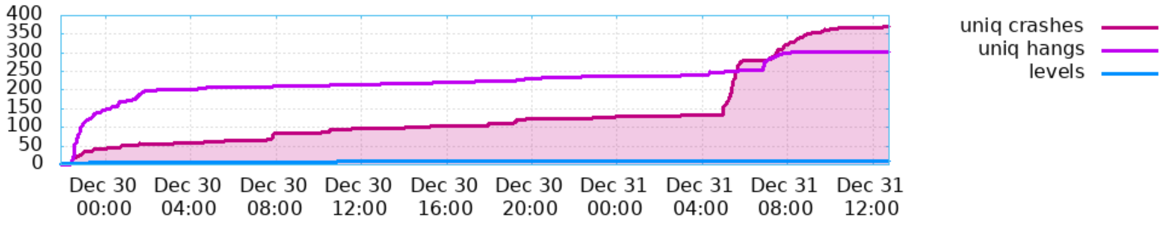


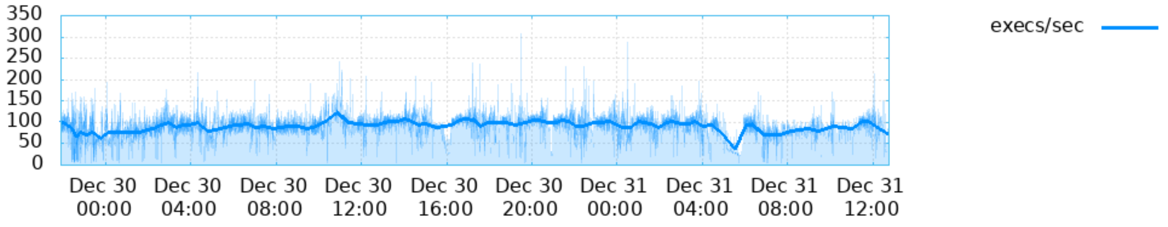




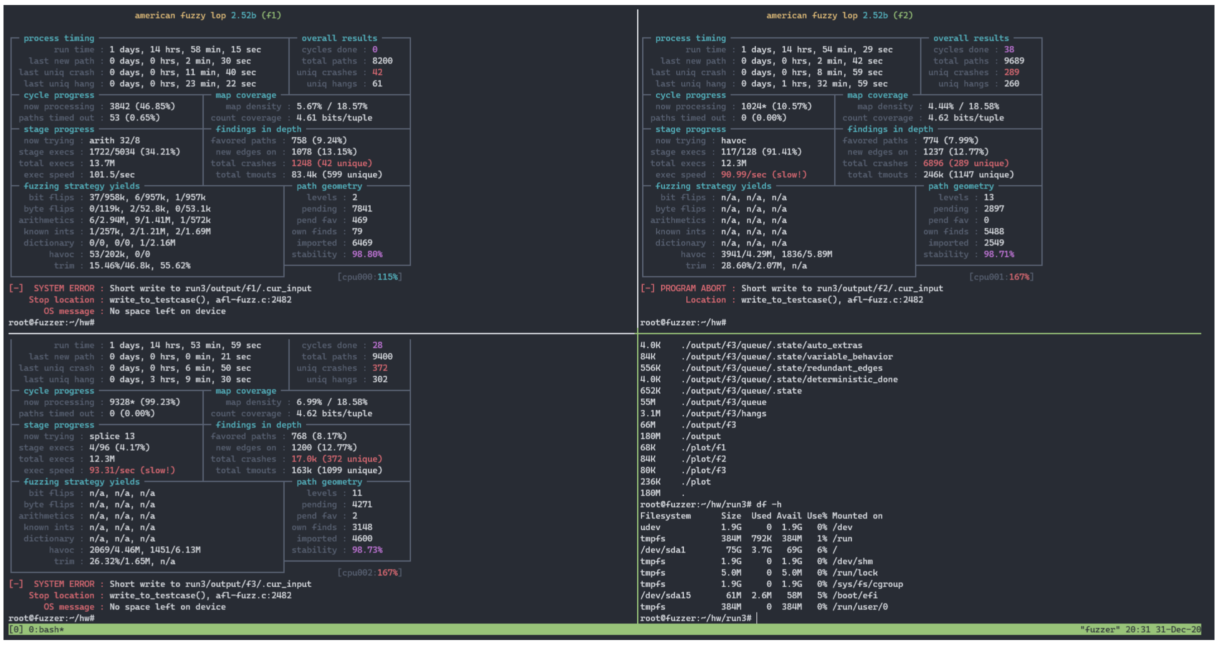
f3 (secondary fuzzer)







