1. Instrumenting the binary

To instrument the jpeg binary, I modified the jpeg makefile by afl-gcc to compile instead of gcc. Then, I ran CC=afl-gcc make from the command line. I initially received PIE errors. I added the -no-pie to compiler flags in make file to remediate this issue.

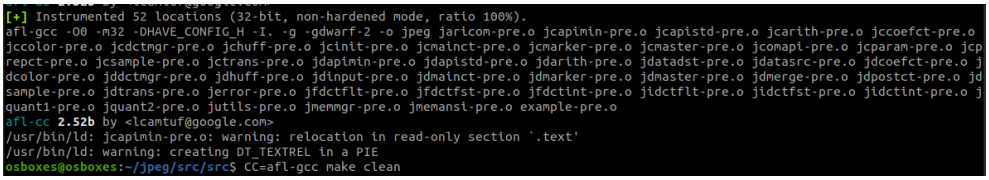

Figure 1: PIE errors initially received after running afl-gcc

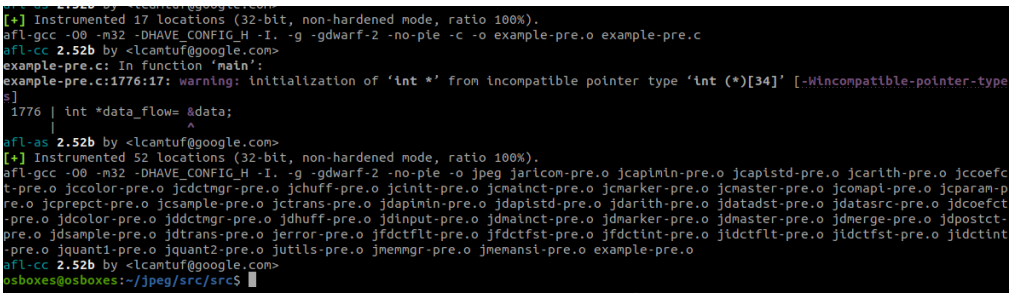

Figure 2: Results after adding the -no-pie compiler flag.

2. Choosing your samples (inputs)

For this part I ran afl-fuzz multiple times to test out different inputs. The results are in the crash section below. I initially chose to run afl-fuzz on the jpeg program with the two included.jpg input files along with the afl-fuzz input .jpg (not_kitty.jpg). The 3 files were logo.jpg not_kitty.jpg and rodeo.jpg.

Then, I ran afl-tmin on those 3 files and ran afl-fuzz with the output of those files. Next, I tried afl-cmin and ran afl-fuzz with the output of that as the input.

Lastly, I found a test corpus on the developer's website (https://lcamtuf.coredump.cx/afl/demo/) for jpegs and jpeg-turbo. There were full images and edge cases. I ran the jpeg/full images for approximately 24 hours and obtained many crashes. Documentation is below as well as attached to the assignment as instructed.

3. Creating bash script for AFL-tmin and using it successfully

To run af-tmin on the input corpus I ran the command: afl-tmin -I <input_file> -o <output_file> -f <<file> <program> <file>

The bash script is attached in assignment submission. This is a screenshot of the work part of it. The rest of the script is error detection/correction and minor cleanup.

```
61  # Loop through input file dir
62  for i in $INPUT_FILES
63  do
64      let c++
65      if test -f "$i" # Checks each thing in folder is a file
66      then
67          OUTPUT_FILE=$(basename -- "${i%.jpg}.min.jpg") #strips path and sticks min in front of jpg
68          #echo "$OUTPUT_FILE" # Test what the files look like
69          #echo "afl-tmin -i "$i" -o "$OUTPUT_DIR/$OUTPUT_FILE" -f test."$c".jpg "$PROGRAM" test."$c".jpg" # Test to see if output is desired
70          afl-tmin -i "$i" -o "$OUTPUT_DIR/$OUTPUT_FILE" -f test."$c".jpg "$PROGRAM" test."$c".jpg # Run command on each input file
71
72      fi
73  done # end loop
```

