

In a new build folder, do the below for 5 rounds

CC=/usr/local/bin/afl-gcc CXX=/usr/local/bin/afl-g++ cmake ..

make

afl-fuzz -i ../testcases -o ../findings ./afldemo

rm -rf ./

Round 1:

For part 1: **Fuzz testing on a vulnerable program**

process timing		overall results
run time : 0 days, 0 hrs, 0 min, 14 sec		cycles done : 1
last new path : 0 days, 0 hrs, 0 min, 2 sec		total paths : 26
last uniq crash : 0 days, 0 hrs, 0 min, 11 sec		uniq crashes : 6
last uniq hang : none seen yet		uniq hangs : 0
cycle progress	map coverage	
now processing : 17.1 (65.4%)	map density : 0.04% / 0.13%	
paths timed out : 0 (0.00%)	count coverage : 1.00 bits/tuple	
stage progress	findings in depth	
now trying : havoc	avored paths : 17 (65.38%)	
stage execs : 22.4k/32.8k (68.47%)	new edges on : 25 (96.15%)	
total execs : 69.2k	total crashes : 239 (6 unique)	
exec speed : 4687/sec	total tmouts : 0 (0 unique)	
fuzzing strategy yields	path geometry	
bit flips : n/a, n/a, n/a	levels : 4	
byte flips : n/a, n/a, n/a	pending : 9	
arithmetics : n/a, n/a, n/a	pend fav : 1	
known ints : n/a, n/a, n/a	own finds : 23	
dictionary : n/a, n/a, n/a	imported : 0	
havoc/splice : 21/37.7k, 0/8805	stability : 100.00%	
py/custom : 0/0, 0/0		
trim : 3.03%/56, n/a		
		[cpu000:100%]

(1) how long was the fuzzing process

14 sec

(2) how many inputs AFL found that could make the program crash or hang.

6

wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes\$ ls

id:000000,sig:06,src:000002,time:80,op:havoc,rep:8

id:000001,sig:11,src:000002,time:147,op:havoc,rep:2

id:000002,sig:06,src:000002,time:1302,op:havoc,rep:4

id:000003,sig:11,src:000002,time:1317,op:havoc,rep:16

id:000004,sig:06,src:000002,time:1585,op:havoc,rep:16

id:000005,sig:06,src:000002,time:3449,op:havoc,rep:4

For part 2: **Isolating and fixing the bugs**

(1) *describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;*

The AFL inputs are in folder /findings/default/crashes, execute the program with this input , and use printf to identify the issue,

- u -1 strin, It seems like N should not be a negative number, it causes malloc errors
- u -18 aBCd 18 aBC11, it seems like negative N causes malloc issues
- u 11 {BCddddd@dddddMddddd0aB9zde, it seems like characters, such as {, other than small letters causes translating issues
- u 10 aBCded 10 aBCde0 aBC aBCd 10 aBCde0 aBCd, it seems like spaces and capital letters causes translating issues
- u 0aBCu 8 4 aBCde, it seems like spaces and capital letters causes translating issues
- u 0aBCu 8 4 aBCde aBCdeed, it seems like spaces and capital letters causes translating issues

(2) *describe how you fixed the bugs and why your fixes worked.*

For INPUT in case a and b: Bound the input N to positive integers, bound the N so that later malloc won't have issues

```
len = strtol(input + 2, &rest, 10); // how many characters of the string to upper-case
if(len < 0){
    printf("Specified length %d should be positive\n", len);
    return 1;
}
```

For INPUT in case c, d, e, f, Bound the input string to only small letters, so that ASCII translation works

```
}
for (i = 0; i != len; i++)
{
    if(rest[i] >= 'a' || rest[i] <= 'z'){
        printf("Letters should be small letters, this program only handles ASCII\n");
        return 1;
    }
    out[i] = rest[i] - 32; // only handles ASCII
}
```

Round 2:

For part 1: Fuzz testing on a vulnerable program

```
american fuzzy lop ++3.01a (default) [fast] {0}

process timing
  run time : 0 days, 0 hrs, 0 min, 16 sec
  last new path : 0 days, 0 hrs, 0 min, 16 sec
  last uniq crash : 0 days, 0 hrs, 0 min, 15 sec
  last uniq hang : none seen yet
cycle progress
  now processing : 6.19 (85.7%)
  paths timed out : 0 (0.00%)
stage progress
  now trying : MOpt-havoc
  stage execs : 0/256 (0.00%)
  total execs : 88.5k
  exec speed : 5548/sec
fuzzing strategy yields
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  havoc/splice : 4/44.4k, 1/44.0k
  py/custom : 0/0, 0/0
  trim : 0.00%/9, n/a
overall results
  cycles done : 12
  total paths : 7
  uniq crashes : 1
  uniq hangs : 0
map coverage
  map density : 0.02% / 0.03%
  count coverage : 1.00 bits/tuple
findings in depth
  favored paths : 6 (85.71%)
  new edges on : 6 (85.71%)
  total crashes : 6 (1 unique)
  total tmouts : 0 (0 unique)
path geometry
  levels : 2
  pending : 0
  pend fav : 0
  own finds : 4
  imported : 0
  stability : 100.00%
[cpu000: 50%]
```

(1) how long was the fuzzing process;

16 sec

(2) how many inputs AFL found that could make the program crash or hang.

```
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ ls
id:0000000,sig:06,src:0000000+0000002,time:783,op:splice,rep:16_README.txt
```

1 input for crash:

u u 3 >B

For part 2: Isolating and fixing the bugs

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

Input "u u 3 >B", execute the program with this input, and use printf to identify the issue, N should be an integer instead of a character

(2) describe how you fixed the bugs and why your fixes worked.

Bound N to a number, atoi returns 0 if it is a number 0 or not a number. Get the first char to see if the N is 0, if N is not 0 and atoi returns 0, then the input is not valid.


```

upper_case_command
char *rest;
len = strtol(input + 2, &rest, 10); // how many characters of the string to upper-case
newlen = atoi(input+2);
int firstN = input[2] - '0';
if(newlen == 0 && firstN != 0){
    printf("N should be a number\n");
    return 1;
}

```

Round 3:

For part 1: **Fuzz testing on a vulnerable program**

american fuzzy lop ++3.01a (default) [fast] {0}			
process timing		overall results	
run time : 0 days, 0 hrs, 0 min, 20 sec		cycles done : 6	
last new path : 0 days, 0 hrs, 0 min, 15 sec		total paths : 10	
last uniq crash : 0 days, 0 hrs, 0 min, 10 sec		uniq crashes : 1	
last uniq hang : none seen yet		uniq hangs : 0	
cycle progress		map coverage	
now processing : 3.7 (30.0%)		map density : 0.02% / 0.04%	
paths timed out : 0 (0.00%)		count coverage : 1.00 bits/tuple	
stage progress		findings in depth	
now trying : M0pt-splice 5		favored paths : 8 (80.00%)	
stage execs : 1/36 (2.78%)		new edges on : 9 (90.00%)	
total execs : 106k		total crashes : 3 (1 unique)	
exec speed : 5037/sec		total tmouts : 0 (0 unique)	
fuzzing strategy yields		path geometry	
bit flips : n/a, n/a, n/a		levels : 2	
byte flips : n/a, n/a, n/a		pending : 1	
arithmetics : n/a, n/a, n/a		pend fav : 0	
known ints : n/a, n/a, n/a		own finds : 7	
dictionary : n/a, n/a, n/a		imported : 0	
havoc/splice : 6/55.9k, 2/50.1k		stability : 100.00%	
py/custom : 0/0, 0/0		[cpu000: 75%]	
trim : 0.00%/23, n/a		^C	

(1) how long was the fuzzing process;

20 sec

(2) how many inputs AFL found that could make the program crash or hang.

1 input

```

wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ ls
id:000000,sig:06,src:000007+000002,time:10698,op:splice,rep:16 README.txt
wenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$ cat id:000000,sig:06,src:000007+000002,time:10698,op:splice,rep:16
000000 10 aB*****'00u aB 0 *****p000A0'00u 10 aBB*****aBwenhui@wenhui:~/Downloads/afl-demo/findings/default/crashes$

```

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

Input “u 0000u 10 aB00000000000000000000u aB0 0 0000000000p000Ã0000u 10 aBB000000000000aB”, execute the program with this input , and use printf to identify the issue, input string should be small letters

(2) describe how you fixed the bugs and why your fixes worked.