At this time the cron job stated in the introductory pages was set to clear `/tmp/magick-*` every hour, but it was revealed not to be sufficient, since temporary files filled up the entire drive.



```
                american fuzzy lop 2.52b (f1)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 1 days, 14 hrs, 58 min, 15 sec     │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 2 min, 30 sec       │  total paths : 8200  │
│ last uniq crash : 0 days, 0 hrs, 11 min, 40 sec      │ uniq crashes : 42    │
│  last uniq hang : 0 days, 0 hrs, 23 min, 22 sec      │   uniq hangs : 61    │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 3842 (46.85%)     │    map density : 5.67% / 18.57%       │
│ paths timed out : 53 (0.65%)        │ count coverage : 4.61 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : arith 32/8            │ favored paths : 758 (9.24%)           │
│ stage execs : 1722/5034 (34.21%)    │  new edges on : 1078 (13.15%)         │
│ total execs : 13.7M                 │ total crashes : 1248 (42 unique)      │
│  exec speed : 101.5/sec             │  total tmouts : 83.4k (599 unique)    │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ───────┤
│   bit flips : 37/958k, 6/957k, 1/957k               │    levels : 2         │
│  byte flips : 0/119k, 2/52.8k, 0/53.1k              │   pending : 7841      │
│ arithmetics : 6/2.94M, 9/1.41M, 1/572k              │  pend fav : 469       │
│  known ints : 1/257k, 2/1.21M, 2/1.69M              │ own finds : 79        │
│  dictionary : 0/0, 0/0, 1/2.16M                     │  imported : 6469      │
│       havoc : 53/202k, 0/0                          │ stability : 98.80%    │
│        trim : 15.46%/46.8k, 55.62%                  ├───────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000:115%]
[-]  SYSTEM ERROR : Short write to run3/output/f1/.cur_input
    Stop location : write_to_testcase(), afl-fuzz.c:2482
       OS message : No space left on device
root@fuzzer:~/hw#
```

```
                american fuzzy lop 2.52b (f2)

┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 1 days, 14 hrs, 54 min, 29 sec     │  cycles done : 38    │
│   last new path : 0 days, 0 hrs, 2 min, 42 sec       │  total paths : 9689  │
│ last uniq crash : 0 days, 0 hrs, 8 min, 59 sec       │ uniq crashes : 289   │
│  last uniq hang : 0 days, 1 hrs, 32 min, 59 sec      │   uniq hangs : 260   │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 1024* (10.57%)    │    map density : 4.44% / 18.58%       │
│ paths timed out : 0 (0.00%)         │ count coverage : 4.62 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc                 │ favored paths : 774 (7.99%)           │
│ stage execs : 117/128 (91.41%)      │  new edges on : 1237 (12.77%)         │
│ total execs : 12.3M                 │ total crashes : 6896 (289 unique)     │
│  exec speed : 90.99/sec (slow!)     │  total tmouts : 246k (1147 unique)    │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ───────┤
│   bit flips : n/a, n/a, n/a                         │    levels : 13        │
│  byte flips : n/a, n/a, n/a                         │   pending : 2897      │
│ arithmetics : n/a, n/a, n/a                         │  pend fav : 0         │
│  known ints : n/a, n/a, n/a                         │ own finds : 5488      │
│  dictionary : n/a, n/a, n/a                         │  imported : 2549      │
│       havoc : 3941/4.29M, 1836/5.89M                │ stability : 98.71%    │
│        trim : 28.60%/2.07M, n/a                     ├───────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu001:167%]
[-] PROGRAM ABORT : Short write to run3/output/f2/.cur_input
         Location : write_to_testcase(), afl-fuzz.c:2482

root@fuzzer:~/hw#
```

```
│        run time : 1 days, 14 hrs, 53 min, 59 sec     │  cycles done : 28    │
│   last new path : 0 days, 0 hrs, 0 min, 21 sec       │  total paths : 9400  │
│ last uniq crash : 0 days, 0 hrs, 6 min, 50 sec       │ uniq crashes : 372   │
│  last uniq hang : 0 days, 3 hrs, 9 min, 30 sec       │   uniq hangs : 302   │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 9328* (99.23%)    │    map density : 6.99% / 18.58%       │
│ paths timed out : 0 (0.00%)         │ count coverage : 4.62 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : splice 13             │ favored paths : 768 (8.17%)           │
│ stage execs : 4/96 (4.17%)          │  new edges on : 1200 (12.77%)         │
│ total execs : 12.3M                 │ total crashes : 17.0k (372 unique)    │
│  exec speed : 93.31/sec (slow!)     │  total tmouts : 163k (1099 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 : 4271      │
│ arithmetics : n/a, n/a, n/a                         │  pend fav : 2         │
│  known ints : n/a, n/a, n/a                         │ own finds : 3148      │
│  dictionary : n/a, n/a, n/a                         │  imported : 4600      │
│       havoc : 2069/4.46M, 1451/6.13M                │ stability : 98.73%    │
│        trim : 26.32%/1.65M, n/a                     ├───────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu002:167%]
[-]  SYSTEM ERROR : Short write to run3/output/f3/.cur_input
    Stop location : write_to_testcase(), afl-fuzz.c:2482
       OS message : No space left on device
root@fuzzer:~/hw#
[0] 0:bash*
```

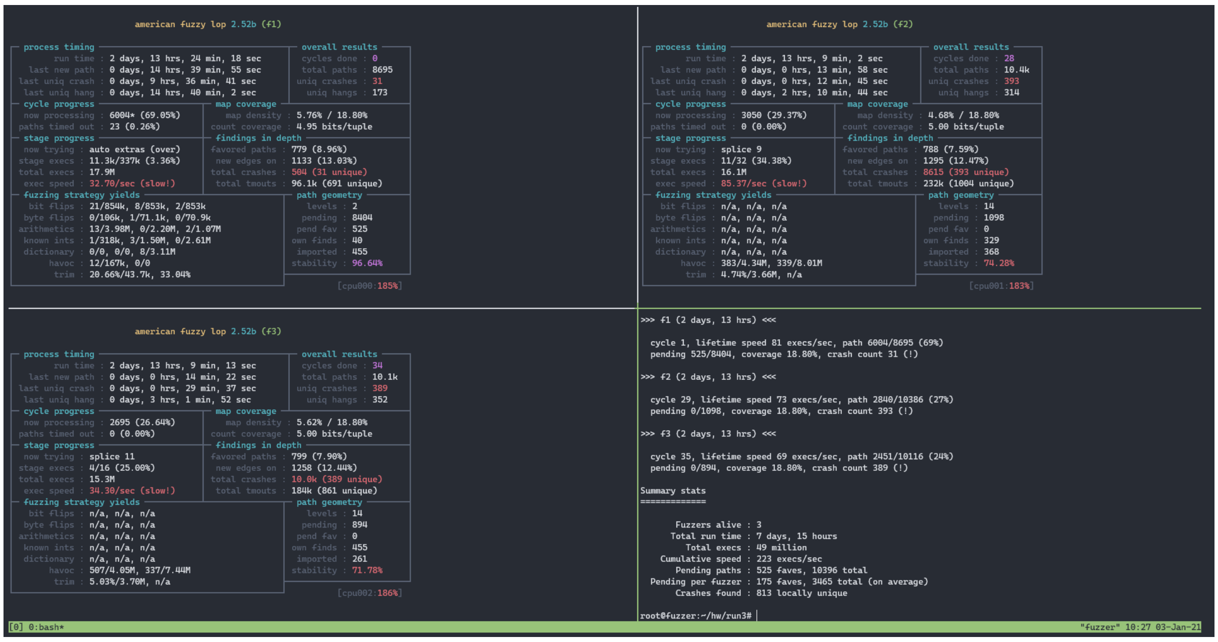
```
4.0K    ./output/f3/queue/.state/auto_extras
84K     ./output/f3/queue/.state/variable_behavior
556K    ./output/f3/queue/.state/redundant_edges
4.0K    ./output/f3/queue/.state/deterministic_done
652K    ./output/f3/queue/.state
55M     ./output/f3/queue
3.1M    ./output/f3/hangs
66M     ./output/f3
180M    ./output
68K     ./plot/f1
84K     ./plot/f2
80K     ./plot/f3
236K    ./plot
180M    .
root@fuzzer:~/hw/run3# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G     0  1.9G   0% /dev
tmpfs           384M  792K  384M   1% /run
/dev/sda1        75G  3.7G   69G   6% /
tmpfs           1.9G     0  1.9G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/sda15       61M  2.6M   58M   5% /boot/efi
tmpfs           384M     0  384M   0% /run/user/0
root@fuzzer:~/hw/run3# |
                                              "fuzzer" 20:31 31-Dec-20
```