Figure 3: Image of the work part of the bash script to run afl-tmin on all items in input folder.

```
osboxes@osboxes:~/jpeg$ sudo ./afl-tminify.sh

Program found!
Inputs directory found!
Test file directory successfully created
Output directory exists ...

afl-tmin 2.52b by <lcamtuf@google.com>

[+] Read 12806 bytes from '/home/osboxes/jpeg/inputs/logo.jpg'.
[*] Performing dry run (mem limit = 50 MB, timeout = 1000 ms)...
[+] Program terminates normally, minimizing in instrumented mode.
[*] Stage #0: One-time block normalization...
[+] Block normalization complete, 6912 bytes replaced.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
    Block length = 1024, remaining size = 12806
    Block length = 512, remaining size = 12806
    Block length = 256, remaining size = 12806
    Block length = 128, remaining size = 12806
    Block length = 64, remaining size = 12806
    Block length = 32, remaining size = 12806
    Block length = 16, remaining size = 12806
    Block length = 8, remaining size = 12806
    Block length = 4, remaining size = 12798
    Block length = 2, remaining size = 12798
    Block length = 1, remaining size = 12792
[+] Block removal complete, 26 bytes deleted.
[*] Stage #2: Minimizing symbols (256 code points)...
[+] Symbol minimization finished, 0 symbols (0 bytes) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 1232 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
    Block length = 1024, remaining size = 12780
    Block length = 512, remaining size = 12780
    Block length = 256, remaining size = 12780
    Block length = 128, remaining size = 12780
    Block length = 64, remaining size = 12780
    Block length = 32, remaining size = 12780
    Block length = 16, remaining size = 12780
    Block length = 8, remaining size = 12780
```

```
[+] Block removal complete, 30 bytes deleted.
[*] Stage #2: Minimizing symbols (256 code points)...
[+] Symbol minimization finished, 0 symbols (0 bytes) replaced.
[*] Stage #3: Character minimization...
[+] Character minimization done, 1916 bytes replaced.
[*] --- Pass #2 ---
[*] Stage #1: Removing blocks of data...
    Block length = 1024, remaining size = 15802
    Block length = 512, remaining size = 15802
    Block length = 256, remaining size = 15802
    Block length = 128, remaining size = 15802
    Block length = 64, remaining size = 15802
    Block length = 32, remaining size = 15802
    Block length = 16, remaining size = 15802
    Block length = 8, remaining size = 15802
    Block length = 4, remaining size = 15802
    Block length = 2, remaining size = 15802
    Block length = 1, remaining size = 15802
[+] Block removal complete, 0 bytes deleted.

     File size reduced by : 0.19% (to 15802 bytes)
    Characters simplified : 53.44%
     Number of execs done : 44149
          Fruitless execs : path=42172 crash=0 hang=0

[*] Writing output to '/home/osboxes/jpeg/outputs/rodeo.min.jpg'...
[+] We're done here. Have a nice day!


Cleanup completed successfully!

osboxes@osboxes:~/jpeg$
```

Figure 4: Images of running the bash script on input corpus:

```
osboxes@osboxes:~/jpeg$ ls tmin_outputs/
logo.min.jpg  not_kitty.min.jpg  rodeo.min.jpg
osboxes@osboxes:~/jpeg$
```

Figure 5: Directory listing of directory containing the output of afl-tmin.

4. Reducing corpus with afl-cmin

To reduce the input corpus I ran the command afl-cmin -i /home/osboxes/jpeg/inputs/ -o /home/osboxes/jpeg/cmin_outputs -f new.jpg /home/osboxes/jpeg/src/src/jpeg new.jpg.

This command ran very quickly. The results are below:

Figure 6: Image of results of afl-cmin command.

5. Running AFL on the target binary

When first running there was an error regarding core dump notifications that needed to be fixed.