All characters in string must be between ‘a’ and ‘z’.

```

    }
    for (i = 0; i != len; i++)
    {
        if(!((rest[i] - 'a' >= 0) && (rest[i] - 'z' <= 0))){
            printf("Letters should be small letters, this program only handles ASCII\n");
            return 1;
        }
        out[i] = rest[i] - 32; // only handles ASCII
    }

```

Round 4:

For part 1:Fuzz testing on a vulnerable program

```

american fuzzy lop ++3.01a (default) [fast] {0}
-- process timing --
    run time : 0 days, 0 hrs, 0 min, 53 sec
    last new path : 0 days, 0 hrs, 0 min, 0 sec
    last uniq crash : 0 days, 0 hrs, 0 min, 30 sec
    last uniq hang : none seen yet
-- cycle progress --
    now processing : 24.5 (80.0%)
    paths timed out : 0 (0.00%)
-- stage progress --
    now trying : M0pt-splice 14
    stage execs : 10/55 (18.18%)
    total execs : 264k
    exec speed : 4997/sec
-- fuzzing strategy yields --
    bit flips : n/a, n/a, n/a
    byte flips : n/a, n/a, n/a
    arithmetics : n/a, n/a, n/a
    known ints : n/a, n/a, n/a
    dictionary : n/a, n/a, n/a
    havoc/splice : 26/108k, 4/156k
    py/custom : 0/0, 0/0
    trim : 0.24%/85, n/a
-- overall results --
    cycles done : 6
    total paths : 30
    uniq crashes : 3
    uniq hangs : 0
-- map coverage --
    map density : 0.05% / 0.13%
    count coverage : 1.10 bits/tuple
-- findings in depth --
    favored paths : 27 (90.00%)
    new edges on : 29 (96.67%)
    total crashes : 24 (3 unique)
    total tmouts : 0 (0 unique)
-- path geometry --
    levels : 4
    pending : 3
    pend fav : 0
    own finds : 27
    imported : 0
    stability : 100.00%
-- [cpu000: 75%]
^C

```

(1) how long was the fuzzing process;

53 sec

(2) how many inputs AFL found that could make the program crash or hang.

3

Round 5:

For part 1: **Fuzz testing on a vulnerable program**

```
american fuzzy lop ++3.01a (default) [fast] {0}

- process timing -
  run time : 0 days, 0 hrs, 2 min, 40 sec
  last new path : 0 days, 0 hrs, 2 min, 40 sec
  last uniq crash : none seen yet
  last uniq hang : none seen yet
- cycle progress -
  now processing : 4.532 (80.0%)
  paths timed out : 0 (0.00%)
- stage progress -
  now trying : M0pt-core-havoc
  stage execs : 0/256 (0.00%)
  total execs : 846k
  exec speed : 5389/sec
- fuzzing strategy yields -
  bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
  havoc/splice : 2/306k, 0/384k
  py/custom : 0/0, 0/0
  trim : 55.70%/17, n/a

- overall results -
  cycles done : 122
  total paths : 5
  uniq crashes : 0
  uniq hangs : 0
- map coverage -
  map density : 0.01% / 0.02%
  count coverage : 1.00 bits/tuple
- findings in depth -
  favored paths : 3 (60.00%)
  new edges on : 3 (60.00%)
  total crashes : 0 (0 unique)
  total tmouts : 0 (0 unique)
- path geometry -
  levels : 2
  pending : 4.29G
  pend fav : 0
  own finds : 2
  imported : 0
  stability : 100.00%

[cpu000:100%]
^C
```

(1) how long was the fuzzing process;

2 mins 40 sec

(2) how many inputs AFL found that could make the program crash or hang.

none

For part 2: **Isolating and fixing the bugs**

(1) describe your process of identifying bugs from AFL-generated inputs and what bugs you found in the program;

N/A

(2) describe how you fixed the bugs and why your fixes worked.

N/A