Figure 2.1: Screenshot of the 3 instances of afl-fuzz exiting due to no space left

After reducing the deletion time to 3 minutes, the process was resumed on each instance by setting the `-i-` flag instead of the usual input. After a cumulative time of 3 days and 6 hours, this was the output of the status monitoring tool `afl-whatsup`:

```
Individual fuzzers
==================

>>> f1 (1 days, 2 hrs) <<<

  cycle 1, lifetime speed 88 execs/sec, path 5411/8351 (64%)
  pending 577/8124, coverage 18.64%, crash count 26 (!)

>>> f2 (1 days, 2 hrs) <<<

  cycle 11, lifetime speed 77 execs/sec, path 7571/9837 (76%)
  pending 0/3582, coverage 18.64%, crash count 200 (!)

>>> f3 (1 days, 2 hrs) <<<

  cycle 14, lifetime speed 74 execs/sec, path 7918/9554 (82%)
  pending 0/2981, coverage 18.64%, crash count 228 (!)

Summary stats
=============

       Fuzzers alive : 3
      Total run time : 3 days, 6 hours
         Total execs : 22 million
    Cumulative speed : 239 execs/sec
       Pending paths : 577 faves, 14687 total
  Pending per fuzzer : 192 faves, 4895 total (on average)
       Crashes found : 454 locally unique
```

Finally after 5 days the fuzzing process was manually terminated.



```
                    american fuzzy lop 2.52b (f1)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 2 days, 13 hrs, 24 min, 18 sec      │  cycles done : 0      │
│   last new path : 0 days, 14 hrs, 39 min, 55 sec      │  total paths : 8695   │
│ last uniq crash : 0 days, 9 hrs, 36 min, 41 sec       │ uniq crashes : 31     │
│  last uniq hang : 0 days, 14 hrs, 40 min, 2 sec       │   uniq hangs : 173    │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 6004* (69.05%)    │    map density : 5.76% / 18.80%        │
│ paths timed out : 23 (0.26%)        │ count coverage : 4.95 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : auto extras (over)    │ favored paths : 779 (8.96%)            │
│ stage execs : 11.3k/337k (3.36%)    │  new edges on : 1133 (13.03%)          │
│ total execs : 17.9M                 │ total crashes : 504 (31 unique)        │
│  exec speed : 32.70/sec (slow!)     │  total tmouts : 96.1k (691 unique)     │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 21/854k, 8/853k, 2/853k               │    levels : 2          │
│  byte flips : 0/106k, 1/71.1k, 0/70.9k              │   pending : 8404       │
│ arithmetics : 13/3.98M, 0/2.20M, 2/1.07M            │  pend fav : 525        │
│  known ints : 1/318k, 3/1.50M, 0/2.61M              │ own finds : 40         │
│  dictionary : 0/0, 0/0, 8/3.11M                     │  imported : 455        │
│       havoc : 12/167k, 0/0                          │ stability : 96.64%     │
│        trim : 20.66%/43.7k, 33.04%                  ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000:185%]
```

```
                    american fuzzy lop 2.52b (f2)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 2 days, 13 hrs, 9 min, 2 sec        │  cycles done : 28     │
│   last new path : 0 days, 0 hrs, 13 min, 58 sec       │  total paths : 10.4k  │
│ last uniq crash : 0 days, 0 hrs, 12 min, 45 sec       │ uniq crashes : 393    │
│  last uniq hang : 0 days, 2 hrs, 10 min, 44 sec       │   uniq hangs : 314    │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 3050 (29.37%)     │    map density : 4.68% / 18.80%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 5.00 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 9              │ favored paths : 788 (7.59%)            │
│ stage execs : 11/32 (34.38%)        │  new edges on : 1295 (12.47%)          │
│ total execs : 16.1M                 │ total crashes : 8615 (393 unique)      │
│  exec speed : 85.37/sec (slow!)     │  total tmouts : 232k (1004 unique)     │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                         │    levels : 14         │
│  byte flips : n/a, n/a, n/a                         │   pending : 1098       │
│ arithmetics : n/a, n/a, n/a                         │  pend fav : 0          │
│  known ints : n/a, n/a, n/a                         │ own finds : 329        │
│  dictionary : n/a, n/a, n/a                         │  imported : 368        │
│       havoc : 383/4.34M, 339/8.01M                  │ stability : 74.28%     │
│        trim : 4.74%/3.66M, n/a                      ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu001:183%]
```

```
                    american fuzzy lop 2.52b (f3)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 2 days, 13 hrs, 9 min, 13 sec       │  cycles done : 34     │
│   last new path : 0 days, 0 hrs, 14 min, 22 sec       │  total paths : 10.1k  │
│ last uniq crash : 0 days, 0 hrs, 29 min, 37 sec       │ uniq crashes : 389    │
│  last uniq hang : 0 days, 3 hrs, 1 min, 52 sec        │   uniq hangs : 352    │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 2695 (26.64%)     │    map density : 5.62% / 18.80%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 5.00 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 11             │ favored paths : 799 (7.90%)            │
│ stage execs : 4/16 (25.00%)         │  new edges on : 1258 (12.44%)          │
│ total execs : 15.3M                 │ total crashes : 10.0k (389 unique)     │
│  exec speed : 34.30/sec (slow!)     │  total tmouts : 184k (861 unique)      │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                         │    levels : 14         │
│  byte flips : n/a, n/a, n/a                         │   pending : 894        │
│ arithmetics : n/a, n/a, n/a                         │  pend fav : 0          │
│  known ints : n/a, n/a, n/a                         │ own finds : 455        │
│  dictionary : n/a, n/a, n/a                         │  imported : 261        │
│       havoc : 507/4.05M, 337/7.44M                  │ stability : 71.78%     │
│        trim : 5.03%/3.70M, n/a                      ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu002:186%]
```

