Process

Setup and Execution

```
sudo apt install afl++
```

Install AFL++

```
mkdir in out
```

Create Input and Output Directories

in: input test cases

out: AFL++ output

```
cp examples/* in/
```

Copy Example Test Cases

```
make cc=afl-clang cflags="-fsanitize=address -
fsanitize=undefined -fsanitize=fuzzer-no-link"
```

Compile with AFL++

address: Enables AddressSanitizer for memory error detection

undefined: Enables UndefinedBehavior Sanitizer for undefined behavior detection

fuzzer-no-link: Enables the fuzzer without link-time optimization

```
sudo -i
echo core >/proc/sys/kernel/core_pattern
logout
```

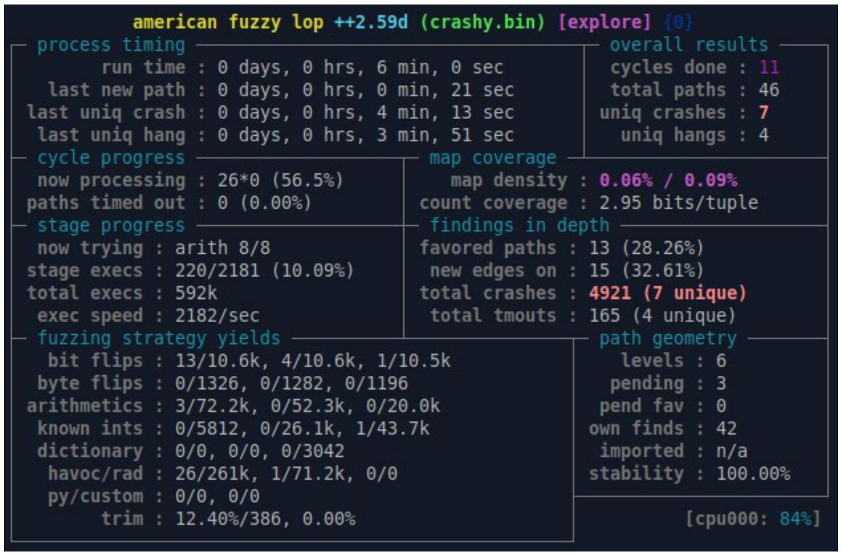
Set Core Dump Pattern

```
afl-fuzz -i in/ -o out/ -m none -- ./crashy.bin @@
```

Run AFL++ Fuzzer

Note: command must include `-m none` *to fix ‘*`PROGRAM ABORT: Fork server crashed`*’ error* [1]

