AFL++ setup

I didn't find difficulties in setting up the tool, I followed the good documentation in the *AFL++ Github repo* and choosed to build AFL++ from source instead of using the docker image **aflplusplus/aflplusplus** because of few GBs of free drive space. I used the **ubuntu:plucky-20241213** 80MB image in which I installed the prerequisites for AFL++ and avoided to download the stuff for QEMU, Unicorn and Frida mode, then I ran **make source-only** and **make install**. Easy.

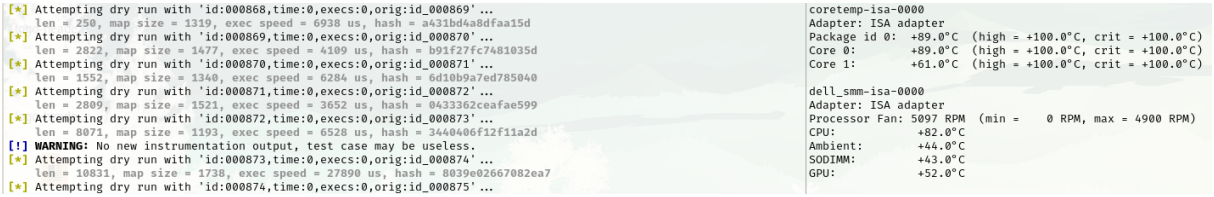
Next, I was looking for an old version of ImageMagick (IM) and in my first 2 setups (before Jan 21st) I selected version **7.0.8-12** which was the oldest release available in the *ImageMagick Github repo*. I downloaded the source code and used AFL++ C/C++ compiler to do source code instrumentation: **AFL_USE_ASAN=1 CC=afl-clang-fast CXX=afl-clang-fast++ CFLAGS="-g" ./configure && make**, then **make install**, where **AFL_USE_ASAN=1** enables the Address Sanitizer that can be used to better detect memory-related errors at runtime.

I then used the following command to start the fuzzing session: **afl-fuzz -i seeds/ -o fuzz_out/ –magick @@ /dev/null**, i.e. starting the fuzzing session on the most used feature of ImageMagick, the conversion of an image file into another image file format.

PAIN: first 2 setups (before Jan 21st)

Unfortunately, my PC has very old hardware and after approximately 20 mins of fuzzing my host decided to stop responding, it freezed and I had to reboot the machine; this happened twice. I already knew that ImageMagick creates many temporary files, so I was also running a script every 10 seconds which cleaned the **/tmp/** directory.

In my second attempt, I asked my boss the authorization to do the SSA homework on my work laptop and he exceptionally agreed. Same setup, same IM version. This time it was even worse, as soon as I started fuzzing the CPU temperature flew to 88°C and kept increasing. I stopped the fuzzing process as soon as it reached 91°C, noticing that the RPMs of the CPU fan were 2/300 above the maximum of 4900 RPMs, because I didn't want to damage the hardware. Here's a proof:



FUN: last setup (Jan 21st)

I borrowed a better laptop just for the homework. To achieve some results quickly I also decided to use and older version of ImageMagick: version **6.9.9-51** available at *ImageMagick's website*.

In this setup, the CPU temperature was pretty much constant at 72°C, so I was confident enough to leave the fuzzer on during the night.

Brief description of AFL++

AFL++ (American Fuzzy Lop++) is a state-of-the-art fuzzing tool designed for the security testing of software. It is an improved and actively maintained fork of Google's AFL, supporting more (and better) input mutations and instrumentation options, also allowing users to create custom mutations.

AFL++ tracks progress based on code coverage: you can think of the code as a tree, where branches are generated by control flow instructions and function calls. The more you explore the tree the more code you cover with your tests.

For each input in the queue, AFL++ performs deterministic and non-deterministic mutations:

- Deterministic mutations are **walking bit flips**, **simple arithmetics** like adding / subtracting integers in the input, **walking byte flips** and **known integers** like **-1**, **MAX_INT** that have an high likelihood of triggering edge conditions. After applying a mutation the program is executed looking for crashes and measuring the code coverage; if the input is interesting, it is added to the queue.
- A non-deterministic mutation is, for example, **MutateHavoc** which applies a random number of random mutations on random positions on the input file. Again, after the mutation the program is executed with the generated input and if it's interesting it is added to the queue.

