0x27 Automated Testing Exercise Session

hexhive

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++[1], 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

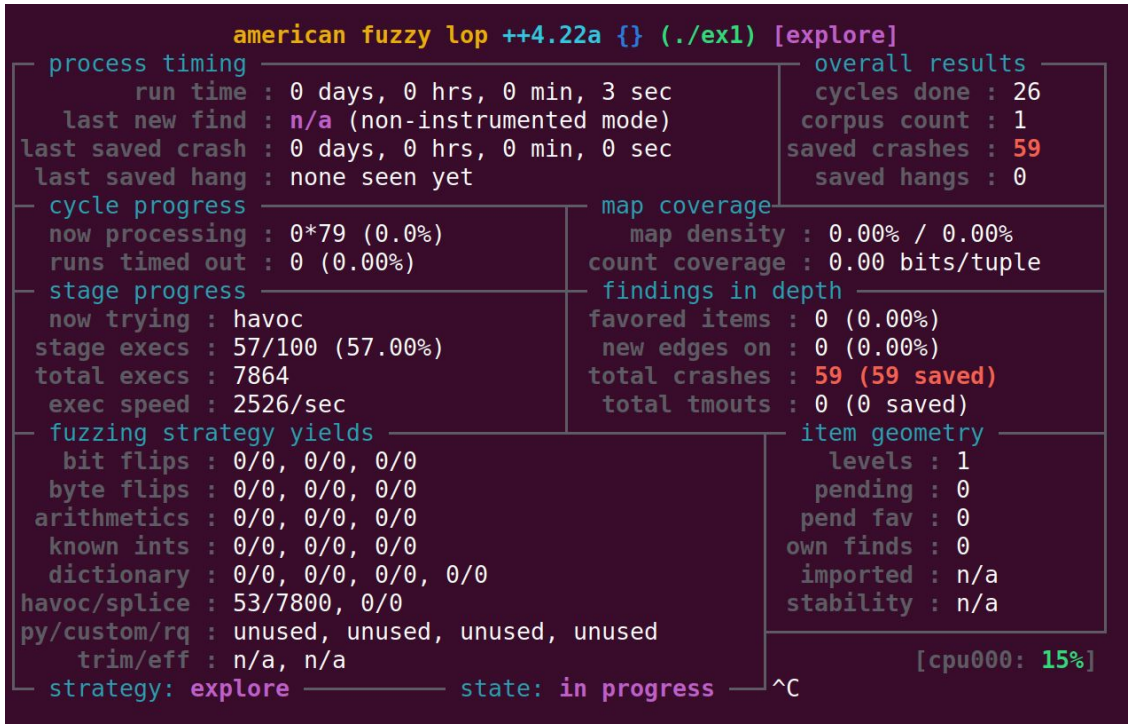
[AFL++ 225c9e640908] /AFLplusplus # `cd /COM402-Testing-Lab/demo/ex1`

demo/ex1 # `make clean && make`

demo/ex1 # `cat run.sh`

Ex1. Blackbox Fuzzing

demo/ex1 # `./run.sh`



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?

