

# 0x27 Automated Testing Exercise Session



**EPFL**

Recap

# Fuzzing

- Automatic and dynamic software testing technique
- Key concept: fastest way to test a program is to run it
  - Runs on bare metal => “efficient”
- Analogy: obstacle course
  - Programs are riddled with control-flow decisions
  - Only one path can be taken at a time
  - `path[-1]` may depend on `path[0:-1]`



- In this Lab, we use AFL++<sup>[1]</sup>, current State-of-the-Art Greybox Fuzzer

[1] <https://github.com/AFLplusplus/AFLplusplus>

# Environment Setup

- Download the Dockerfile  
<https://github.com/kdsjZh/COM402-Testing-Lab/blob/master/docker/Dockerfile>
- `docker build -t com402/testing:2024 /path/to/dockerfile`
- `docker run -it com402/testing:2024 bash`

# Ex1. Blackbox Fuzzing

# Program Under Test

```
unsigned int size;
if (read(fd, &size, 4) != 4) {
    perror("Failed to read size");
    close(fd);
    return 1;
}

// Vulnerable buffer: fixed-size but could overflow based on file contents
char buffer[MAX_BUFFER_SIZE];

// Read 'size' bytes into the buffer,
if (read(fd, buffer, size) < 0) {
    perror("Failed to read file content");
    close(fd);
    return 1;
}

// Null-terminate the buffer
buffer[MAX_BUFFER_SIZE - 1] = '\0';
```

# Program Under Test

```
unsigned int size;
if (read(fd, &size, 4) != 4) {
    perror("Failed to read size");
    close(fd);
    return 1;
}

// Vulnerable buffer: fixed-size but could overflow based on file contents
char buffer[MAX_BUFFER_SIZE];

// Read 'size' bytes into the buffer,
if (read(fd, buffer, size) < 0) {
    perror("Failed to read file content");
    close(fd);
    return 1;
}

// Null-terminate the buffer
buffer[MAX_BUFFER_SIZE - 1] = '\0';
```

if size > MAX\_BUFFER\_SIZE,  
stack buffer overflows!

# Ex1. Blackbox Fuzzing

```
[AFL++ 225c9e640908] /AFLplusplus # cd /COM402-Testing-Lab/demo/ex1
```

```
demo/ex1 # make clean && make
```

```
demo/ex1 # cat run.sh
```

## Ex1. Blackbox Fuzzing

```
demo/ex1 # ./run.sh
```

```

american fuzzy lop ++4.22a {} (./ex1) [explore]
┌────────── process timing ───────────┐ ┌────────── overall results ───────────┐
│ run time : 0 days, 0 hrs, 0 min, 3 sec │ │ cycles done : 26 │
│ last new find : n/a (non-instrumented mode) │ │ corpus count : 1 │
│ last saved crash : 0 days, 0 hrs, 0 min, 0 sec │ │ saved crashes : 59 │
│ last saved hang : none seen yet │ │ saved hangs : 0 │
└────────── cycle progress ───────────┘ └────────── map coverage ───────────┘
│ now processing : 0*79 (0.0%) │ │ map density : 0.00% / 0.00% │
│ runs timed out : 0 (0.00%) │ │ count coverage : 0.00 bits/tuple │
└────────── stage progress ───────────┘ └────────── findings in depth ───────────┘
│ now trying : havoc │ │ favored items : 0 (0.00%) │
│ stage execs : 57/100 (57.00%) │ │ new edges on : 0 (0.00%) │
│ total execs : 7864 │ │ total crashes : 59 (59 saved) │
│ exec speed : 2526/sec │ │ total tmouts : 0 (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 : n/a │
│ havoc/splice : 53/7800, 0/0 │ │ stability : n/a │
│ py/custom/rq : unused, unused, unused, unused │ │ │
│ trim/eff : n/a, n/a │ │ │
└────────── strategy: explore ───────────┘ └────────── state: in progress ───────────┘

```

# Ex1. Blackbox Fuzzing

For Simple Programs, Blackbox fuzzers works pretty well.

But what if code becomes complex?