Reference: *https://github.com/ercoppa/wifs2023-tutorial*

Findings on ImageMagick 6.9.9-51

As **seeds** I used 16500 small images of different formats "selected for optimal edge coverage" that I found at *https://lcamtuf.coredump.cx/afl/demo/*. Then I ran **afl-cmin -i seeds/ -o min_corpus_convert_seeds/ – convert @@ /dev/null**, where **convert** is the old name for the IM tool. **afl-cmin** is used when you have a big corpus of seeds, many inputs might be redundant or provide no additional coverage, so the tool reduces the input set to the smallest possible able to achieve the same coverage, making the fuzzer more efficient because it doesn't have to spend time on duplicate test cases.

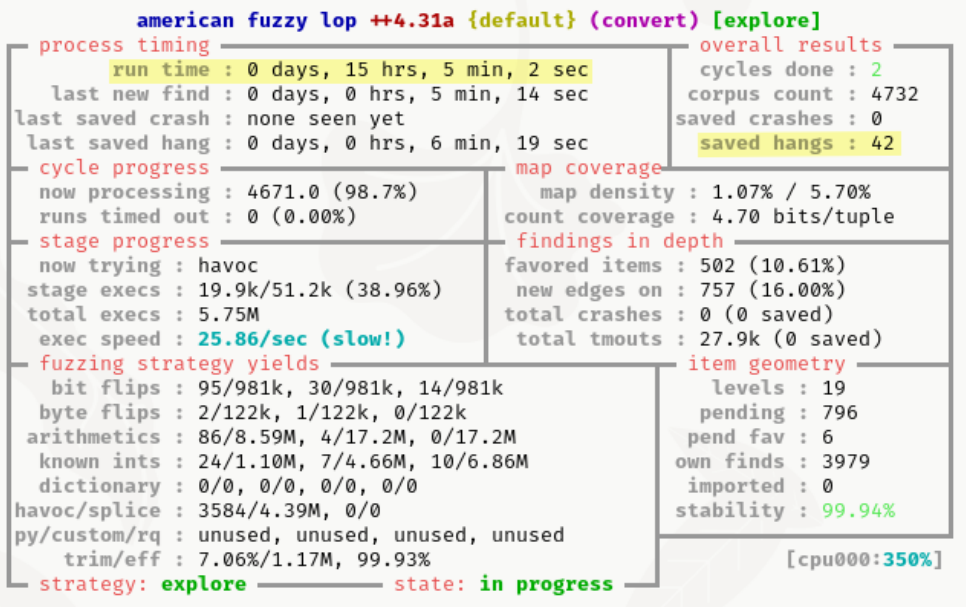
```
root@c3eca0d8ad9e:/code# afl-cmin -i seeds/ -o min_corpus_convert_seeds/ -- convert @@ /dev/null
corpus minimization tool for AFL++ (awk version)

[*] Are you aware of the '-T all' parallelize option that improves the speed for large/slow corpuses?
[+] Setting AFL_MAP_SIZE=111279
[*] Testing the target binary ...
[+] OK, 1072 tuples recorded.
[*] Obtaining traces for 16500 input files in 'seeds/'.
    Processing 16500 files (forkserver mode) ...
[*] Processing traces for input files in 'seeds/'.
    Processing file 4779/16500[!] WARNING: file id:001411,src:000636,op:flip4,pos:165,+cov.jpg is crashing the target, ignoring ...
    Processing file 4846/16500[!] WARNING: file id:000655,sync:jpeg9,src:001411,+cov.jpg is crashing the target, ignoring ...
    Processing file 5977/16500[!] WARNING: file id:010499,src:007081,op:havoc,rep:16.tif is crashing the target, ignoring ...
    Processing file 6130/16500[!] WARNING: file id:003635,src:002257,op:arith8,pos:188,val:+3.tif is crashing the target, ignoring ...
    Processing file 16500/16500
    Processing tuple 14679/14679 with count 16496 ...
[+] Found 14679 unique tuples across 16500 files.
[+] Narrowed down to 754 files, saved in 'min_corpus_convert_seeds/'.
root@c3eca0d8ad9e:/code#
```

Figure 1: I'll start the fuzzing session with 754 input files instead of 16500

You can see from **afl-cmin** that 4 input files crashed **convert**, but this is not true, the execution just took more time than the “accepted” 140ms default timeout threshold.