AFL++ Output

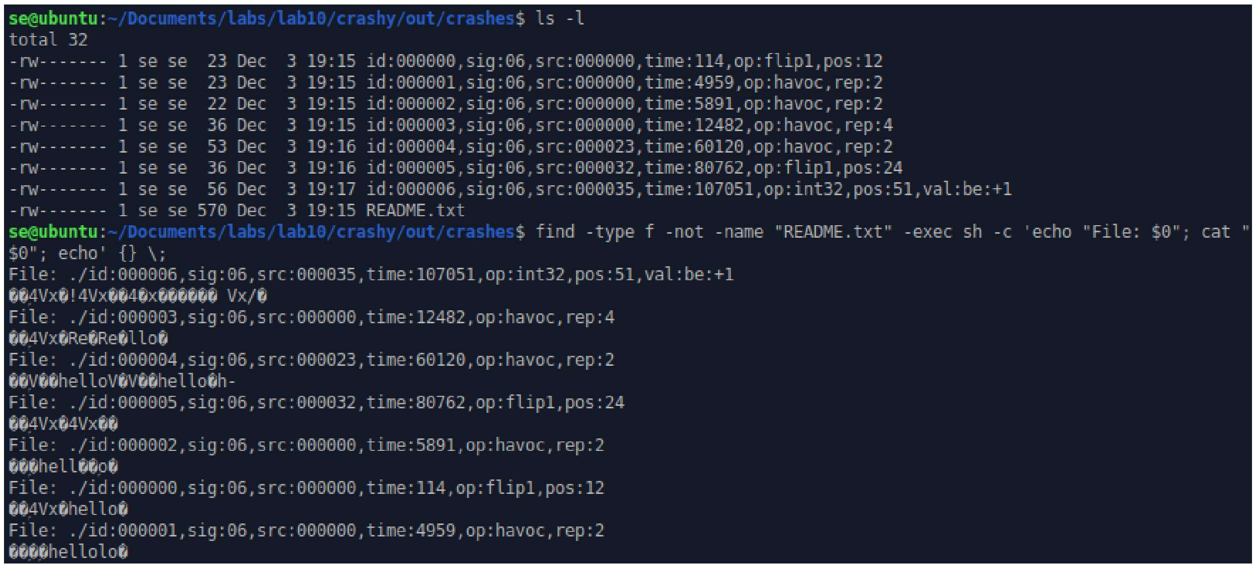


```
          american fuzzy lop ++2.59d (crashy.bin) [explore] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 6 min, 0 sec       │  cycles done : 11    │
│   last new path : 0 days, 0 hrs, 0 min, 21 sec      │  total paths : 46    │
│ last uniq crash : 0 days, 0 hrs, 4 min, 13 sec      │ uniq crashes : 7     │
│  last uniq hang : 0 days, 0 hrs, 3 min, 51 sec      │   uniq hangs : 4     │
├─ cycle progress ───────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 26*0 (56.5%)     │    map density : 0.06% / 0.09%        │
│ paths timed out : 0 (0.00%)        │ count coverage : 2.95 bits/tuple      │
├─ stage progress ───────────────────┼─ findings in depth ───────────────────┤
│  now trying : arith 8/8            │ favored paths : 13 (28.26%)           │
│ stage execs : 220/2181 (10.09%)    │  new edges on : 15 (32.61%)           │
│ total execs : 592k                 │ total crashes : 4921 (7 unique)       │
│  exec speed : 2182/sec             │  total tmouts : 165 (4 unique)        │
├─ fuzzing strategy yields ──────────┴─────────────┬─ path geometry ────────┤
│   bit flips : 13/10.6k, 4/10.6k, 1/10.5k         │    levels : 6          │
│  byte flips : 0/1326, 0/1282, 0/1196             │   pending : 3          │
│ arithmetics : 3/72.2k, 0/52.3k, 0/20.0k          │  pend fav : 0          │
│  known ints : 0/5812, 0/26.1k, 1/43.7k           │ own finds : 42         │
│  dictionary : 0/0, 0/0, 0/3042                   │  imported : n/a        │
│   havoc/rad : 26/261k, 1/71.2k, 0/0              │ stability : 100.00%    │
│   py/custom : 0/0, 0/0                           ├────────────────────────┘
│        trim : 12.40%/386, 0.00%                  │          [cpu000: 84%]
└──────────────────────────────────────────────────┘
```

[1] https://github.com/ocaml-multicore/ocaml-multicore/issues/497

Writeup

For lab 10, I executed a fuzzing process to the "Crashy" program using AFL++. The goal was to identify crashes, with a particular focus on potential exploitable vulnerabilities, such as buffer overflows. As shown from the screenshot above, AFL++ found 7 unique crashes and 4 unique hangs. A list of the crashes can be seen below.



```
se@ubuntu:~/Documents/labs/lab10/crashy/out/crashes$ ls -l
total 32
-rw------- 1 se se  23 Dec  3 19:15 id:000000,sig:06,src:000000,time:114,op:flip1,pos:12
-rw------- 1 se se  23 Dec  3 19:15 id:000001,sig:06,src:000000,time:4959,op:havoc,rep:2
-rw------- 1 se se  22 Dec  3 19:15 id:000002,sig:06,src:000000,time:5891,op:havoc,rep:2
-rw------- 1 se se  36 Dec  3 19:15 id:000003,sig:06,src:000000,time:12482,op:havoc,rep:4
-rw------- 1 se se  53 Dec  3 19:16 id:000004,sig:06,src:000023,time:60120,op:havoc,rep:2
-rw------- 1 se se  36 Dec  3 19:16 id:000005,sig:06,src:000032,time:80762,op:flip1,pos:24
-rw------- 1 se se  56 Dec  3 19:17 id:000006,sig:06,src:000035,time:107051,op:int32,pos:51,val:be:+1
-rw------- 1 se se 570 Dec  3 19:15 README.txt
se@ubuntu:~/Documents/labs/lab10/crashy/out/crashes$ find -type f -not -name "README.txt" -exec sh -c 'echo "File: $0"; cat "
$0"; echo' {} \;
File: ./id:000006,sig:06,src:000035,time:107051,op:int32,pos:51,val:be:+1
��4Vx�!4Vx��4�x������ Vx/�
File: ./id:000003,sig:06,src:000000,time:12482,op:havoc,rep:4
��4Vx�Re�Re�llo�
File: ./id:000004,sig:06,src:000023,time:60120,op:havoc,rep:2
��V��helloV�V��hello�h-
File: ./id:000005,sig:06,src:000032,time:80762,op:flip1,pos:24
��4Vx�4Vx��
File: ./id:000002,sig:06,src:000000,time:5891,op:havoc,rep:2
���hell��o�
File: ./id:000000,sig:06,src:000000,time:114,op:flip1,pos:12
��4Vx�hello�
File: ./id:000001,sig:06,src:000000,time:4959,op:havoc,rep:2
����hellolo�
```

