IDATT2503 - Exercise 3 - Edvard Berdal Eek

18.09.2025

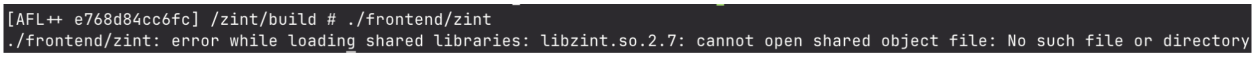
Task 1 - Binary fuzzing with AFL++ in QEMU mode:

Setting up fuzzing environment:

Again, my obstical in completing the assignment was to set up the environment to do the task. I first initialized a AFL++ container the zint folder mounted as a volume with the following command:

```
docker run --rm -it \
  -v "$(pwd)/zint-2.7.1:/zint" \
  aflplusplus/aflplusplus /bin/bash
```

When trying to execute the binaries produced after doing CMake i got this error:



```
[AFL++ e768d84cc6fc] /zint/build # ./frontend/zint
./frontend/zint: error while loading shared libraries: libzint.so.2.7: cannot open shared object file: No such file or directory
```

After conferring with ChatGPT i was advised to add the environment variable '**LD_LIBRARY_PATH**=/zint/build/backend'. The final docker command than became:

```
docker run --rm -it \
  -v "$(pwd)/zint-2.7.1:/zint" \
  -e LD_LIBRARY_PATH=/zint/build/backend \
  aflplusplus/aflplusplus /bin/bash
```

Executing the binaries was now possible. After testing **afl-fuzz -Q ./frontend/zint**, i was prompted with an error requireing me to provide a input and output directory. I then created the directories **out_dir** and finally **in_dir** with a text file including the string **123456789012**. With a bit of tweeking on the command i finally got the AFL++ cli up and running, with a crash registered shortly after. The command that produced the crash with AFL++ was:

```
afl-fuzz -i /zint/in_dir -o /zint/out_dir -Q -- ./frontend/zint -b 13 -i @@
```

Looking at the crash report in **out_dir** with the name: **id:000000,sig:06,src:000001,time:1438,execs:1761,op:havoc,rep:2**, containing the input that crashed the program which was: **++13**. I then used the crash report as the input file for the program when executing it with **gdb**.

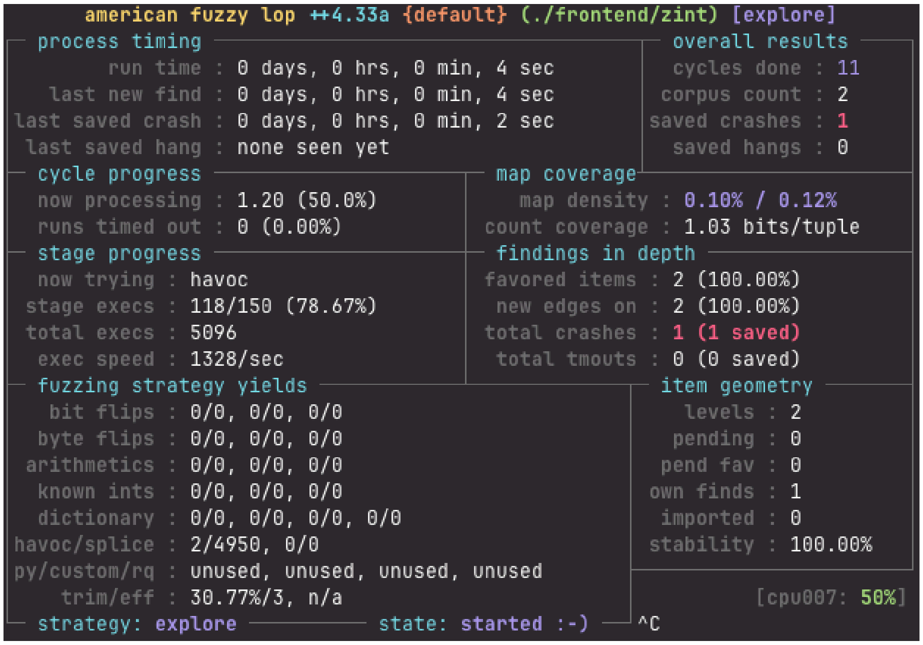
gdb --args ./frontend/zint -b 13 -i /zint/out_dir/default/crashes/id\:000000\,sig\:06\,src\:000001\,time\:1438\,execs\:1761\,op\:havoc\,rep\:2

Task 2 - Follow up questions:

1. **How many total executions and minutes did it take before you found a crash?**

The final fuzzing attempt i did was very quick. It only required 1761 executions which took 1.438 seconds.