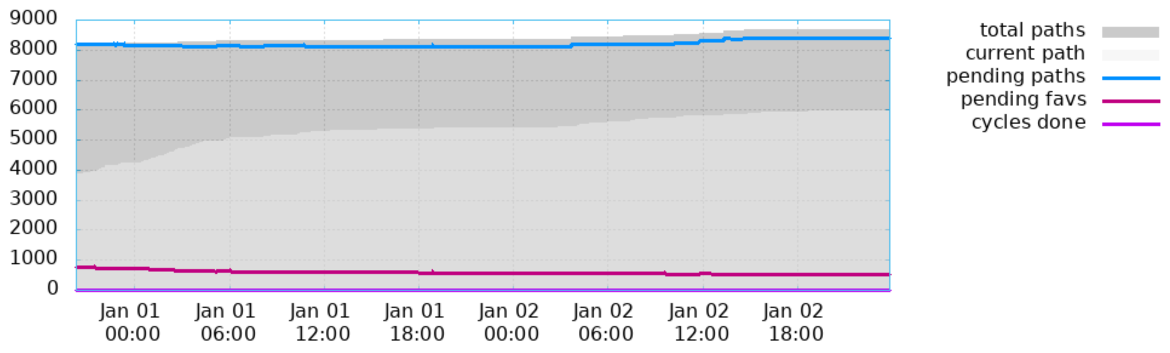
```
>>> f1 (2 days, 13 hrs) <<<

  cycle 1, lifetime speed 81 execs/sec, path 6004/8695 (69%)
  pending 525/8404, coverage 18.80%, crash count 31 (!)

>>> f2 (2 days, 13 hrs) <<<

  cycle 29, lifetime speed 73 execs/sec, path 2840/10386 (27%)
  pending 0/1098, coverage 18.80%, crash count 393 (!)

>>> f3 (2 days, 13 hrs) <<<

  cycle 35, lifetime speed 69 execs/sec, path 2451/10116 (24%)
  pending 0/894, coverage 18.80%, crash count 389 (!)

Summary stats
=============

       Fuzzers alive : 3
      Total run time : 7 days, 15 hours
         Total execs : 49 million
    Cumulative speed : 223 execs/sec
       Pending paths : 525 faves, 10396 total
  Pending per fuzzer : 175 faves, 3465 total (on average)
       Crashes found : 813 locally unique

root@fuzzer:~/hw/run3# 
```

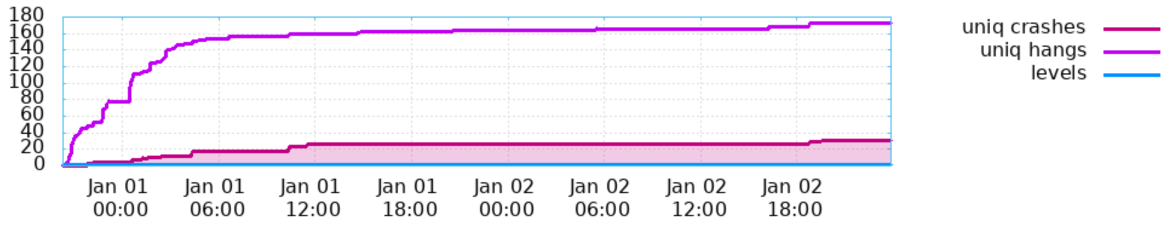
```
[0] 0:bash*                                                "fuzzer" 10:27 03-Jan-21
```

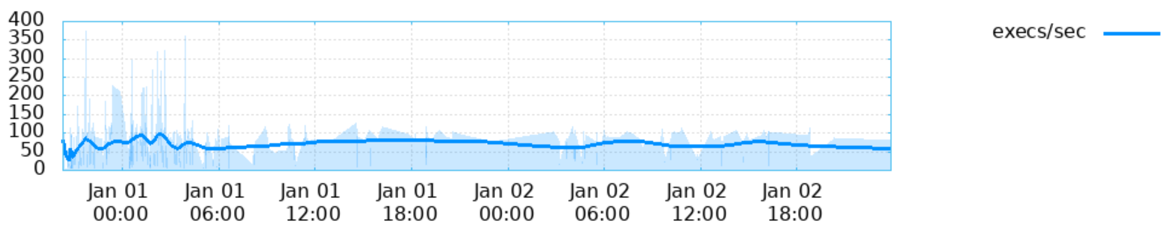
Figure 2.2: Screenshot of the status of the 3 instances of afl-fuzz after 4 days

The graphs of this resumed execution until the end are the following.

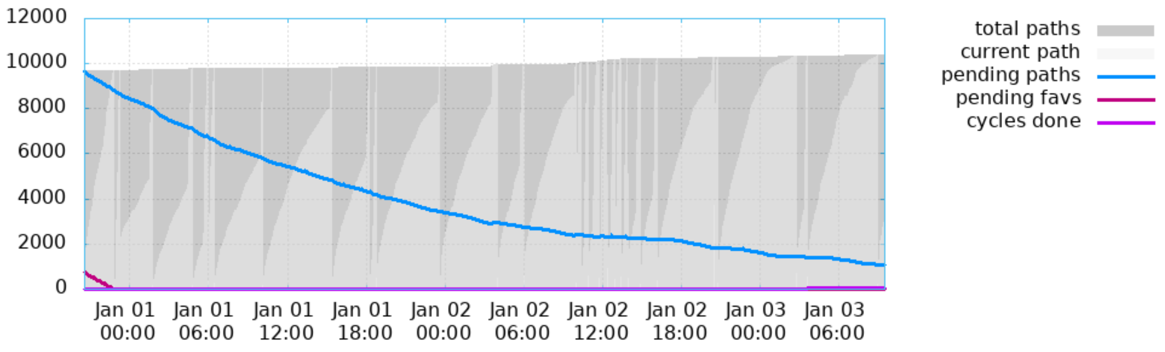
f1 (main fuzzer)