The final command that starts the fuzzing process is **afl-fuzz -i min_corpus_convert_seeds/ -o fuzz_out/ – convert @@ /dev/null**. After 15 hours of fuzzing, I found 42 hangs:



All the hangs have been analyzed with **strace** and **gdb backtrace**, this is a summary:

```
# Loop + pwrite64() in WritePixelCacheRegion() magick/cache.c
id:000008,src:000824,time:8944213,execs:1091831,op:havoc,rep:12
id:000009,src:000824,time:8945398,execs:1091839,op:havoc,rep:14
id:000011,src:002363,time:9998559,execs:1217353,op:havoc,rep:7
id:000017,src:001989,time:16969722,execs:1959837,op:havoc,rep:2
id:000018,src:001369,time:20718371,execs:2328836,op:havoc,rep:1
id:000024,src:002985,time:37769470,execs:3950281,op:havoc,rep:2
id:000025,src:003693,time:39200227,execs:4087523,op:havoc,rep:3
id:000028,src:003673,time:39644659,execs:4148210,op:havoc,rep:7
id:000029,src:003673,time:39661487,execs:4149477,op:havoc,rep:3
id:000031,src:003887,time:40020442,execs:4180257,op:havoc,rep:4
id:000032,src:004252,time:45504282,execs:4909460,op:havoc,rep:1
id:000035,src:004590,time:51916667,execs:5610649,op:havoc,rep:2

# Loop + CPU Usage
id:000036,src:004590,time:51938990,execs:5613549,op:havoc,rep:6
id:000037,src:004610,time:52004522,execs:5622123,op:havoc,rep:1
id:000038,src:004646,time:52611595,execs:5646275,op:havoc,rep:7
id:000039,src:004612,time:52877021,execs:5658185,op:havoc,rep:7
id:000040,src:004608,time:53332321,execs:5719243,op:havoc,rep:4
```

All the input file IDs that are missing in the image were considered hangs by AFL++ because the program didn’t return before 140ms, the default timeout threshold, but they were actually exiting with 0 or 1 with some delay.

All the real hangs with ID between **8** and **35** were ending up in the same piece of code that writes in the temporary file generated by ImageMagick, so in a file like **/tmp/magick-RANDOMSTRING**.

Follows the **strace** and **gdb** outputs:

```
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2470187568) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2470709696) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2471231824) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2471753952) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2472276080) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2472798208) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2473320336) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2473842464) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2474364592) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2474886720) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2475408848) = 522128
pwrite64(4, "\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0\377\377\377\377\377\377\0\0"..., 522128, 2475930976) = 522128
```

```
(gdb) r convert id\:000008\,src\:000824\,time\:8944213\,execs\:1091831\,op\:havoc\,rep\:12 /dev/null
Starting program: /usr/local/bin/convert convert id\:000008\,src\:000824\,time\:8944213\,execs\:1091831\,op\:havoc\,r
ep\:12 /dev/null
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
^C
Program received signal SIGINT, Interrupt.
0×00007fe1249c4187 in __libc_pwrite64 (fd=fd@entry=4, buf=0×7fe123dbc000, count=442360,
    offset=offset@entry=905068560) at ../sysdeps/unix/sysv/linux/pwrite64.c:25
warning: 25     ../sysdeps/unix/sysv/linux/pwrite64.c: No such file or directory
(gdb) bt
#0  0×00007fe1249c4187 in __libc_pwrite64
    (fd=fd@entry=4, buf=0×7fe123dbc000, count=442360, offset=offset@entry=905068560)
    at ../sysdeps/unix/sysv/linux/pwrite64.c:25
#1  0×00007fe125b6cae4 in WritePixelCacheRegion
    (cache_info=0×564baa7557f0, offset=905068560, length=905068560, buffer=0×7fe123dbc000 "\377\377\377\377\377\377")
 at magick/cache.c:3784
#2  WritePixelCachePixels
    (cache_info=cache_info@entry=0×564baa7557f0, nexus_info=<optimized out>, exception=0×564baa755698)
    at magick/cache.c:5946
#3  0×00007fe125b342cd in SyncAuthenticPixelCacheNexus
    (image=0×564baa752420, nexus_info=0×7fe123dbc000, exception=0×6bff8) at magick/cache.c:5523
#4  0×00007fe125cca4c8 in SetImageBackgroundColor (image=image@entry=0×564baa752420) at magick/image.c:2454
#5  0×00007fe125f81b68 in ReadPCXImage (image_info=0×564baa6e3580, exception=0×564baa6d2b30) at coders/pcx.c:363
#6  0×00007fe125bbd922 in ReadImage
    (image_info=image_info@entry=0×564baa6df330, exception=exception@entry=0×564baa6d2b30)
    at magick/constitute.c:492
#7  0×00007fe125bbf4a5 in ReadImages
    (image_info=image_info@entry=0×564baa6db190, exception=exception@entry=0×564baa6d2b30)
    at magick/constitute.c:870
#8  0×00007fe125900739 in ConvertImageCommand
    (image_info=0×564baa6db190, argc=4, argv=0×564baa6d6750, metadata=0×0, exception=0×564baa6d2b30)
    at wand/convert.c:633
#9  0×00007fe1259d3ba1 in MagickCommandGenesis
    (image_info=0×564baa6d6f50, command=0×7fe1258ffd20 <ConvertImageCommand>, argc=argc@entry=4, argv=argv@entry=0×7f
fcda487e88, metadata=metadata@entry=0×0, exception=exception@entry=0×564baa6d2b30) at wand/mogrify.c:172
#10 0×0000564ba9c4348c in ConvertMain (argc=4, argv=0×7ffcda487e88) at utilities/convert.c:81
#11 main (argc=4, argv=0×7ffcda487e88) at utilities/convert.c:92
```

