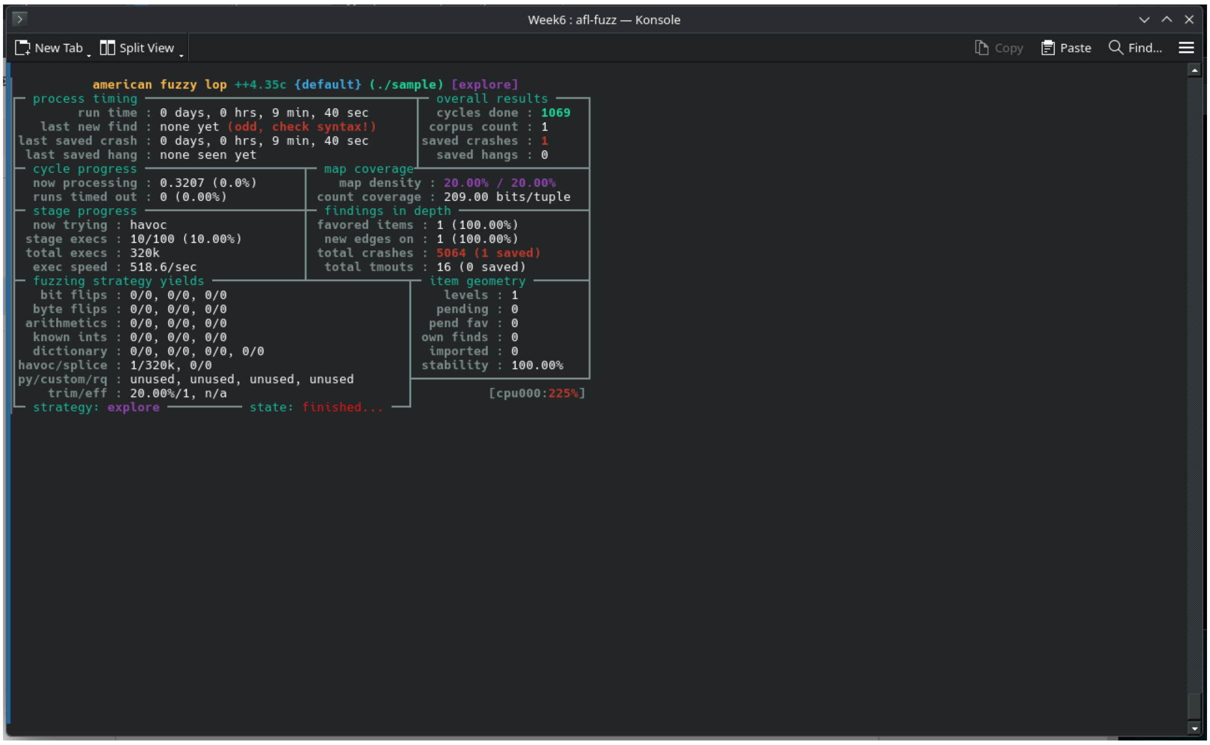
Week 6 Task 2

1. Take a screenshot of AFL++ report screen of your own execution with crashes



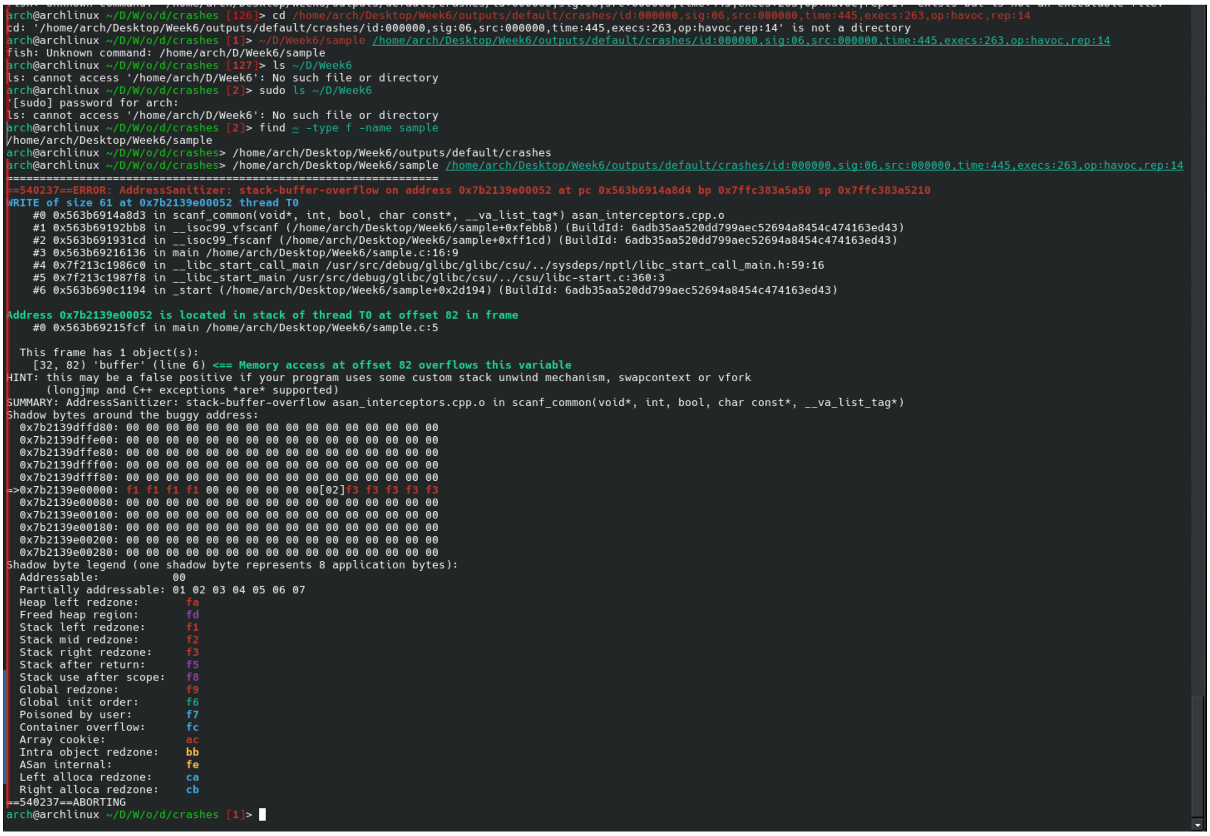
```
        american fuzzy lop ++4.35c {default} (./sample) [explore]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 9 min, 40 sec      │  cycles done : 1069  │
│   last new find : none yet (odd, check syntax!)     │ corpus count : 1     │
│last saved crash : 0 days, 0 hrs, 9 min, 40 sec      │saved crashes : 1     │
│ last saved hang : none seen yet                     │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 0.3207 (0.0%)      │    map density : 20.00% / 20.00%    │
│  runs timed out : 0 (0.00%)          │ count coverage : 209.00 bits/tuple  │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc                  │ favored items : 1 (100.00%)         │
│ stage execs : 10/100 (10.00%)        │  new edges on : 1 (100.00%)         │
│ total execs : 320k                   │ total crashes : 5064 (1 saved)      │
│  exec speed : 518.6/sec              │  total tmouts : 16 (0 saved)        │
├─ fuzzing strategy yields ────────────┴───────────┬─ item geometry ─────────┤
│   bit flips : 0/0, 0/0, 0/0                      │    levels : 1           │
│  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 : 0           │
│  dictionary : 0/0, 0/0, 0/0, 0/0                 │  imported : 0           │
│havoc/splice : 1/320k, 0/0                        │ stability : 100.00%     │
│py/custom/rq : unused, unused, unused, unused     ├─────────────────────────┘
│    trim/eff : 20.00%/1, n/a                      │          [cpu000:225%]
└─ strategy: explore ────────── state: finished... ┘
```