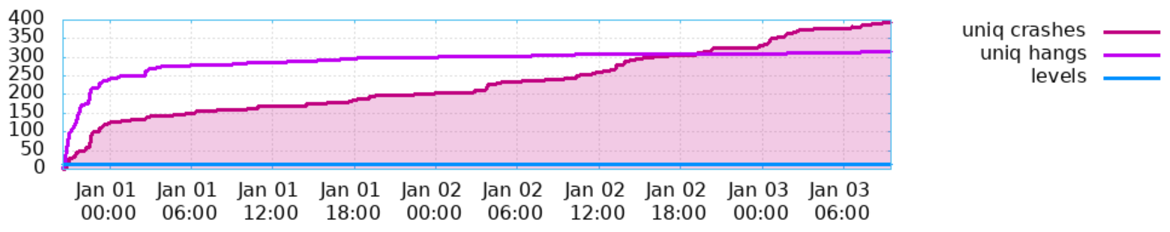


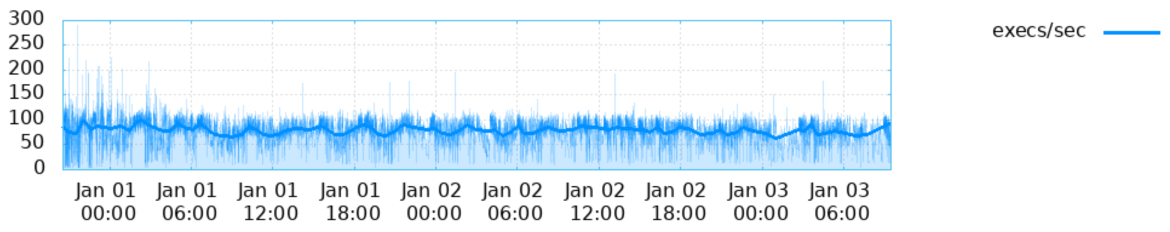




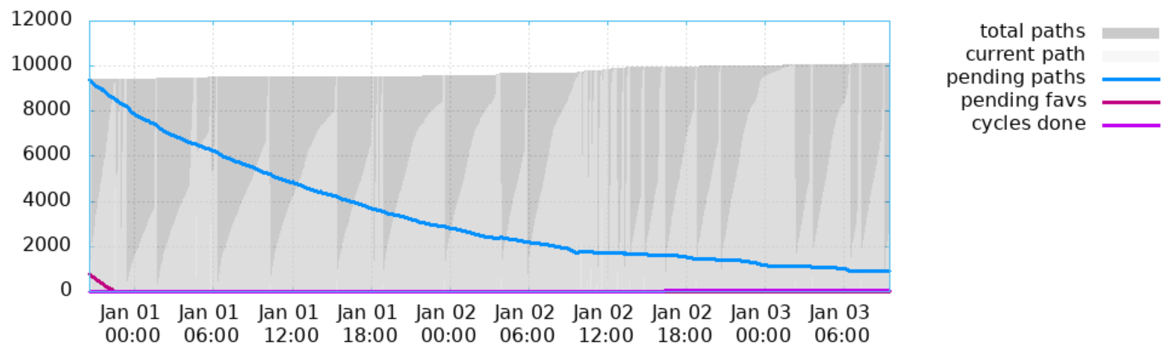
f2 (secondary fuzzer)

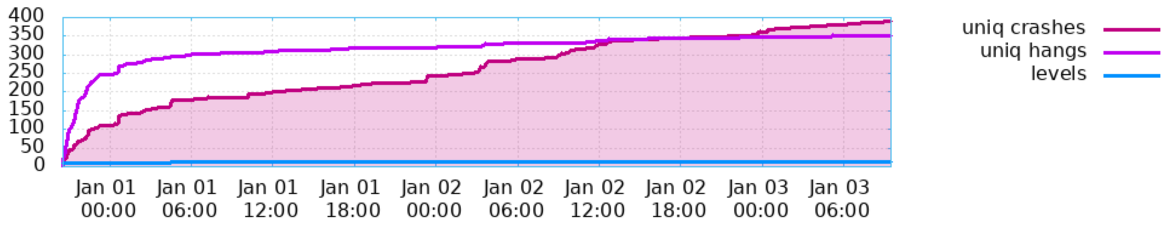


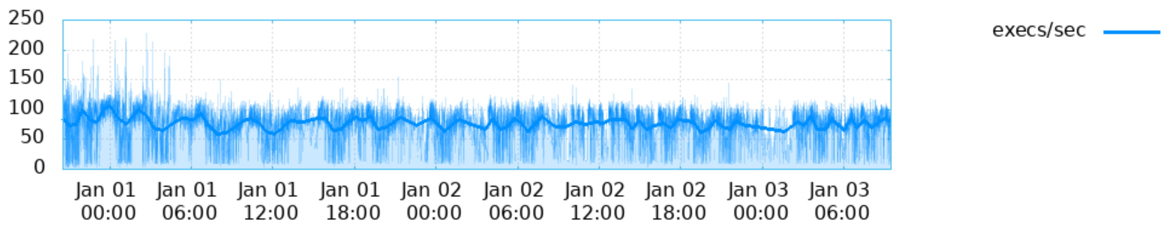




f3 (secondary fuzzer)







Crash analysis

The crash results of this run were analyzed with this strategy:

1. Get a list of the input mutations that caused crashes.
2. Get a list of `convert` output for each of them.
3. Compute statistics on the types of crashes and the code line involved.