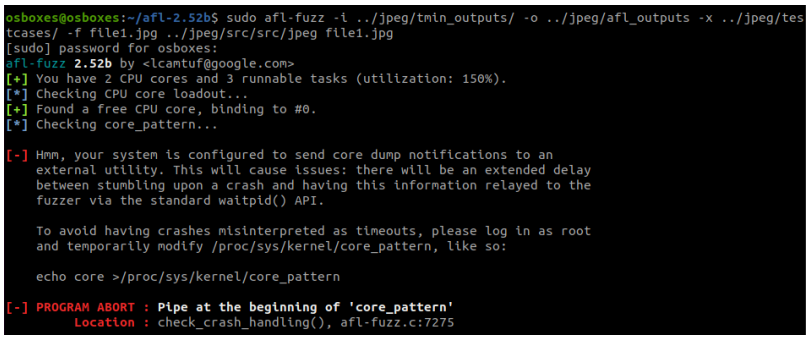

Figure 7: Image of core dump errors.

After following instructions from the error, the issue was fixed, but there was another error regarding a large dictionary file.

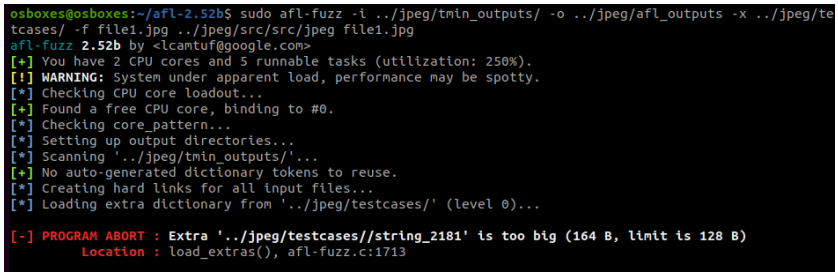

Figure 8: Image of large dictionary file error.

To fix this issue I deleted this file from the dictionary location. After re-runnng everything worked as expected.

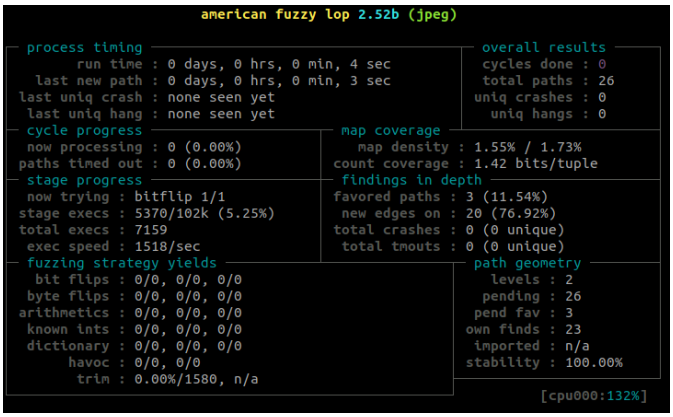

Figure 9: Image of first run of afl-fuzz to make sure all was OK.

6. Creating a dictionary

To create the dictionary cases I used script from the Push the Red Button blog (https://moyix.blogspot.com/2016/07/fuzzing-with-afl-is-an-art.html). I slightly modified the script to check if testcases dir exists and if not, create it before running.

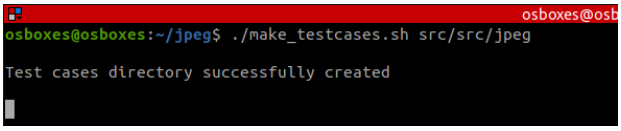

Figure 10: Image of make_testcases.sh bash script to create a dictionary.

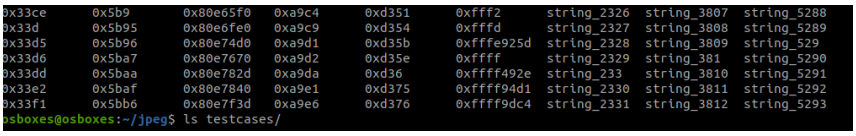

Figure 11: Image of directory listing of the dictionary testcases directory.