## Ex2. Coverage Feedback

# Program Under Test

```
unsigned char *size_bytes = (unsigned char *)&size;  
unsigned char magic_bytes[2] = { 0xef, 0xbe};
```

```
// Check and print messages for each matching byte  
if (size_bytes[0] == magic_bytes[0]) {  
    printf("Byte 1 is match!\n");  
} else {  
    printf("Byte 1 does not match.\n");  
    return 1;  
}
```

```
if (size_bytes[1] == magic_bytes[1]) {  
    printf("Byte 2 is match!\n");  
} else {  
    printf("Byte 2 does not match.\n");  
    return 1;  
}
```

# Program Under Test

```
unsigned char *size_bytes = (unsigned char *)&size;  
unsigned char magic_bytes[2] = { 0xef, 0xbe};
```

```
// Check and print messages for each matching byte
```

```
if (size_bytes[0] == magic_bytes[0]) {  
    printf("Byte 1 is match!\n");
```

```
} else {  
    printf("Byte 1 does not match.\n");  
    return 1;  
}
```

```
if (size_bytes[1] == magic_bytes[1]) {  
    printf("Byte 2 is match!\n");
```

```
} else {  
    printf("Byte 2 does not match.\n");  
    return 1;  
}
```

First two bytes of size have to match magic bytes, else program exit

## Ex2. Coverage-Guided Fuzzing

```
[AFL++ 225c9e640908] /AFLplusplus # cd /COM402-Testing-Lab/demo/ex2
```

```
demo/ex2 # make clean && make
```

```
demo/ex2 # $AFL_PATH/afl-fuzz -i in/ -o out/ -n -- ./ex2 @@
```

Try Blackbox fuzzing for 5 minutes.

Does Blackbox fuzzing works? Why/Why not?

## Ex2. Coverage-Guided Fuzzing

```
demo/ex2 # make clean && CC=afl-cc make
```

```
demo/ex2 # cat run.sh
```

We build with afl-cc, a clang wrapper that compile program and **instrument coverage**

## Ex2. Converge-Guided Fuzzing

```
american fuzzy lop ++4.22a { } (./ex2) [explore]
├─ process timing ─────────── overall results ───────────
│   run time : 0 days, 0 hrs, 0 min, 30 sec          cycles done : 244
│   last new find : n/a (non-instrumented mode)      corpus count : 1
└─ last saved crash : none seen yet                  saved crashes : 0
    last saved hang : none seen yet                   saved hangs : 0
├─ cycle progress ─────────── map coverage ───────────
│   now processing : 0*733 (0.0%)                    map density : 0.00% / 0.00%
│   runs timed out : 0 (0.00%)                       count coverage : 0.00 bits/tuple
└─ stage progress ─────────── findings in depth ───────────
    now trying : havoc                                favored items : 0 (0.00%)
    stage execs : 94/100 (94.00%)                     new edges on : 0 (0.00%)
    total execs : 73.3k                               total crashes : 0 (0 saved)
    exec speed : 2392/sec                             total tmouts : 0 (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 : n/a
│   havoc/splice : 0/73.2k, 0/0                       stability : n/a
└─ py/custom/rq : unused, unused, unused, unused
    trim/eff : n/a, n/a
└─ strategy: explore ─────────── state: in progress ─── ^C
```

## Without Coverage

| american fuzzy lop ++4.22a [default] (./ex2) [explore] |                                   |
|--------------------------------------------------------|-----------------------------------|
| process timing                                         | overall results                   |
| run time : 0 days, 0 hrs, 0 min, 14 sec                | cycles done : 200                 |
| last new find : 0 days, 0 hrs, 0 min, 14 sec           | corpus count : 4                  |
| last saved crash : 0 days, 0 hrs, 0 min, 1 sec         | saved crashes : 1                 |
| last saved hang : none seen yet                        | saved hangs : 0                   |
| cycle progress                                         | map coverage                      |
| now processing : 2.388 (50.0%)                         | map density : 14.29% / 35.71%     |
| runs timed out : 0 (0.00%)                             | count coverage : 77.80 bits/tuple |
| stage progress                                         | findings in depth                 |
| now trying : havoc                                     | favored items : 4 (100.00%)       |
| stage execs : 98/100 (98.00%)                          | new edges on : 4 (100.00%)        |
| total execs : 120k                                     | total crashes : 1 (1 saved)       |
| exec speed : 7991/sec                                  | total tmouts : 0 (0 saved)        |
| fuzzing strategy yields                                | item geometry                     |
| bit flips : 0/0, 0/0, 0/0                              | levels : 3                        |
| 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 : 3                     |
| dictionary : 0/0, 0/0, 0/0, 0/0                        | imported : 0                      |
| havoc/splice : 4/120k, 0/0                             | stability : 100.00%               |
| py/custom/rq : unused, unused, unused, unused          |                                   |
| trim/eff : n/a, n/a                                    |                                   |
| strategy: explore                                      | [cpu000: 10%]                     |
| state: started :-)                                     | ^C                                |

## With Coverage

## Ex2. Coverage-Guided Fuzzing

If the magic bytes are validated byte-by-byte, coverage feedback works well,

But what if program compares 4 bytes at once?