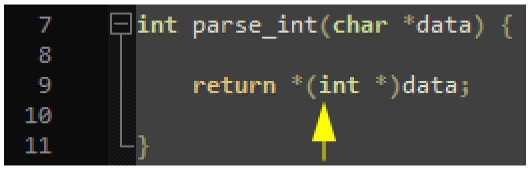
However, after an analysis of these crashes, it was determined that they share a similar root cause. It is important to note that AFL++'s definition of 'unique' is more related to the exploration of the program's execution paths and diversity of inputs, rather than the specific nature of the crashes. As such, despite having similar causes, AFL++ treats them as unique since they take different code paths within the target program. Therefore, to maintain conciseness in this report, a breakdown of the vulnerability is consolidated and outlined below.

Cause of Crashes

All identified crashes are rooted in a heap buffer overflow within the `parse_string` function of the Crashy program. This vulnerability arises from the `parse_string` function using `strcpy` (line 33) to copy the input string into a dynamically allocated buffer (`str`). However, the allocated buffer (`str`) is not large enough to hold the copied string, which leads to a heap buffer overflow.

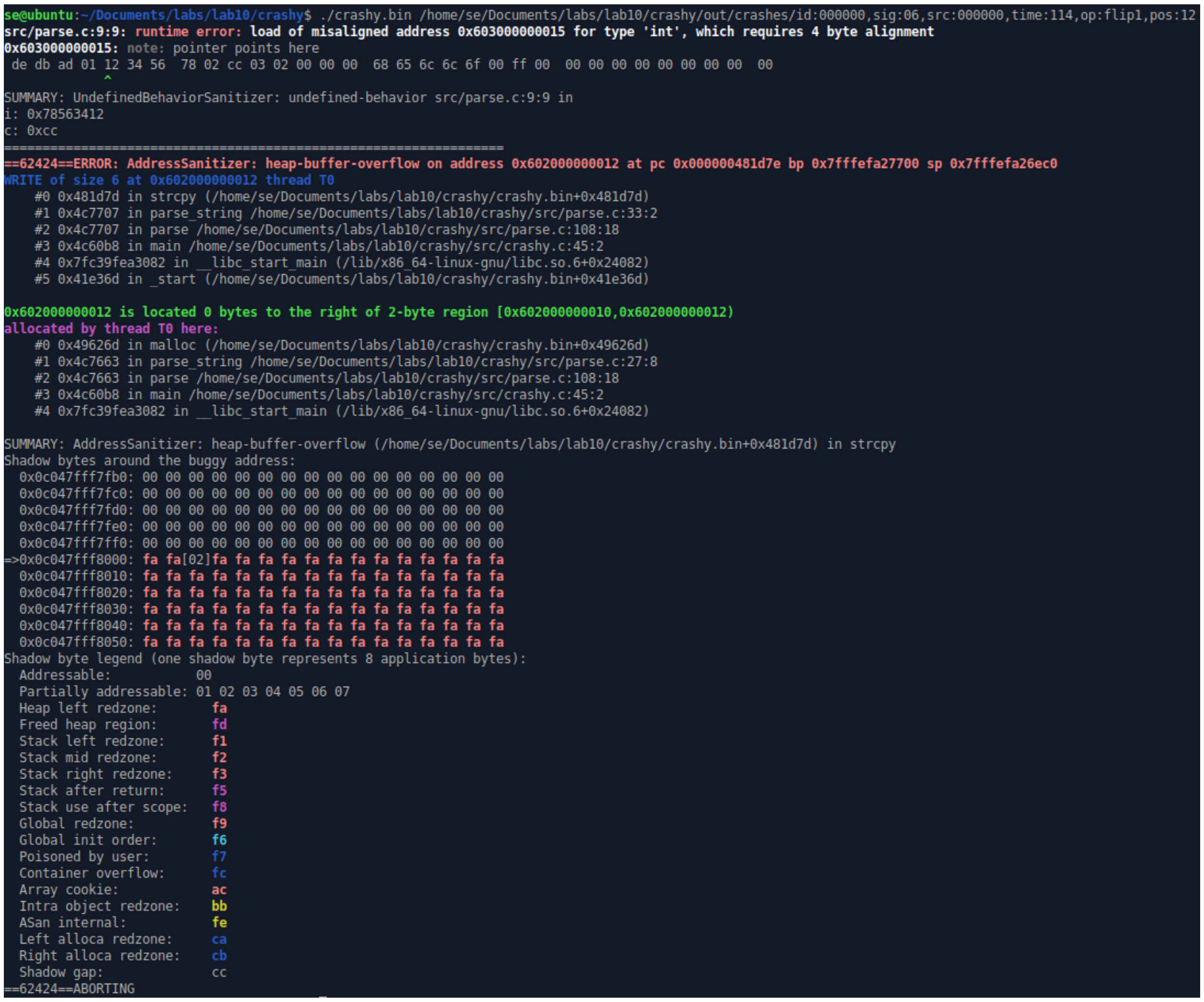
Crash Locations

Each crash occurs in the `parse_int` function within the `parse.c` file, specifically at line 9 (`src/parse.c:9:9`).



```
 7  int parse_int(char *data) {
 8
 9      return *(int *)data;
10
11  }
```

Crashing Sample



```
se@ubuntu:~/Documents/labs/lab10/crashy$ ./crashy.bin /home/se/Documents/labs/lab10/crashy/out/crashes/id:000000,sig:06,src:000000,time:114,op:flip1,pos:12
src/parse.c:9:9: runtime error: load of misaligned address 0x603000000015 for type 'int', which requires 4 byte alignment
0x603000000015: note: pointer points here
 de db ad 01 12 34 56  78 02 cc 03 02 00 00 00  68 65 6c 6c 6f 00 ff 00  00 00 00 00 00 00 00 00  00