7. Finding some crashes

To run the fuzzer, I used the command `sudo afl-fuzz -i ../jpeg/<input_folder>/ -o ../jpeg/<output_folder -x ../jpeg/testcases -f file1.jpg ../jpeg/src/src/jpeg file1.jpg`

The initial run with afl-fuzz was with just the included input jpegs and the standard one that comes with afl-fuzz as noted above. The dictionary created in the earlier step was also use. I let the tool run for almost 9 hrs and it got 1 crash and 1 hang.

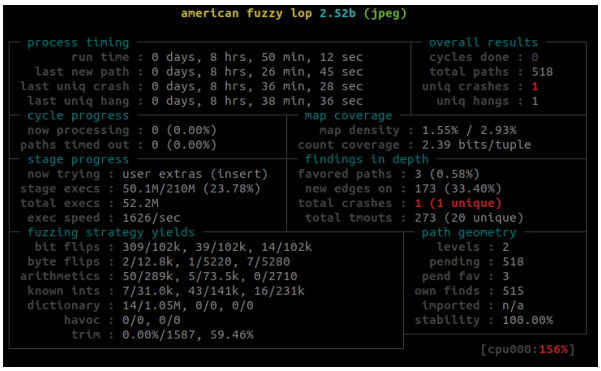

Figure 12: Image of afl-fuzz output for initial corpus before stopping run.

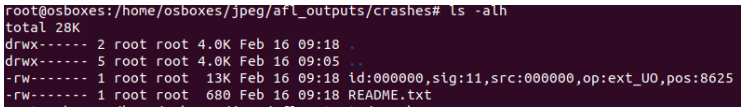

Figure 13: Image of crash directory for initial run.

Next, I ran afl-fuzz with the tmin output from the three-image corpus as the inputs along with the dictionary created. I ran this for a little over 13 hrs and got 0 crashes and 2 unique hangs.

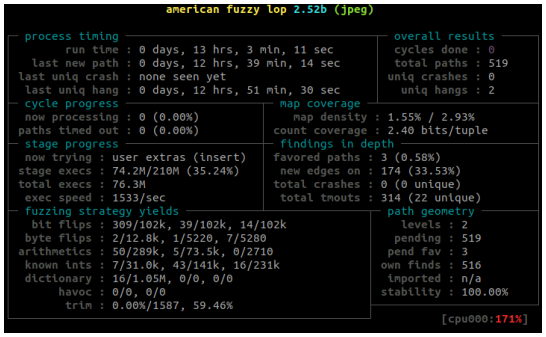

Figure 14: Figure 11: Image of afl-fuzz output for tmin output corpus before stopping run.

Next, I ran afl-fuzz with the afl-cmin produced outputs as the inputs along with the dictionary. I ran this for about 11.5hrs and received 1 crash and 2 unique hangs.

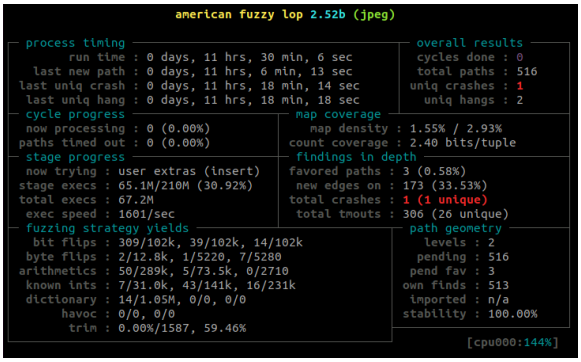

Figure 15: Image of afl-fuzz output for afl-cmin output corpus before stopping run.

```
root@osboxes:/home/osboxes/jpeg/afl_output3/crashes# ls -alh
total 28K
drwx------ 2 root root 4.0K Feb 17 10:51 .
drwx------ 5 root root 4.0K Feb 17 10:39 ..
-rw------- 1 root root  13K Feb 17 10:51 id:000000,sig:11,src:000000,op:ext_UO,pos:7396
-rw------- 1 root root  680 Feb 17 10:51 README.txt
```

Figure 16: Image of crash directory for cmin input run.