2. **What can you say about the fuzzing coverage?**



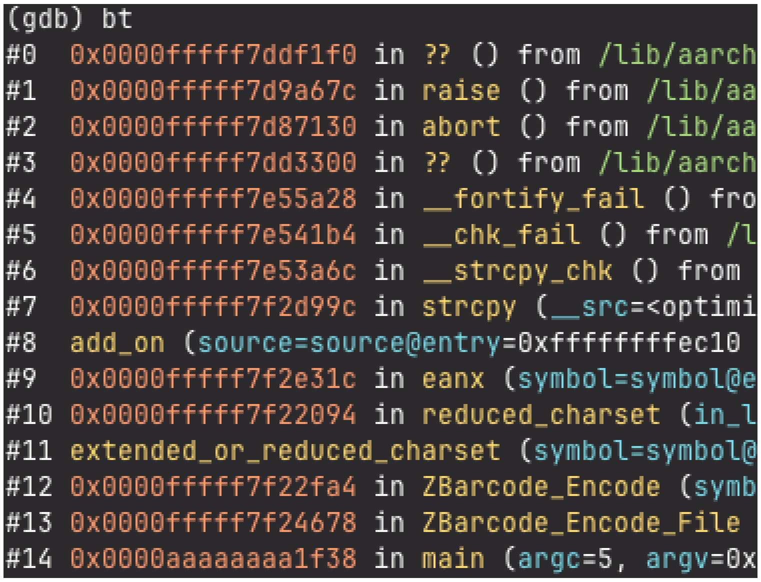
```
       american fuzzy lop ++4.33a {default} (./frontend/zint) [explore]
┌─ process timing ────────────────────────────────────┬─ overall results ───┐
│        run time : 0 days, 0 hrs, 0 min, 4 sec       │  cycles done : 11   │
│   last new find : 0 days, 0 hrs, 0 min, 4 sec       │ corpus count : 2    │
│last saved crash : 0 days, 0 hrs, 0 min, 2 sec       │saved crashes : 1    │
│ last saved hang : none seen yet                     │  saved hangs : 0    │
├─ cycle progress ─────────────────────┬─ map coverage┴─────────────────────┤
│  now processing : 1.20 (50.0%)       │    map density : 0.10% / 0.12%     │
│  runs timed out : 0 (0.00%)          │ count coverage : 1.03 bits/tuple   │
├─ stage progress ─────────────────────┼─ findings in depth ────────────────┤
│  now trying : havoc                  │ favored items : 2 (100.00%)        │
│ stage execs : 118/150 (78.67%)       │  new edges on : 2 (100.00%)        │
│ total execs : 5096                   │ total crashes : 1 (1 saved)        │
│  exec speed : 1328/sec               │  total tmouts : 0 (0 saved)        │
├─ fuzzing strategy yields ────────────┴─────────────┬─ item geometry ──────┤
│   bit flips : 0/0, 0/0, 0/0                        │    levels : 2        │
│  byte flips : 0/0, 0/0, 0/0                        │   pending : 0        │
│ arithmetics : 0/0, 0/0, 0/0                        │  pend fav : 0        │
│  known ints : 0/0, 0/0, 0/0                        │ own finds : 1        │
│  dictionary : 0/0, 0/0, 0/0, 0/0                   │  imported : 0        │
│havoc/splice : 2/4950, 0/0                          │ stability : 100.00%  │
│py/custom/rq : unused, unused, unused, unused       ├──────────────────────┘
│    trim/eff : 30.77%/3, n/a                        │          [cpu007: 50%]
└─ strategy: explore ────────── state: started :-) ──┘^C
```

As the time to find the crash was so short there is not so much to go on as to coverage. It might also be an indicator that the bug was easy to produce and therefore be quite critical.

3. **What was the input AFL++ discovered that caused the crash?**

The input that AFL++ discovered was **++13.**

4. **In which function (in the zint source code) do we see the issue?**



```
(gdb) bt
#0  0x0000fffff7ddf1f0 in ?? () from /lib/aarch
#1  0x0000fffff7d9a67c in raise () from /lib/aa
#2  0x0000fffff7d87130 in abort () from /lib/aa
#3  0x0000fffff7dd3300 in ?? () from /lib/aarch
#4  0x0000fffff7e55a28 in __fortify_fail () fro
#5  0x0000fffff7e541b4 in __chk_fail () from /l
#6  0x0000fffff7e53a6c in __strcpy_chk () from
#7  0x0000fffff7f2d99c in strcpy (__src=<optimi
#8  add_on (source=source@entry=0xffffffffec10
#9  0x0000fffff7f2e31c in eanx (symbol=symbol@e
#10 0x0000fffff7f22094 in reduced_charset (in_l
#11 extended_or_reduced_charset (symbol=symbol@
#12 0x0000fffff7f22fa4 in ZBarcode_Encode (symb
#13 0x0000fffff7f24678 in ZBarcode_Encode_File
#14 0x0000aaaaaaaa1f38 in main (argc=5, argv=0x
```

Both the bug ticket thread and the gdb backtrace indicate that there is an issue with **strcpy** being called in the **add_on** function.

5. **What was the fix proposed by the zint maintainers?**

According to Robin Stuart, the fix to this particular bug was to add a check for multiple + characters. “**I have added a check to the UPC/EAN code to throw an error if more than one + character is included in the input data.”.** However, looking further down the thread it seems to be more issues in the program. The lack of discovering these issues might indicate poor coverage in my fuzzing attempts.