## Ex3. Magic Bytes

# Program Under Test

```
// Check and print messages for each matching byte
if (size == MAGIC_BYTES) {
    printf("Magic Bytes match!\n");
} else {
    printf("Magic Bytes not match.\n");
    return 1;
}
```

# Program Under Test

```
// Check and print messages for each matching byte
if (size == MAGIC_BYTES) {
    printf("Magic Bytes match!\n");
} else {
    printf("Magic Bytes not match.\n");
    return 1;
}
```

Check 4 Bytes at once

## Ex3. Magic Bytes

```
demo/ex2 # cd /COM402-Testing-Lab/demo/ex3/
```

```
demo/ex3 # make clean && CC=afl-cc make
```

```
demo/ex3 # $AFL_PATH/afl-fuzz -i in/ -o out/ -- ./ex3 @@
```

Try fuzzing for 5 minutes

Does coverage guided fuzzing still work for magic bytes comparison?

## Ex3. Magic Bytes

CmpLog<sup>[1]</sup> is a technique that extract the compared value from register and fill it back to input.

It's enabled in AFL++ if we compile program with `AFL_LLVM_CMPLOG=1`

[1] Aschermann, Cornelius, et al. "REDQUEEN: Fuzzing with Input-to-State Correspondence." *NDSS*. Vol. 19. 2019.

## Ex3. Magic Bytes

```
demo/ex3 # CC=afl-cc make ex3 && mv ex3 ex3.afl
```

```
demo/ex3 # CC=afl-cc AFL_LLVM_CMPLOG=1 make ex3 && mv ex3 ex3.cmplog
```

```
demo/ex3 # $AFL_PATH/afl-fuzz -i in/ -o out/ -c ./ex3.cmplog -l 3 -- ./ex3.afl @@
```

We build one binary with cmplog instrumentation and run AFL++ with cmplog mutator enabled

# Ex3. Magic Bytes

```
american fuzzy lop ++4.22a {default} (./ex3.afl) [explore]
process timing
  run time : 0 days, 0 hrs, 0 min, 40 sec
  last new find : 0 days, 0 hrs, 0 min, 40 sec
  last saved crash : none seen yet
  last saved hang : none seen yet
cycle progress
  now processing : 0.1685 (0.0%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 72/100 (72.00%)
  total execs : 332k
  exec speed : 8180/sec
fuzzing strategy yields
  bit flips : 0/0, 0/0, 0/0
  byte flips : 0/0, 0/0, 0/0
  arithmetics : 0/0, 0/0, 0/0
  known ints : 0/0, 0/0, 0/0
  dictionary : 0/0, 0/0, 0/0, 0/0
  havoc/splice : 1/332k, 0/0
  py/custom/rq : unused, unused, unused, unused
  trim/eff : 22.22%/2, n/a
overall results
  cycles done : 830
  corpus count : 2
  saved crashes : 0
  saved hangs : 0
map coverage
  map density : 15.38% / 23.08%
  count coverage : 129.00 bits/tuple
findings in depth
  favored items : 2 (100.00%)
  new edges on : 2 (100.00%)
  total crashes : 0 (0 saved)
  total tmouts : 0 (0 saved)
item geometry
  levels : 2
  pending : 0
  pend fav : 0
  own finds : 1
  imported : 0
  stability : 100.00%
[cpu000: 10%]
strategy: explore state: started :-)^C
```