Next, I ran afl-fuzz with a custom corpus/jpeg inputs found on: https://lcamtuf.coredump.cx/afl/demo/ . There were 1777 input images that were supposedly minimized previously in the corpus. They were described as “full” jpeg images. I ran this for over 24 hours and this run produced 20 unique crashes/161 total and 8 unique hangs/310 total. The crashes are attached in the assignment submission labeled crashes.tar.gz.

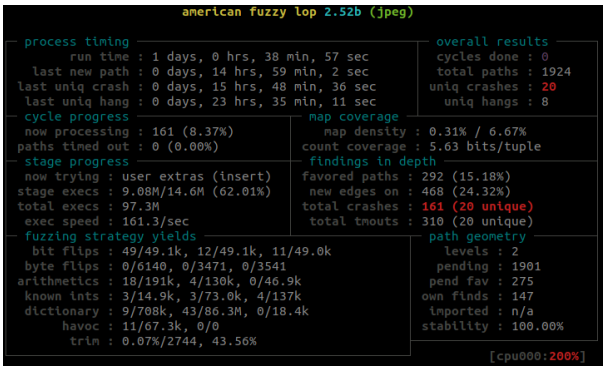

Figure 17: Image of afl-fuzz output for custom corpus before stopping run.

```
root@osboxes:/home/osboxes/jpeg/afl_external_output/crashes# ls
id:000000,sig:11,src:000001,op:ext_UI,pos:198  id:000011,sig:11,src:000074,op:ext_UI,pos:175
id:000001,sig:11,src:000006,op:ext_UI,pos:175  id:000012,sig:06,src:000074,op:ext_UI,pos:287
id:000002,sig:11,src:000006,op:ext_UI,pos:177  id:000013,sig:11,src:000089,op:ext_UI,pos:148
id:000003,sig:11,src:000006,op:ext_UI,pos:287  id:000014,sig:11,src:000093,op:ext_UI,pos:287
id:000004,sig:06,src:000006,op:ext_UI,pos:287  id:000015,sig:06,src:000093,op:ext_UI,pos:287
id:000005,sig:11,src:000006,op:ext_UI,pos:342  id:000016,sig:11,src:000093,op:ext_UI,pos:308
id:000006,sig:11,src:000006,op:ext_UI,pos:430  id:000017,sig:11,src:000118,op:ext_UI,pos:175
id:000007,sig:11,src:000007,op:ext_UI,pos:287  id:000018,sig:06,src:000118,op:ext_UI,pos:287
id:000008,sig:06,src:000007,op:ext_UI,pos:287  id:000019,sig:06,src:000132,op:ext_UI,pos:287
id:000009,sig:11,src:000007,op:ext_UI,pos:341  README.txt
id:000010,sig:11,src:000072,op:ext_UO,pos:177
```

Figure 18: Image of crash directory for custom corpus run.

Lastly, I ran afl-fuzz with a custom corpus/jpeg inputs found on: https://lcamtuf.coredump.cx/afl/demo/ . I used the edge case images this time. There were 304 input images that were supposedly minimized previously in the corpus. I ran afl-fuzz for almost 16 hrs with this corpus and received 13 unique crashes/195 total and 10 unique hangs/374 total. The crashes looked to be the same as the previous run but only 13 of them.