So, without much knowledge on ImageMagick's codebase, I imagine that the **PixelCacheRegion** is the temporary file generated by ImageMagick and it seems like it is writing data in an infinite loop, as you can see from these scary screenshots and the panic CTRL+C:

```
root@c3eca0d8ad9e:/# ls -lh /tmp/magick*
-rw------- 1 root root 21G Jan 23 18:55 /tmp/magick-3554840y1P8lVrN6SXi
root@c3eca0d8ad9e:/# ^C
```

Spazio libero: 839,0 MiB

This is dangerous as an attacker could upload a 83 byte PCX image and drain the entire disk of a remote server! I could not find an existing CVE for this issue. Have a look at *CVE Mitre ImageMagick keyword*. I can see some "Memory Exhaustion" vulnerabilities when reading PCX images, but when I look at the details they refer to memory allocation in RAM.

The second class of hangs with IDs between **36** and **40** have the following **strace** output that repeats indefinitely:

```
read(3, "\1\332\1\1\0\1\0@\0g\0\1\0\0\0d\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0" ... , 4096) = 704
times({tms_utime=163 /* 1.63 s */, tms_stime=63 /* 0.63 s */, tms_cutime=0, tms_cstime=0}) = 1759012400
times({tms_utime=163 /* 1.63 s */, tms_stime=63 /* 0.63 s */, tms_cutime=0, tms_cstime=0}) = 1759012400
fstat(3, {st_mode=S_IFREG|0600, st_size=704, ...}) = 0
brk(0×55b7e42b9000)                     = 0×55b7e42b9000
read(3, "", 4096)                       = 0
lseek(3, 0, SEEK_CUR)                   = 704
lseek(3, 0, SEEK_SET)                   = 0
brk(0×55b7e42b7000)                     = 0×55b7e42b7000
brk(0×55b7e42b0000)                     = 0×55b7e42b0000
brk(0×55b7e42ae000)                     = 0×55b7e42ae000
read(3, "\1\332\1\1\0\1\0@\0g\0\1\0\0\0d\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0" ... , 4096) = 704
times({tms_utime=163 /* 1.63 s */, tms_stime=63 /* 0.63 s */, tms_cutime=0, tms_cstime=0}) = 1759012400
times({tms_utime=163 /* 1.63 s */, tms_stime=63 /* 0.63 s */, tms_cutime=0, tms_cstime=0}) = 1759012400
fstat(3, {st_mode=S_IFREG|0600, st_size=704, ...}) = 0
brk(0×55b7e42d0000)                     = 0×55b7e42d0000
read(3, "", 4096)                       = 0
lseek(3, 0, SEEK_CUR)                   = 704
lseek(3, 0, SEEK_SET)                   = 0
brk(0×55b7e42ce000)                     = 0×55b7e42ce000
brk(0×55b7e42c7000)                     = 0×55b7e42c7000
brk(0×55b7e42c5000)                     = 0×55b7e42c5000
```