             ^
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior src/parse.c:9:9 in
i: 0x78563412
c: 0xcc
=================================================================
==62424==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000012 at pc 0x000000481d7e bp 0x7fffefa27700 sp 0x7fffefa26ec0
WRITE of size 6 at 0x602000000012 thread T0
    #0 0x481d7d in strcpy (/home/se/Documents/labs/lab10/crashy/crashy.bin+0x481d7d)
    #1 0x4c7707 in parse_string /home/se/Documents/labs/lab10/crashy/src/parse.c:33:2
    #2 0x4c7707 in parse /home/se/Documents/labs/lab10/crashy/src/parse.c:108:18
    #3 0x4c60b8 in main /home/se/Documents/labs/lab10/crashy/src/crashy.c:45:2
    #4 0x7fc39fea3082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24082)
    #5 0x41e36d in _start (/home/se/Documents/labs/lab10/crashy/crashy.bin+0x41e36d)

0x602000000012 is located 0 bytes to the right of 2-byte region [0x602000000010,0x602000000012)
allocated by thread T0 here:
    #0 0x49626d in malloc (/home/se/Documents/labs/lab10/crashy/crashy.bin+0x49626d)
    #1 0x4c7663 in parse_string /home/se/Documents/labs/lab10/crashy/src/parse.c:27:8
    #2 0x4c7663 in parse /home/se/Documents/labs/lab10/crashy/src/parse.c:108:18
    #3 0x4c60b8 in main /home/se/Documents/labs/lab10/crashy/src/crashy.c:45:2
    #4 0x7fc39fea3082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24082)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/home/se/Documents/labs/lab10/crashy/crashy.bin+0x481d7d) in strcpy
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa[02]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==62424==ABORTING
```

Note: Crashing samples for each input are included in the folder for further analysis.