The directories were structured like:

```
.
|-- output
|   |-- f1
|   |   |-- crashes
|   |   |-- crashes.2020-12-31-21:03:18
|   |   |-- hangs
|   |   |-- hangs.2020-12-31-21:03:18
|   |   +-- queue
|   |-- f2
|   |   |-- crashes
|   |   |-- crashes.2020-12-31-21:18:07
|   |   |-- hangs
|   |   |-- hangs.2020-12-31-21:18:07
|   |   +-- queue
|   +-- f3
|       |-- crashes
|       |-- crashes.2020-12-31-21:18:24
|       |-- hangs
|       |-- hangs.2020-12-31-21:18:24
|       +-- queue
+-- plot
    |-- f1
    |-- f2
    +-- f3
```

At this point, a list of the crashing mutations can be easily obtained with:

```
find output/*/crashes* -type f > crash_inputs.txt
```

Then it is possible to save the output of `convert` for each of these mutations with:

```
for file in $(cat crash_inputs.txt); do echo [Input]: $file; echo -ne [Output]:; convert $file /dev/null;
↪  echo ""; done > crash_errors.txt 2>&1
```

At this point the produced `crash_errors.txt` will look like:

```
[Input]: output/f1/crashes/id:000007,sig:11,src:004399,op:havoc,rep:8
[Output]:Segmentation fault

[Input]: output/f1/crashes/id:000024,sig:11,src:005295,op:ext_AO,pos:1
[Output]:Segmentation fault

[Input]: output/f1/crashes/id:000028,sig:11,src:005958,op:havoc,rep:16
[Output]:Segmentation fault

[Input]: output/f1/crashes/id:000014,sig:11,src:004941,op:arith8,pos:22,val:+19
[Output]:Segmentation fault

[...]
```

Other than this, it is needed to run a debugger on each mutation, to get more insights on the crash, as well as the line of code involved; this can be quickly achieved by running `gdb` non-interactively on each line in `crash_inputs.txt`:

```
for input in $(cat crash_inputs.txt); do echo ""; echo "[+] Debugging file: $input"; gdb -ex "r $input
↪  /dev/null" -batch convert; done > cat_gdb.txt 2>&1
```

The resulting `cat_gdb.txt` will look like:

```
[+] Debugging file: output/f1/crashes/id:000007,sig:11,src:004399,op:havoc,rep:8
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff78f6a4a in AcquireQuantumInfo (image_info=image_info@entry=0x0, image=image@entry=0x555555
133         quantum_info->endian=image->endian;

[...]

[+] Debugging file: output/f2/crashes/id:000333,sig:11,src:009092+009242,op:splice,rep:2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
ReadRLEImage (image_info=0x555555565710, exception=<optimized out>) at coders/rle.c:536
536

[...]
```

At this point each block of `cat_gdb.txt` is structured in such a way that is easily parsable to extract:

- The signal or the description of the crash.
- The input file that caused the crash.
- If the crash is caused by the fuzzed program: the function name, as well as the position in the code.

Error category	**Count**
Segmentation fault	1325
malloc()	108
free()	13
other	6
Total	1452

File:line	**Function**	**Count**
magick/quantum.c:133	AcquireQuantumInfo	1043
coders/rle.c:536	ReadRLEImage	112
coders/rle.c:532	ReadRLEImage	40
coders/rle.c:428	ReadRLEImage	21
coders/pcx.c:583	ReadPCXImage	15
coders/rle.c:427	ReadRLEImage	13
coders/pcx.c:565	ReadPCXImage	1
coders/rle.c:373	ReadRLEImage	1
		1246

Table 2.1: Run 3 errors classification, position and count

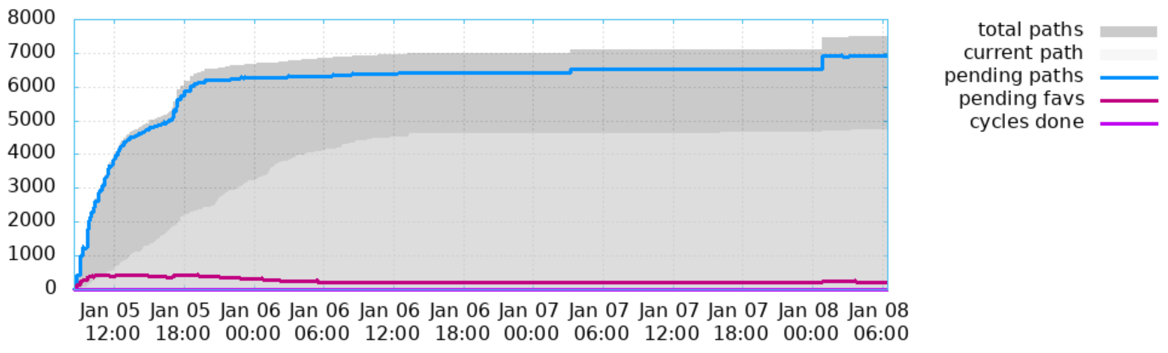
From the debugger analysis emerged that errors that are not segmentation faults are either generated from outside `convert`, or are correctly handled by it (namely not crashes); for this reason the second table only shows the position of the segmentation faults originating in `convert`.

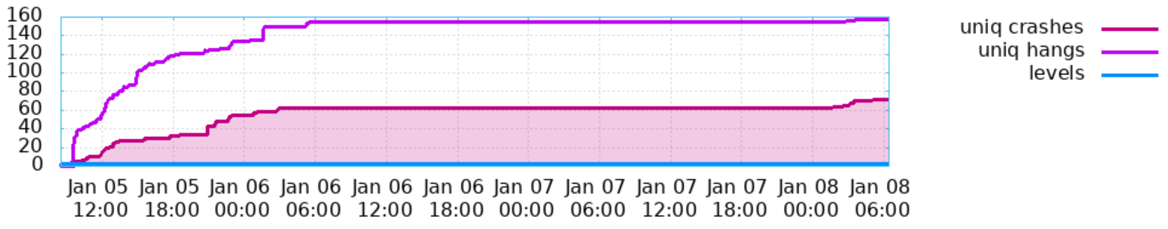
Qualitative considerations on the found flaws are in the last chapter.