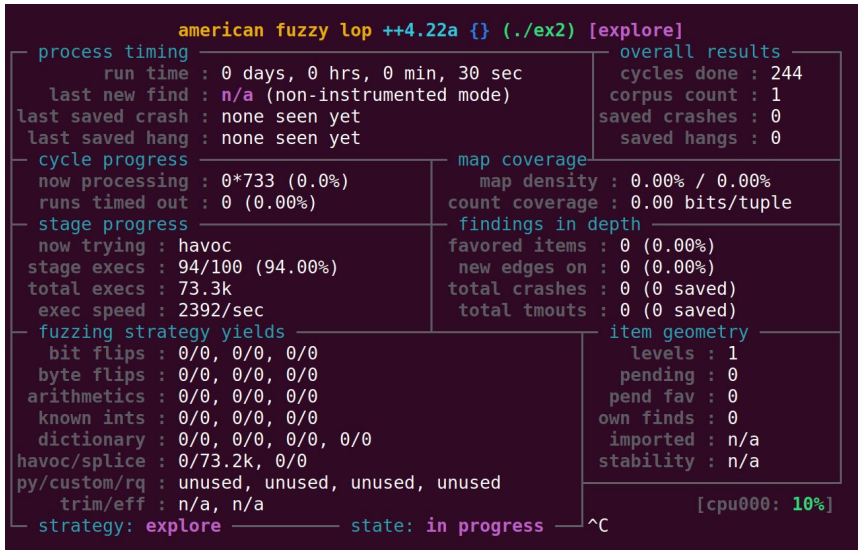
Ex2. Coverage-Guided Fuzzing

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

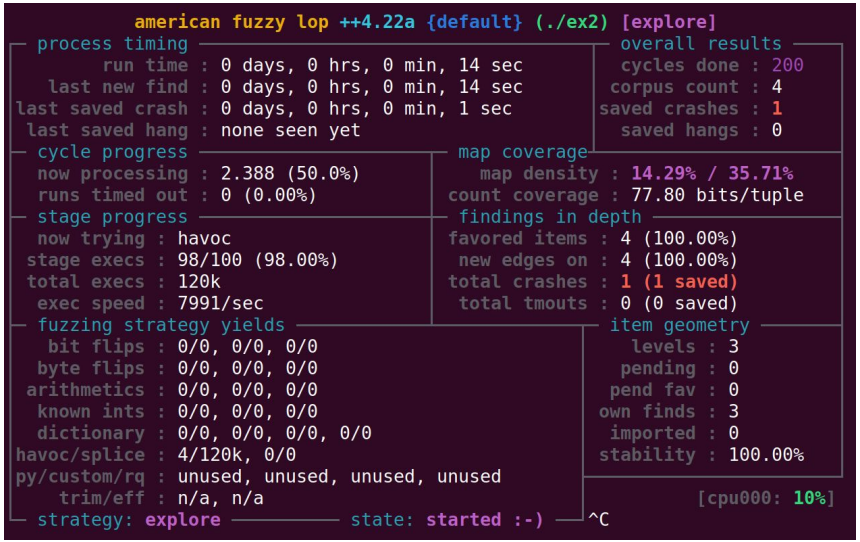
```
demo/ex2 # cat run.sh
```

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

Ex2. Converge-Guided Fuzzing



Without Coverage



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[1] 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

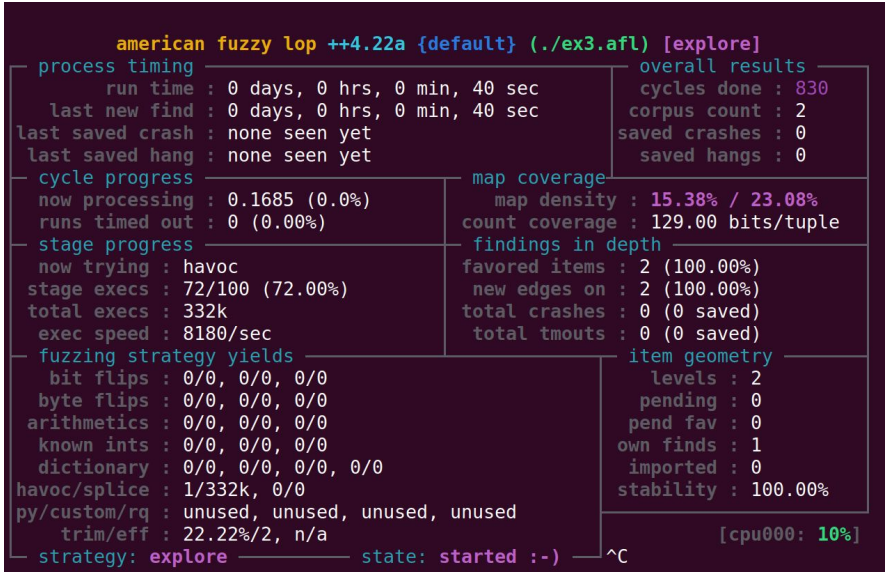
[1] Aschermann, Cornelius, et al. "REDQUEEN: Fuzzing with Input-to-State Correspondence." *NDSS*. Vol. 19. 2019.