Without Cmplog

```
american fuzzy lop ++4.22a {default} (./ex3.afl) [explore]
process timing
  run time : 0 days, 0 hrs, 0 min, 4 sec
  last new find : 0 days, 0 hrs, 0 min, 4 sec
  last saved crash : 0 days, 0 hrs, 0 min, 1 sec
  last saved hang : none seen yet
cycle progress
  now processing : 2.74 (66.7%)
  runs timed out : 0 (0.00%)
stage progress
  now trying : havoc
  stage execs : 55/100 (55.00%)
  total execs : 35.6k
  exec speed : 7184/sec
fuzzing strategy yields
  bit flips : 0/0, 0/0, 0/0
  byte flips : 0/0, 0/0, 0/0
  arithmetics : 0/0, 0/0, 0/0
  known ints : 0/0, 0/0, 0/0
  dictionary : 0/0, 0/0, 0/0, 0/0
  havoc/splice : 1/17.4k, 1/18.0k
  py/custom/rq : unused, unused, 0/22, 1/49
  trim/eff : disabled, n/a
overall results
  cycles done : 34
  corpus count : 3
  saved crashes : 1
  saved hangs : 0
map coverage
  map density : 15.38% / 30.77%
  count coverage : 97.00 bits/tuple
findings in depth
  favored items : 3 (100.00%)
  new edges on : 3 (100.00%)
  total crashes : 2 (1 saved)
  total tmouts : 0 (0 saved)
item geometry
  levels : 2
  pending : 0
  pend fav : 0
  own finds : 2
  imported : 0
  stability : 100.00%
[cpu000: 10%]
strategy: explore state: started :-)^C
```

With Cmplog

## Ex3. Magic Bytes

All our assumption is based on that vulnerability will crash the program, e.g.,  
overwriting the return address in the stack

But what if the vulnerability does not crash the program?

## Ex4. Sanitizer

# Program Under Test

```
char *buffer = (char *)malloc(MAX_BUFFER_SIZE);
```

Now buffer is on heap, overflow does not directly crash the program

## Ex4. Sanitizer

```
demo/ex3 # cd /COM402-Testing-Lab/demo/ex4
```

```
demo/ex4 # make clean && CC=afl-cc make
```

```
demo/ex4 # $AFL_PATH/afl-fuzz -i in/ -o out/ -- ./ex4 @@
```

Fuzz for 5 minutes, any finding?

Is the bug being triggered?

## Ex4. Sanitizer

AddressSanitizer<sup>[1]</sup> is a technique that stops the program execution if the memory safety is violated, i.e. don't wait until program cannot execute, stop as soon as its wrong.

It's integrated both in gcc and clang, enabled in AFL++ if we compile with `AFL_USE_ASAN=1`

Tip: set ``ulimit -c unlimited``

[1] Serebryany, Konstantin, et al. "{AddressSanitizer}: A fast address sanity checker." *2012 USENIX annual technical conference (USENIX ATC 12)*. 2012.

## Ex4. Sanitizer

```
american fuzzy lop ++4.22a {default} (./ex4) [explore]
├─ process timing ──────────────────────────────────────────── overall results ────────────────────────────────────────────
│   run time      : 0 days, 0 hrs, 0 min, 30 sec                cycles done : 255
│   last new find  : 0 days, 0 hrs, 0 min, 29 sec              corpus count : 4
│   last saved crash : none seen yet                          saved crashes : 0
│   last saved hang  : none seen yet                          saved hangs   : 0
├─ cycle progress ─────────────────────────────────────────── map coverage ───────────────────────────────────────────
│   now processing : 2.369 (50.0%)                             map density  : 14.29% / 35.71%
│   runs timed out : 0 (0.00%)                                count coverage : 77.80 bits/tuple
├─ stage progress ─────────────────────────────────────────── findings in depth ───────────────────────────────────────────
│   now trying     : havoc                                       favored items  : 4 (100.00%)
│   stage execs    : 2/100 (2.00%)                            new edges on  : 4 (100.00%)
│   total execs    : 249k                                        total crashes : 0 (0 saved)
│   exec speed     : 8169/sec                                    total tmouts  : 0 (0 saved)
├─ fuzzing strategy yields ─────────────────────────────────── item geometry ───────────────────────────────────────────
│   bit flips      : 0/0, 0/0, 0/0                             levels       : 3
│   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    : 3
│   dictionary     : 0/0, 0/0, 0/0, 0/0                       imported     : 0
│   havoc/splice   : 3/249k, 0/0                               stability    : 100.00%
│   py/custom/rq   : unused, unused, unused, unused
│   trim/eff       : 23.53%/3, n/a
└─ strategy: explore ─────────────────────────────────── state: started :- ) ─────────────────────────────────── ^C
```