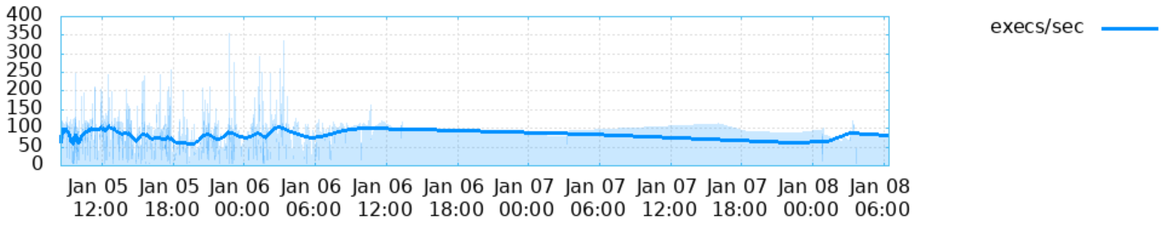
2.4 Run 4

The objective of this extra run was to test the impact of the number of input files, and was only used the previously stated PNG image from Google's AFL repository[2]. This attempt was kept running for 3 days, and produced slightly better results than the previous run.

f1 (main fuzzer)

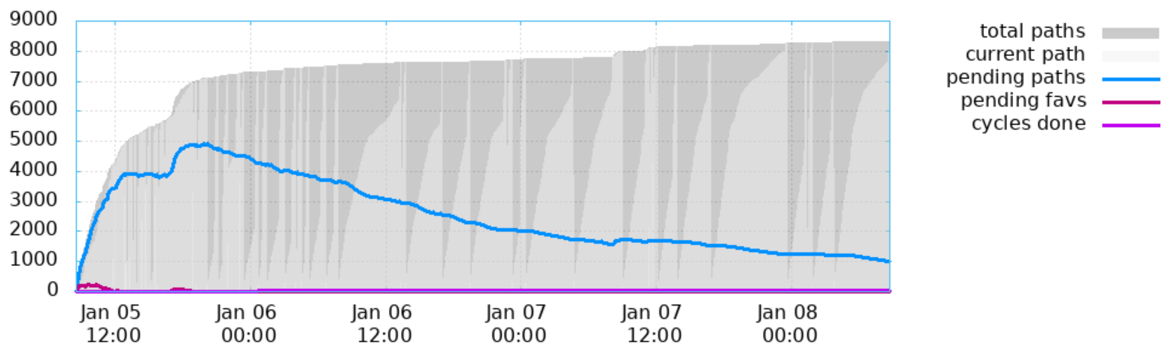


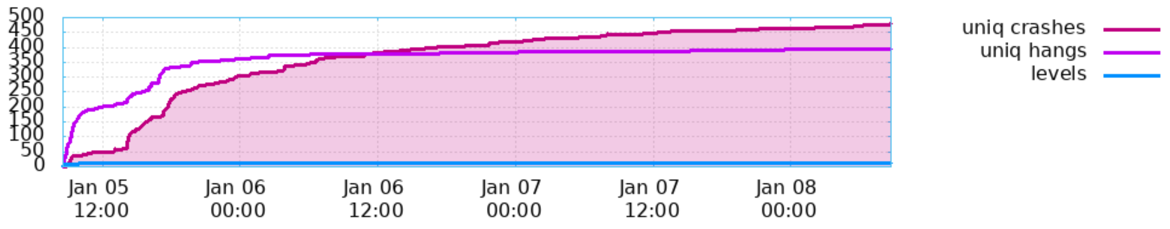


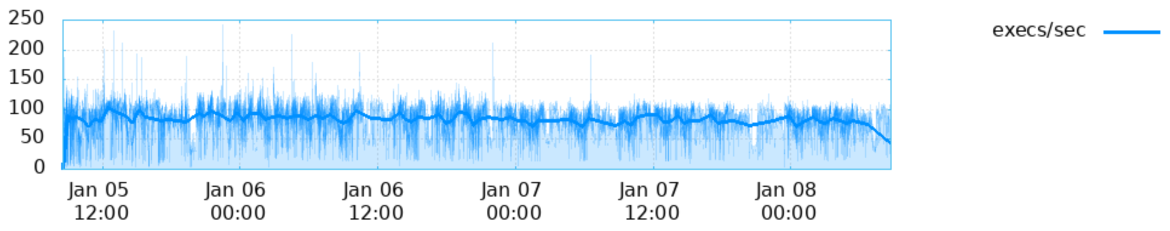


[2]`https://github.com/google/AFL/blob/master/testcases/images/png/not_kitty.png`

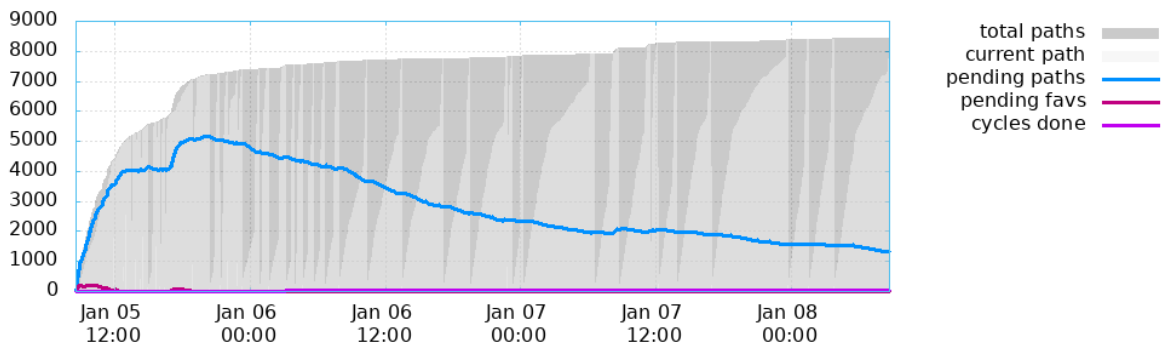
f2 (secondary fuzzer)