2. Give all the required commands for completing this task.
   - # Install AFL++ sudo pacman -Sy afl++
   - # Compile with AFL instrumentation + ASan AFL_USE_ASAN=1 afl-cc -o sample sample.c
   - # Create input corpus mkdir inputs echo "test" > inputs/seed.txt
   - # Create output directory mkdir outputs
   - # Fix core_pattern so AFL++ can detect crashes echo core | sudo tee /proc/sys/kernel/core_pattern
   - # Run AFL++ afl-fuzz -i inputs -o outputs -- ./sample @@
   - # Reproduce a crash /home/arch/Desktop/Week6/sample \ /home/arch/Desktop/Week6/outputs/default/crashes/id:000000,sig:06,src:000000,time:445,execs:263,op:havoc,rep:14
3. Copy-paste the AddressSanitizer output. Does it identify the line of code in the program, which causes the problem? What other information does it tell?

```
arch@archlinux ~/D/W/o/d/crashes> /home/arch/Desktop/Week6/sample
/home/arch/Desktop/Week6/outputs/default/crashes/id:000000,sig:06,src:000000,time:445,ex
ecs

:263,op:havoc,rep:14

=================================================================
==540237==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7b2139e00052 at pc 0x563b6914a8d4 bp 0x7ffc383a5a50 sp 0x7ffc383a5210

WRITE of size 61 at 0x7b2139e00052 thread T0

  #0 0x563b6914a8d3 in scanf_common(void*, int, bool, char const*, __va_list_tag*) asan_interceptors.cpp.o

  #1 0x563b69192bb8 in __isoc99_vfscanf (/home/arch/Desktop/Week6/sample+0xfebb8) (BuildId: 6adb35aa520dd799aec52694a8454c474163ed43)

  #2 0x563b691931cd in __isoc99_fscanf (/home/arch/Desktop/Week6/sample+0xff1cd) (BuildId: 6adb35aa520dd799aec52694a8454c474163ed43)

  #3 0x563b69216136 in main /home/arch/Desktop/Week6/sample.c:16:9

  #4 0x7f213c1986c0 in __libc_start_call_main /usr/src/debug/glibc/glibc/csu/../sysdeps/nptl/libc_start_call_main.h:59:16

  #5 0x7f213c1987f8 in __libc_start_main /usr/src/debug/glibc/glibc/csu/../csu/libc-start.c:360:3

  #6 0x563b690c1194 in _start (/home/arch/Desktop/Week6/sample+0x2d194) (BuildId: 6adb35aa520dd799aec52694a8454c474163ed43)


Address 0x7b2139e00052 is located in stack of thread T0 at offset 82 in frame

  #0 0x563b69215fcf in main /home/arch/Desktop/Week6/sample.c:5


 This frame has 1 object(s):

  [32, 82) 'buffer' (line 6) <== Memory access at offset 82 overflows this variable

HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork

    (longjmp and C++ exceptions *are* supported)

SUMMARY: AddressSanitizer: stack-buffer-overflow asan_interceptors.cpp.o in scanf_common(void*, int, bool, char const*, __va_list_tag*)

Shadow bytes around the buggy address:

 0x7b2139dffd80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

 0x7b2139dffe00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

 0x7b2139dffe80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
  0x7b2139dfff00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139dfff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x7b2139e00000: f1 f1 f1 f1 00 00 00 00 00 00[02]f3 f3 f3 f3 f3
  0x7b2139e00080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
```