Ex3. Magic Bytes

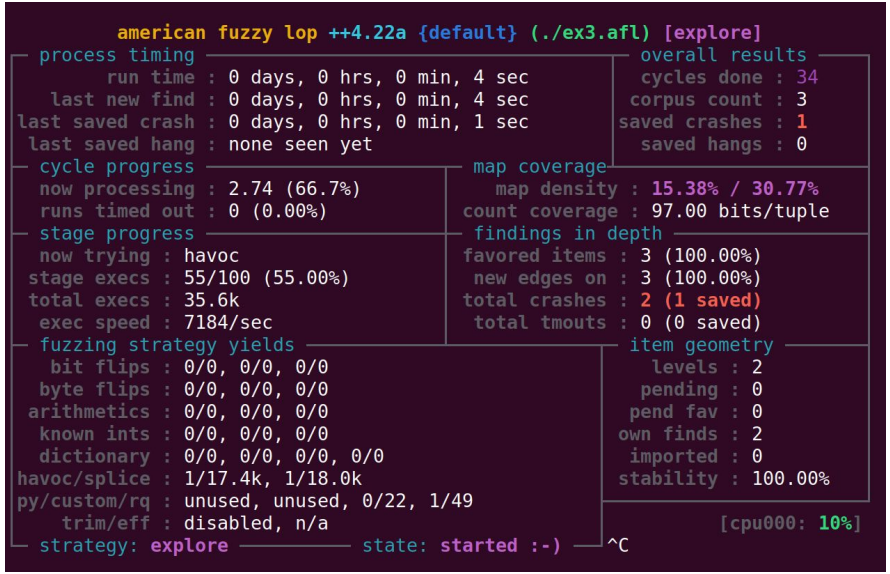
```
demo/ex3 # CC=afl-cc make ex3 && mv ex3 ex3.afl

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



Without Cmplog



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

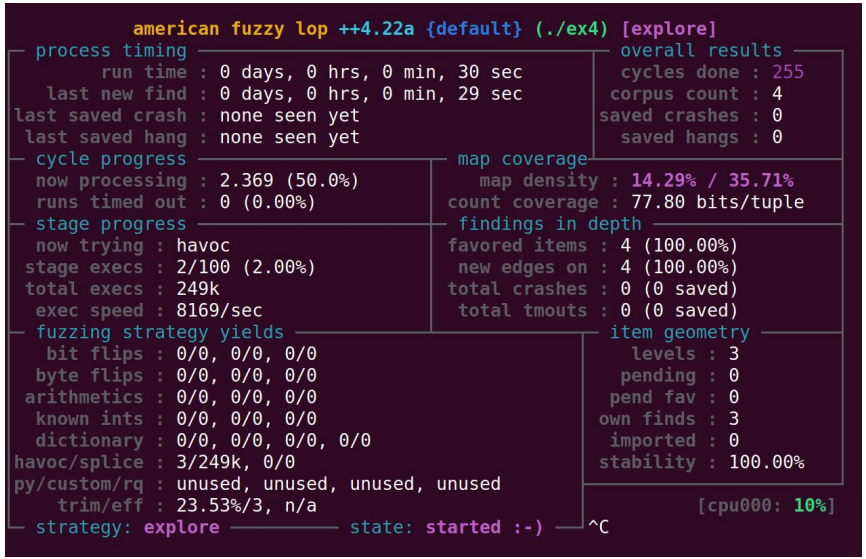
AddressSanitizer[1] is a technique that stops the program execution if the memory safety is violated, i.e. don’t wait until program cannot execute, stop as soon as its wrong.

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

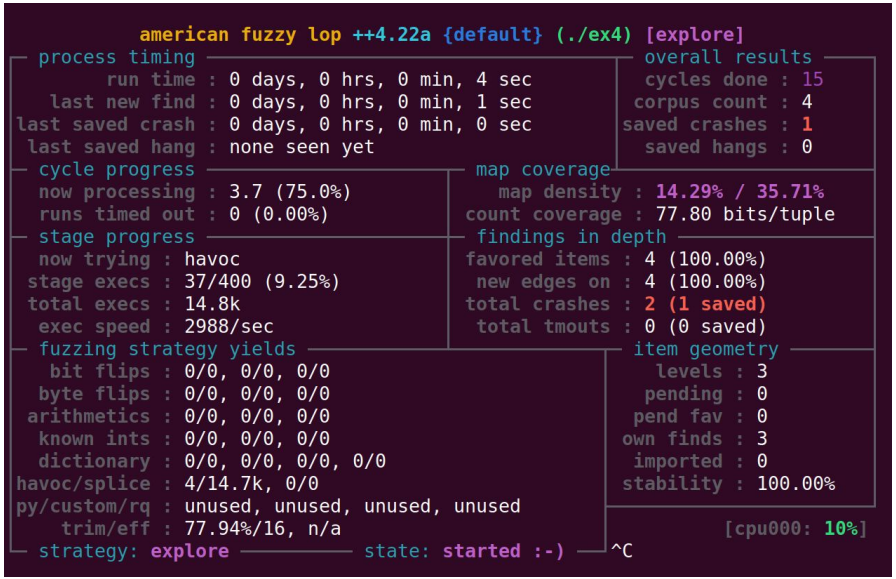
Tip: set `ulimit -c unlimited`

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

Ex4. Sanitizer



Without ASan



With ASan

Ex5. Crash Analysis

Ex5. Crash Analysis

Crashes are not equal to bugs!

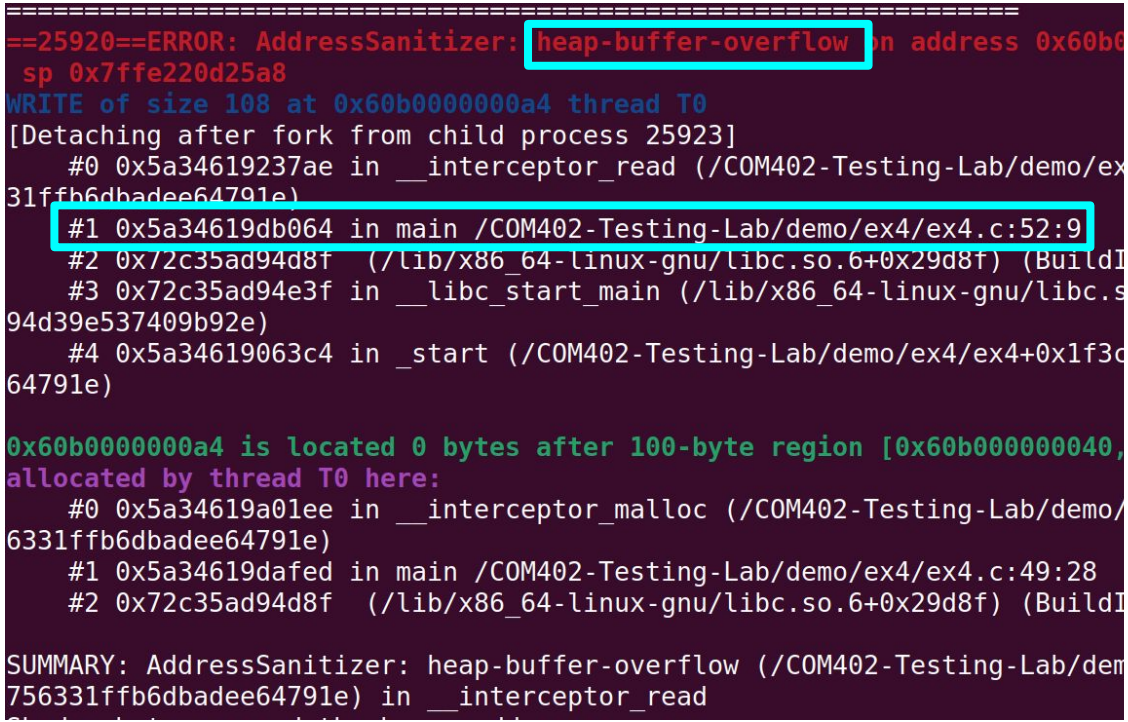
We still need to manually analyze the crashes to verify if its exploitable.

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

Ex5. Crash Analysis

```
=================================================================
==25920==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60b0000000a4 at pc 0x5a34619237af bp 0x7ffe220d2dd0
 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: 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)

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: 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




```
    // 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 0x
 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) (B
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: 490fef8403240c
    #3 0x72c35ad94e3f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x29e3f) (Bui
94d39e537409b92e)
    #4 0x5a34619063c4 in _start (/COM402-Testing-Lab/demo/ex4/ex4+0x1f3c4) (BuildId: 8d2c
64791e)

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)
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: 490fef8403240c

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

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