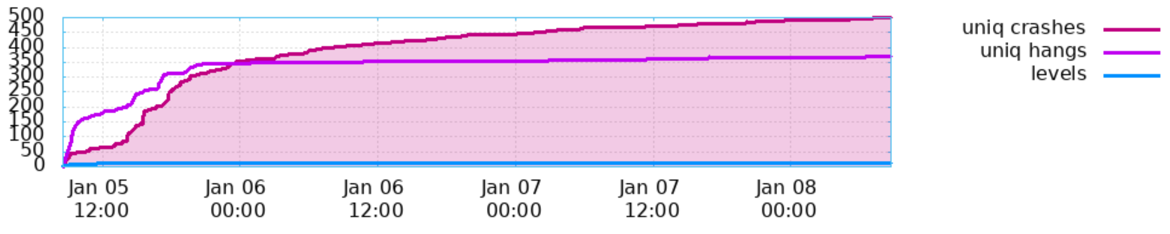


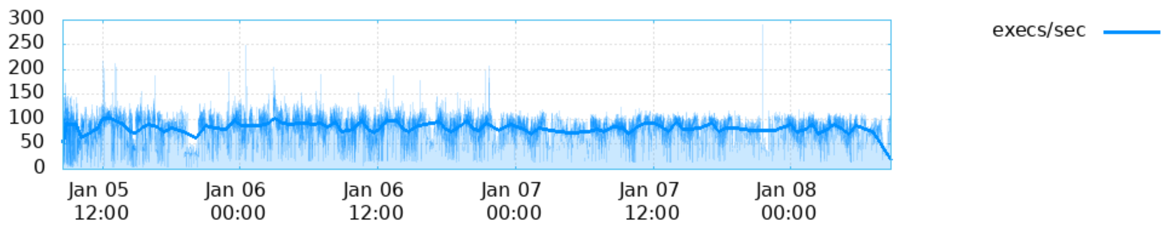




f3 (secondary fuzzer)







Crash analysis

The same analysis strategy was used for this run, which resulted in the following statistics:

Error category	**Count**
Segmentation fault	1325
malloc()	108
other	9
free()	1
Total	1008

File:line	**Function**	**Count**
magick/quantum.c:133	AcquireQuantumInfo	743
coders/rle.c:536	ReadRLEImage	64
coders/rle.c:532	ReadRLEImage	55
magick/magic.c:1049	**DestroyMagicElement**	**9**
coders/rle.c:428	ReadRLEImage	9
coders/pcx.c:565	ReadPCXImage	8
coders/rle.c:427	ReadRLEImage	5
magick/splay-tree.c:1487	**Splay**	**3**
coders/rle.c:373	ReadRLEImage	1
coders/pcx.c:583	ReadPCXImage	1
magick/splay-tree.c:841	**GetNextValueInSplayTree**	**1**
coders/pcx.c:550	**ReadPCXImage**	**1**
magick/hashmap.c:427	**DestroyLinkedList**	**1**
coders/rle.c:395	**ReadRLEImage**	**1**
	Total	902

Table 2.2: Run 4 errors classification, position and count

This run highlighted new positions (in bold) that were not found in the previous attempt, which is the opposite of the expectations, because:

- The previous run was executed with the afl-generated testcases for PNG images, that - quoting the website - “exercise a remarkable variety of features in common image parsers and *are a superior starting point* for manual testing or targeted fuzzing work”.
- This run was executed with just one simple and little PNG image for one day less, but produced not only a comparable amount of results, but also had more coverage.

3. Final considerations

In the 3rd and 4th runs a total of more than 2000 segmentation faults were found; most of them are not unique, meaning that multiple inputs lead to them. Most of these faults are not directly related to a CVE, but the functions in which the flaw originated have been involved in different CVEs.

For example the crash in `magick/quantum.c:133` is in the function `AcquireQuantumInfo`, that is related to `CVE-2016-8862` for the use of the function `AcquireMagickMemory`, since the tested version of ImageMagick is vulnerable.

```c
MagickExport QuantumInfo *AcquireQuantumInfo(const ImageInfo *image_info, Image *image)
{
  MagickBooleanType status;
  QuantumInfo *quantum_info;

  quantum_info=(QuantumInfo *) AcquireMagickMemory(sizeof(*quantum_info));
  // ... 9 lines cut ...
  quantum_info->endian=image->endian; // SIGSEGV
  return(quantum_info);
}
```

The crashes in `coders/rle.c:395` and `coders/rle.c:373` are in `ReadRLEImage`, that has several CVEs such as `CVE-2016-10050`[1], whose correction[2] involves code that is strictly related to the crashing lines.

```c
// ...
case RunDataOp:
{
  // ...
  pixel=(unsigned char) ReadBlobByte(image);
  p=rle_pixels+((image->rows-y-1)*image->columns*number_planes)+x*number_planes+plane;
  for (i=0; i < (ssize_t) operand; i++) {
    if ((y < (ssize_t) image->rows) && ((x+i) < (ssize_t) image->columns))
      *p=pixel; // SIGSEGV
    p+=number_planes;
  }
  // ...
}
```

A more explicit argument holds for `coders/rle.c:427` and `coders/rle.c:428`: in this case the code had been directly involved in a patch to fix `CVE-2014-9844`[3] in ImageMagick 6.8.9.9-4 (2014)[4,5,6]:

```
if ((number_planes >= 3) && (number_colormaps >= 3))
    for (i=0; i < (ssize_t) number_pixels; i++)
        for (x=0; x < (ssize_t) number_planes; x++)
        {
-           *p=colormap[x*map_length+(*p & mask)];
+            index=ConstrainColormapIndex(image,x*map_length+(*p & mask));
+            *p=colormap[index];
            p++;
        }
```

[1] https://www.cvedetails.com/cve/CVE-2016-10050/
[2] https://github.com/ImageMagick/ImageMagick/commit/73fb0aac5b958521e1511e179ecc0ad49f70ebaf
[3] https://www.cvedetails.com/cve/CVE-2014-9844
[4] https://bugzilla.redhat.com/show_bug.cgi?id=1343502
[5] https://salsa.debian.org/debian/imagemagick/-/commit/18bcefc754f73ad1422e8d2c8db58c51816ac151
[6] https://lists.debian.org/debian-lts/2015/05/msg00047.html

In conclusion of this experiment, correctly using the tool requires some experience to maximize the performance and to gather useful data. The main difficulties are related to the compilation of the program to test, because an improper configuration can lead to sub-optimal performance or even ineffective fuzzing, thus time wasting.

The input testcases are also important for fuzzing performance and code coverage, as seen in the comparison between the last two runs.