It reads data from FD 3 which is the input file, retrieves process execution times, reads file metadata with **fstat()**, then allocates some memory on the heap and repeats from the **read**. It doesn't make sense to read again the same position of the input file, it isn't even supposed to change, so it will always read the same value. It seems it is in an infinite loop. Well, memory will end at some point.

```
(gdb) r convert id\:000036\,src\:004590\,time\:51938990\,execs\:5613549\,op\:havoc\,rep\:6 /dev/null
Starting program: /usr/local/bin/convert convert id\:000036\,src\:004590\,time\:51938990\,execs\:5613549\,op\:havoc\,
rep\:6 /dev/null
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
^C
Program received signal SIGINT, Interrupt.
0×00007ffb0a7dbe72 in GetImageListLength (images=0×55bbc6d6da70, images@entry=0×55bbd1fc9a70) at magick/list.c:693
693         assert(images ≠ images→previous);
(gdb) bt
#0  0×00007ffb0a7dbe72 in GetImageListLength (images=0×55bbc6d6da70, images@entry=0×55bbd1fc9a70) at magick/list.c:69
3
#1  0×00007ffb0a6637a7 in OpenPixelCache (image=image@entry=0×55bbd1fc9a70, mode=mode@entry=IOMode, exception=excepti
on@entry=0×55bbd1fccce8) at magick/cache.c:3906
#2  0×00007ffb0a62cc56 in GetImagePixelCache (image=0×55bbd1fc9a70, clone=clone@entry=MagickTrue, exception=0×55bbd1f
ccce8) at magick/cache.c:1919
#3  0×00007ffb0a664b6c in SyncImagePixelCache (image=0×1, exception=0×55bbc6d579a0) at magick/cache.c:5670
#4  0×00007ffb0aae6bbf in ReadSGIImage (image_info=0×55bba849c580, exception=0×55bba848bb30) at coders/sgi.c:377
#5  0×00007ffb0a6b7922 in ReadImage (image_info=image_info@entry=0×55bba8498330, exception=exception@entry=0×55bba848
bb30) at magick/constitute.c:492
#6  0×00007ffb0a6b94a5 in ReadImages (image_info=image_info@entry=0×55bba8494190, exception=exception@entry=0×55bba84
8bb30) at magick/constitute.c:870
#7  0×00007ffb0a3fa739 in ConvertImageCommand (image_info=0×55bba8494190, argc=4, argv=0×55bba848f750, metadata=0×0,
exception=0×55bba848bb30) at wand/convert.c:633
#8  0×00007ffb0a4cdba1 in MagickCommandGenesis
    (image_info=0×55bba848ff50, command=0×7ffb0a3f9d20 <ConvertImageCommand>, argc=argc@entry=4, argv=argv@entry=0×7f
ffbc7af378, metadata=metadata@entry=0×0, exception=exception@entry=0×55bba848bb30) at wand/mogrify.c:172
#9  0×000055bba631848c in ConvertMain (argc=4, argv=0×7fffbc7af378) at utilities/convert.c:81
#10 main (argc=4, argv=0×7fffbc7af378) at utilities/convert.c:92
```

In this case the issue occurs when reading an SGI image. How do I know that file descriptor 3 is the input file?

```
root@c3eca0d8ad9e:/# cat /proc/3559380/fdinfo/3
pos:    704
flags:  0100000
mnt_id: 1009
ino:    14283526
root@c3eca0d8ad9e:/# find / -inum 14283526 2>/dev/null
^C
root@c3eca0d8ad9e:/# find /code/fuzz_out/default/hangs/ -inum 14283526 2>/dev/null
/code/fuzz_out/default/hangs/id:000036,src:004590,time:51938990,execs:5613549,op:havoc,rep:6
```

Also, stdin (0), stdout (1) and stderr (2) are always there and ImageMagick takes a file as input.

This bug can be used to carry a DoS due to memory exhaustion and it can be faster if a server allows to upload multiple files. Also for this, I couldn't map a CVE.

Since I couldn't map CVEs for the findings, I tried to pass the input files to the latest version of ImageMagick, but without surprise the program didn't hang and exited normally.