## Without ASan

```

american fuzzy lop ++4.22a {default} (./ex4) [explore]
├─ process timing ────────────┬────────── overall results ───────────
│   run time : 0 days, 0 hrs, 0 min, 4 sec │ cycles done : 15
│   last new find : 0 days, 0 hrs, 0 min, 1 sec │ corpus count : 4
│ last saved crash : 0 days, 0 hrs, 0 min, 0 sec │ saved crashes : 1
│ last saved hang : none seen yet │ saved hangs : 0
├─ cycle progress ───────────┬──────── map coverage ───────────
│ now processing : 3.7 (75.0%) │ map density : 14.29% / 35.71%
│ runs timed out : 0 (0.00%) │ count coverage : 77.80 bits/tuple
├─ stage progress ───────────┬──────── findings in depth ───────────
│ now trying : havoc │ favored items : 4 (100.00%)
│ stage execs : 37/7400 (9.25%) │ new edges on : 4 (100.00%)
│ total execs : 14.8k │ total crashes : 2 (1 saved)
│ exec speed : 2988/sec │ total tmouts : 0 (0 saved)
├─ fuzzing strategy yields ───┬──────── item geometry ───────────
│ bit flips : 0/0, 0/0, 0/0 │ levels : 3
│ 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 : 3
│ dictionary : 0/0, 0/0, 0/0, 0/0 │ imported : 0
│ havoc/splice : 4/14.7k, 0/0 │ stability : 100.00%
│ py/custom/rq : unused, unused, unused, unused
│ trim/eff : 77.94%/16, n/a
└─ strategy: explore ───────── state: started :- ) ^C

```

## With ASan

# Ex5. Crash Analysis

## Ex5. Crash Analysis

Crashes are not equal to bugs!

We still need to manually analyze the crashes to verify if its exploitable.

Take ex4 as example, run # `./ex4 /path/to/crash`

# Ex5. Crash Analysis

```
=====
==25920==ERROR: AddressSanitizer: heap-buffer-overflow in address 0x60b000000a4 at pc 0x5a34619237af bp 0x7ffe220d2dd0
sp 0x7ffe220d25a8
WRITE of size 108 at 0x60b000000a4 thread T0
[Detaching after fork from child process 25923]
#0 0x5a34619237ae in __interceptor_read (/COM402-Testing-Lab/demo/ex4/ex4+0x3c7ae) (BuildId: 8d20f5e6139030f47c7563
31ffb6dbadee64791e)
#1 0x5a34619db064 in main /COM402-Testing-Lab/demo/ex4/ex4.c:52:9
#2 0x72c35ad94d8f (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildId: 490fef8403240c91833978d494d39e537409b92e)
#3 0x72c35ad94e3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x29e3f) (BuildId: 490fef8403240c91833978d4
94d39e537409b92e)
#4 0x5a34619063c4 in _start (/COM402-Testing-Lab/demo/ex4/ex4+0x1f3c4) (BuildId: 8d20f5e6139030f47c756331ffb6dbadee
64791e)

0x60b000000a4 is located 0 bytes after 100-byte region [0x60b000000040,0x60b0000000a4)
allocated by thread T0 here:
#0 0x5a34619a01ee in __interceptor_malloc (/COM402-Testing-Lab/demo/ex4/ex4+0xb91ee) (BuildId: 8d20f5e6139030f47c75
6331ffb6dbadee64791e)
#1 0x5a34619dafed in main /COM402-Testing-Lab/demo/ex4/ex4.c:49:28
#2 0x72c35ad94d8f (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildId: 490fef8403240c91833978d494d39e537409b92e)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/COM402-Testing-Lab/demo/ex4/ex4+0x3c7ae) (BuildId: 8d20f5e6139030f47c
756331ffb6dbadee64791e) in __interceptor_read
```