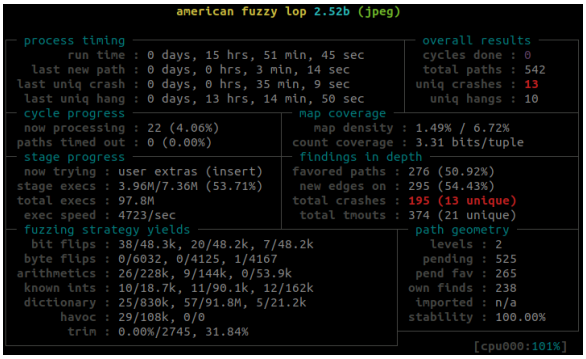

Figure 19: Image of run of jpeg edge cases.

```
root@osboxes:/home/osboxes/jpeg/edges_output# ll crashes/
total 64
drwx------ 2 root root 4096 Feb 19 20:20 ./
drwx------ 5 root root 4096 Feb 19 05:03 ../
-rw------- 1 root root  312 Feb 19 05:56 id:000000,sig:11,src:000003,op:ext_UI,pos:177
-rw------- 1 root root  312 Feb 19 05:56 id:000001,sig:11,src:000003,op:ext_UI,pos:198
-rw------- 1 root root  312 Feb 19 06:19 id:000002,sig:11,src:000005,op:ext_UI,pos:175
-rw------- 1 root root  312 Feb 19 06:19 id:000003,sig:11,src:000005,op:ext_UI,pos:177
-rw------- 1 root root  508 Feb 19 06:50 id:000004,sig:11,src:000007,op:ext_UI,pos:287
-rw------- 1 root root  508 Feb 19 06:50 id:000005,sig:06,src:000007,op:ext_UI,pos:287
-rw------- 1 root root  508 Feb 19 06:54 id:000006,sig:11,src:000007,op:ext_UI,pos:354
-rw------- 1 root root  508 Feb 19 07:27 id:000007,sig:06,src:000008,op:ext_UI,pos:287
-rw------- 1 root root  508 Feb 19 08:04 id:000008,sig:06,src:000011,op:ext_UI,pos:287
-rw------- 1 root root  312 Feb 19 09:38 id:000009,sig:11,src:000017,op:ext_UI,pos:177
-rw------- 1 root root  508 Feb 19 10:38 id:000010,sig:06,src:000019,op:ext_UI,pos:287
-rw------- 1 root root  508 Feb 19 14:14 id:000011,sig:06,src:000020,op:ext_UI,pos:287
-rw------- 1 root root  508 Feb 19 20:20 id:000012,sig:06,src:000021,op:ext_UI,pos:287
-rw------- 1 root root  680 Feb 19 05:56 README.txt
```

Figure 20: Images of crashes directory.

8. Getting it to run in parallel

I originally ran the tests above utilizing 4GB RAM and 2 CPUs 1 core each in a VMware Pro VM. For this test I gave the VM 2 CPUs with 2 cores. I ran 1 master process with 3 slaves. While it was only using half of my CPU cores, I did not run this for very long as it was very taxing (heat/Fan and RAM) on my computer while trying to accomplish other things.

To run the fuzzer in parallel first I created a sync directory named sync_dir. Then ran the master processes followed by the three slaves. I gave them the id scheme of fuzzer##.

Because the fuzzer needs a separate temporary file I just named them parallelFile#.jpg. I could have also let afl-fuzz do that automatically.

The following are the commands I used:

```
sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x
../jpeg/testcases -M fuzzer01 -f parallelFile1.jpg ../jpeg/src/src/jpeg
parallelFile1.jpg
```

```
sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x
../jpeg/testcases -S fuzzer02 -f parallelFile2.jpg ../jpeg/src/src/jpeg
parallelFile2.jpg
```

```
sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x
../jpeg/testcases -S fuzzer03 -f parallelFile3.jpg ../jpeg/src/src/jpeg
parallelFile3.jpg
```

```
sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x
../jpeg/testcases -S fuzzer04 -f parallelFile4.jpg ../jpeg/src/src/jpeg
parallelFile4.jpg
```