==540237==ABORTING



```
arch@archlinux ~/D/W/o/d/crashes [126]> cd /home/arch/Desktop/Week6/outputs/default/crashes/id:000000,sig:06,src:000000,time:445,execs:263,op:havoc,rep:14
cd: '/home/arch/Desktop/Week6/outputs/default/crashes/id:000000,sig:06,src:000000,time:445,execs:263,op:havoc,rep:14' is not a directory
arch@archlinux ~/D/W/o/d/crashes [1]> ~/D/Week6/sample /home/arch/Desktop/Week6/outputs/default/crashes/id:000000,sig:06,src:000000,time:445,execs:263,op:havoc,rep:14
fish: Unknown command: /home/arch/D/Week6/sample
arch@archlinux ~/D/W/o/d/crashes [127]> ls ~/D/Week6
ls: cannot access '/home/arch/D/Week6': No such file or directory
arch@archlinux ~/D/W/o/d/crashes [2]> sudo ls ~/D/Week6
'[sudo] password for arch:
ls: cannot access '/home/arch/D/Week6': No such file or directory
arch@archlinux ~/D/W/o/d/crashes [2]> find ~ -type f -name sample
/home/arch/Desktop/Week6/sample
arch@archlinux ~/D/W/o/d/crashes> /home/arch/Desktop/Week6/outputs/default/crashes
arch@archlinux ~/D/W/o/d/crashes> /home/arch/Desktop/Week6/sample /home/arch/Desktop/Week6/outputs/default/crashes/id:000000,sig:06,src:000000,time:445,execs:263,op:havoc,rep:14
=================================================================
==540237==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7b2139e00052 at pc 0x563b6914a8d4 bp 0x7ffc383a5a50 sp 0x7ffc383a5210
WRITE of size 61 at 0x7b2139e00052 thread T0
    #0 0x563b6914a8d3 in scanf_common(void*, int, bool, char const*, __va_list_tag*) asan_interceptors.cpp.o
    #1 0x563b69192bb8 in __isoc99_vfscanf (/home/arch/Desktop/Week6/sample+0xfebb8) (BuildId: 6adb35aa520dd799aec52694a8454c474163ed43)
    #2 0x563b691931cd in __isoc99_fscanf (/home/arch/Desktop/Week6/sample+0xff1cd) (BuildId: 6adb35aa520dd799aec52694a8454c474163ed43)
    #3 0x563b69216136 in main /home/arch/Desktop/Week6/sample.c:16:9
    #4 0x7f213c1986c0 in __libc_start_call_main /usr/src/debug/glibc/glibc/csu/../sysdeps/nptl/libc_start_call_main.h:59:16
    #5 0x7f213c1987f8 in __libc_start_main /usr/src/debug/glibc/glibc/csu/../csu/libc-start.c:360:3
    #6 0x563b690c1194 in _start (/home/arch/Desktop/Week6/sample+0x2d194) (BuildId: 6adb35aa520dd799aec52694a8454c474163ed43)

Address 0x7b2139e00052 is located in stack of thread T0 at offset 82 in frame
    #0 0x563b69215fcf in main /home/arch/Desktop/Week6/sample.c:5

  This frame has 1 object(s):
    [32, 82) 'buffer' (line 6) <== Memory access at offset 82 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow asan_interceptors.cpp.o in scanf_common(void*, int, bool, char const*, __va_list_tag*)
Shadow bytes around the buggy address:
  0x7b2139dffd80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139dffe00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139dffe80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139dfff00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139dfff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x7b2139e00000: f1 f1 f1 f1 00 00 00 00 00 00[02]f3 f3 f3 f3 f3
  0x7b2139e00080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00180: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x7b2139e00280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
==540237==ABORTING
arch@archlinux ~/D/W/o/d/crashes [1]> █
```

Does AddressSanitizer identify the line of code?

Yes. AddressSanitizer clearly identifies the exact line in the program that caused the crash:

main /home/arch/Desktop/Week6/sample.c:16:9

This tells you the overflow happens at line 16 of **sample.c**

What else does AddressSanitizer tell you?