# Ex5. Crash Analysis

```
=====
==25920==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60b0000000a4
    sp 0x7ffe220d25a8
WRITE of size 108 at 0x60b0000000a4 thread T0
[Detaching after fork from child process 25923]
    #0 0x5a34619237ae in __interceptor_read (/COM402-Testing-Lab/demo/ex4/ex4.0
31ffb6dbadee64791e)
    #1 0x5a34619db064 in main /COM402-Testing-Lab/demo/ex4/ex4.c:52:9
    #2 0x72c35ad94d8f (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildID: 1
    #3 0x72c35ad94e3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+
94d39e537409b92e)
    #4 0x5a34619063c4 in _start (/COM402-Testing-Lab/demo/ex4/ex4+0x1f3c4)
64791e)

0x60b0000000a4 is located 0 bytes after 100-byte region [0x60b000000040,
allocated by thread T0 here:
    #0 0x5a34619a01ee in __interceptor_malloc (/COM402-Testing-Lab/demo/ex4
6331ffb6dbadee64791e)
    #1 0x5a34619dafed in main /COM402-Testing-Lab/demo/ex4/ex4.c:49:28
    #2 0x72c35ad94d8f (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildID: 1

SUMMARY: AddressSanitizer: heap-buffer-overflow (/COM402-Testing-Lab/demo/ex4
756331ffb6dbadee64791e) in __interceptor_read
```

```
// Read 'size' bytes into the buffer,
if (read(fd, buffer, size) < 0) {
    perror("Failed to read file
content");
    close(fd);
    return 1;
}
```

# Ex5. Crash Analysis

```
(gdb) set args /path/to/crash
```

```
(gdb) b ex4.c:52
```

```
Breakpoint 1 at 0xf3fff: file ex4.c, line 52.
```

```
(gdb) r
```

```
Starting program: /COM402-Testing-Lab/demo/ex4/ex4
```

```
out/default/crashes/id\:000000\,sig\:06\,src\:000003\,time\:6471\,execs\:19701\,op\:havoc\,rep  
\:16
```

```
...
```

```
Byte 1 is match!
```

```
Byte 2 is match!
```

```
Breakpoint 1, main (argc=2, argv=0x7ffe2567e6e8) at ex4.c:52
```

```
52          if (read(fd, buffer, size) < 0) {
```

```
(gdb) display size
```

```
1: size = 2763964143
```

Size larger than buffer, Overflow!

## Ex5. Crash Analysis

```
=====
==25920==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60b0000000a4 at pc 0x7f56220d25a8
sp_0x7ffe220d25a8
WRITE of size 108 at 0x60b0000000a4 thread T0
[Detaching after fork from child process 25923]
#0 0x5a34619237ae in __interceptor_read (/COM402-Testing-Lab/demo/ex4/ex4+0x3c7ae) (BuildId: 8d2064791e)
#1 0x5a34619db064 in main /COM402-Testing-Lab/demo/ex4/ex4.c:52:9
#2 0x72c35ad94d8f (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildId: 490fef8403240c)
#3 0x72c35ad94e3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x29e3f) (BuildId: 490fef8403240c)
#4 0x5a34619063c4 in _start (/COM402-Testing-Lab/demo/ex4/ex4+0x1f3c4) (BuildId: 8d2064791e)
0x60b0000000a4 is located 0 bytes after 100-byte region [0x60b000000040,0x60b0000000a4)
allocated by thread T0 here:
#0 0x5a34619a01ee in __interceptor_malloc (/COM402-Testing-Lab/demo/ex4/ex4+0xb91ee) (BuildId: 8d2064791e)
#1 0x5a34619dafed in main /COM402-Testing-Lab/demo/ex4/ex4.c:49:28
#2 0x72c35ad94d8f (/lib/x86_64-linux-gnu/libc.so.6+0x29d8f) (BuildId: 490fef8403240c)
SUMMARY: AddressSanitizer: heap-buffer-overflow (/COM402-Testing-Lab/demo/ex4/ex4+0x3c7ae) in __interceptor_read
Stack frame (frames: 1)
#0 0x5a34619237ae in __interceptor_read (/COM402-Testing-Lab/demo/ex4/ex4+0x3c7ae) (BuildId: 8d2064791e)
```

## Ex5. Crash Analysis

Why the size is 2763964143, but the actual write size is 108?

```
demo/ex4 # ls -lh
```

```
out/default/crashes/id\:000000\,sig\:06\,src\:000003\,time\:6471\,execs\:19701\,op\:havoc\,rep\:16
```

```
-rw----- 1 root root 112 Nov  2 14:09
```

```
out/default/crashes/id:000000,sig:06,src:000003,time:6471,execs:19701,op:havoc,rep:16
```