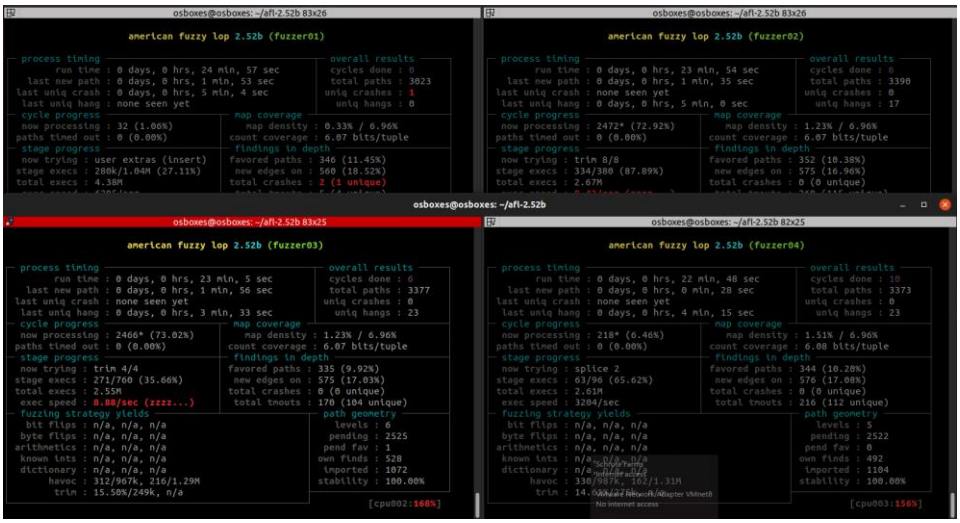

Figure 21: Image of Master and 3 Slave instances of afl_fuzz running.

```
root@osboxes:/home/osboxes/jpeg/sync_dir# ll
total 276
drwx------  9 root     root      4096 Feb 19 03:18 ./
drwxrwxr-x 17 osboxes osboxes    4096 Feb 19 03:01 ../
drwx------  2 root     root      4096 Feb 19 03:01 crashes/
-rw-------  1 root     root     65536 Feb 19 03:01 fuzz_bitmap
drwx------  6 root     root      4096 Feb 19 03:16 fuzzer01/
drwx------  6 root     root      4096 Feb 19 03:17 fuzzer02/
drwx------  6 root     root      4096 Feb 19 03:18 fuzzer03/
drwx------  6 root     root      4096 Feb 19 03:18 fuzzer04/
-rw-------  1 root     root       820 Feb 19 03:02 fuzzer_stats
drwx------  2 root     root      4096 Feb 19 03:01 hangs/
-rw-------  1 root     root       435 Feb 19 03:02 plot_data
drwx------  3 root     root    176128 Feb 19 03:02 queue/
root@osboxes:/home/osboxes/jpeg/sync_dir#
```

Figure 22: sync_dir folder showing all the instances of the fuzzer.

```
root@osboxes:/home/osboxes/jpeg/sync_dir/fuzzer01# ps -ef | grep -i afl-fuzz
root        2226    2130  0 03:16 pts/0    00:00:00 sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -M fuzzer01 -f parallelFile1.jpg ../jpeg/src/src/jpeg parallelFile1
.jpg
root        2227    2226  7 03:16 pts/0    00:00:56 afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -M fuzzer01 -f parallelFile1.jpg ../jpeg/src/src/jpeg parallelFile1.jpg
root      138372    2145  0 03:17 pts/1    00:00:00 sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -S fuzzer02 -f parallelFile2.jpg ../jpeg/src/src/jpeg parallelFile2
.jpg
root      139457  138372  9 03:17 pts/1    00:01:13 afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -S fuzzer02 -f parallelFile2.jpg ../jpeg/src/src/jpeg parallelFile2.jpg
root      329472  172382  0 03:17 pts/4    00:00:00 sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -S fuzzer03 -f parallelFile3.jpg ../jpeg/src/src/jpeg parallelFile3
.jpg
root      329498  329472  8 03:17 pts/4    00:00:59 afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -S fuzzer03 -f parallelFile3.jpg ../jpeg/src/src/jpeg parallelFile3.jpg
root      467567  184793  0 03:18 pts/5    00:00:00 sudo afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -S fuzzer04 -f parallelFile4.jpg ../jpeg/src/src/jpeg parallelFile4
.jpg
root      488058  467567  9 03:18 pts/5    00:01:01 afl-fuzz -i ../jpeg/external_inputs/ -o ../jpeg/sync_dir -x ../jpeg/testcases -S fuzzer04 -f parallelFile4.jpg ../jpeg/src/src/jpeg parallelFile4.jpg
```

Figure 23: Image of process listing of 4 instances of afl-fuzz running in parallel.