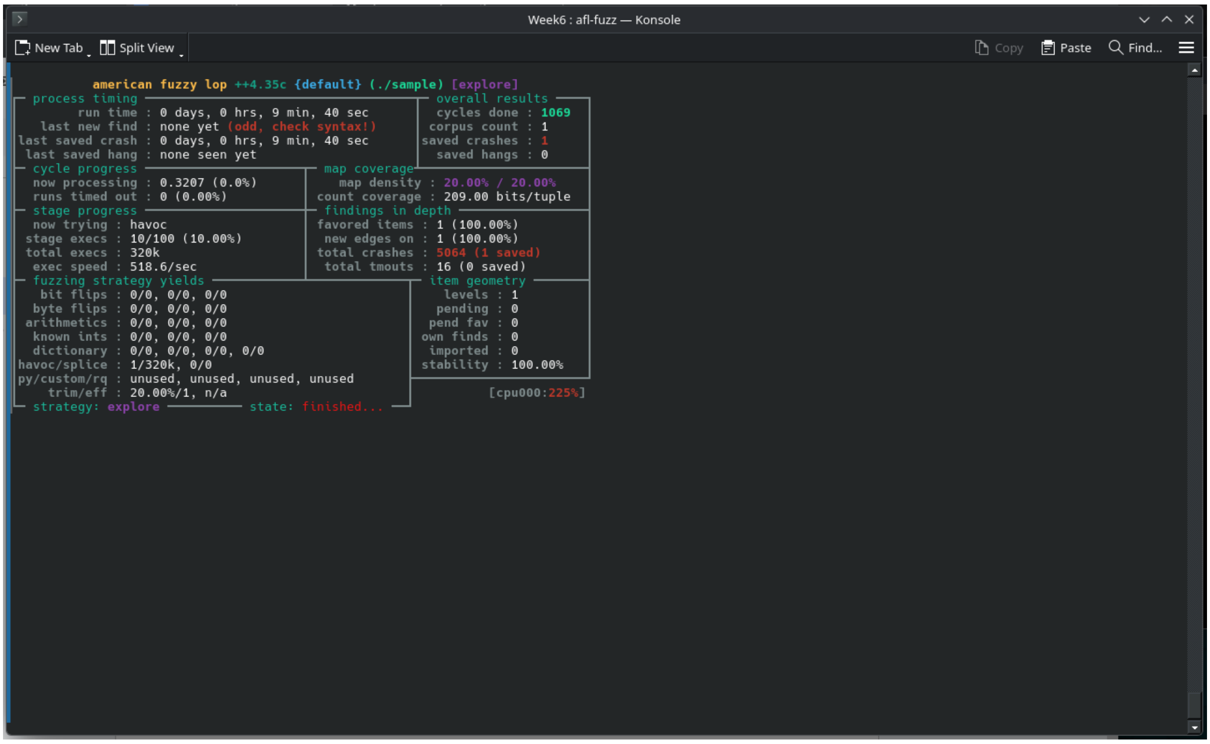
- The type of bug: stack-buffer-overflow
- The invalid memory access size: WRITE of size 61
- The exact memory address where the overflow occurred
- A stack trace showing the call chain
- The location of the vulnerable variable: **[32, 82) 'buffer' (line 6)**
- A shadow memory map showing redzones and poisoned areas
- The signal that terminated the program (SIGABRT)

4. How many crashes did the fuzzer find? How many were unique?

From my AFL++ screenshot:

- Total crashes: 5064
- Unique crashes saved: 1

AFL++ found 5064 crashes, of which 1 was unique.



```
        american fuzzy lop ++4.35c {default} (./sample) [explore]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 9 min, 40 sec      │  cycles done : 1069  │
│   last new find : none yet (odd, check syntax!)     │ corpus count : 1     │
│last saved crash : 0 days, 0 hrs, 9 min, 40 sec      │saved crashes : 1     │
│ last saved hang : none seen yet                     │  saved hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────────┤
│  now processing : 0.3207 (0.0%)      │    map density : 20.00% / 20.00%    │
│  runs timed out : 0 (0.00%)          │ count coverage : 209.00 bits/tuple  │
├─ stage progress ─────────────────────┼─ findings in depth ─────────────────┤
│  now trying : havoc                  │ favored items : 1 (100.00%)         │
│ stage execs : 10/100 (10.00%)        │  new edges on : 1 (100.00%)         │
│ total execs : 320k                   │ total crashes : 5064 (1 saved)      │
│  exec speed : 518.6/sec              │  total tmouts : 16 (0 saved)        │
├─ fuzzing strategy yields ────────────┴─────────────┬─ item geometry ───────┤
│   bit flips : 0/0, 0/0, 0/0                        │    levels : 1         │
│  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 : 0         │
│  dictionary : 0/0, 0/0, 0/0, 0/0                   │  imported : 0         │
│havoc/splice : 1/320k, 0/0                          │ stability : 100.00%   │
│py/custom/rq : unused, unused, unused, unused       ├───────────────────────┘
│    trim/eff : 20.00%/1, n/a                        │          [cpu000:225%]
└─ strategy: explore ────────── state: finished... ──┘
```

5. How many cycles did the program do? What does "cycles" mean?

Cycles done: 1069

- A "cycle" means AFL++ has processed the entire queue of interesting inputs once.
- Each cycle applies new mutation strategies to try to discover new execution paths.

6. When you should stop the fuzzer? Explain.

- You should stop the fuzzer when it stops finding new paths or new unique crashes for a long time.
- When "last new path" and "last uniq crash" stop updating, the fuzzer has reached coverage saturation.

7. What fuzzing strategy is being used at the time of the screenshot?

- Strategy: explore
- Stage: havoc

At the time of the screenshot, AFL++ was using the explore strategy and was in the havoc stage.