Graham Dungan
March 28, 2025
UIN: 332001764

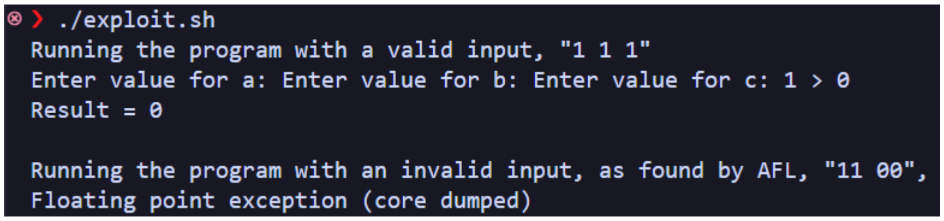
CSCE 413: Software Security
Class 28: Fuzzing

Demonstration Quickstart

This demonstration `exploit.sh` shows a vulnerable program `vuln` running with both valid and invalid input, as supplied by the output of AFL, causing a crash.
To run this demo,

1. Enter the `Class28` directory.
   `cd Class28`
2. *(Optional)* If the script does not have execution permission, add such permissions.
   `chmod +x ./exploit.sh`
3. Run the `exploit.sh` script.
   `./exploit.sh`

What follows is a screenshot of the output of `exploit.sh`.



```
❯ ./exploit.sh
  Running the program with a valid input, "1 1 1"
  Enter value for a: Enter value for b: Enter value for c: 1 > 0
  Result = 0

  Running the program with an invalid input, as found by AFL, "11 00",
  Floating point exception (core dumped)
```

1 Create a vulnerable application (to any vulnerability)

For this assignment, I have created a file named `vuln.c`.

```c
 1 #include <stdio.h>
 2 int *comparison_function(int a, int b, int c) {
 3     int result;
 4     // Compute the result
 5     if (a < b) {
 6         if (b > 10) {
 7             result = c / b - a;
 8         } else {
 9             result = c * b - a;
10         }
11     } else {
12         if (a >= 10) {
13             result = c / b - b;  // Intentional bug
14         } else {
15             result = c * a - b;
16         }
17     }
18     // Compare the result
19     if (a < result) {
20         printf("%d < %d\n", a, result);
21     } else {
22         printf("%d > %d\n", a, result);
23     }
24     printf("Result = %d\n", result);
25     return 0;
26 }
27 int main() {
28     int a, b, c;
29     printf("Enter value for a: ");
30     scanf("%d", &a);
31     printf("Enter value for b: ");
32     scanf("%d", &b);
33     printf("Enter value for c: ");
34     scanf("%d", &c);
35     comparison_function(a, b, c);
36     return 0;
37 }
```

Ultimately, this program does nothing of importance. The program requests three values, `a`, `b`, `c`, and does various comparison and arithmetical operations on them. The many branching paths were made to simulate the logical flow of a more complex program. This file contains a simple divide-by-zero vulnerability due to a ”typo” made on line 13, which does not check the value of `b` before it divides `c`. A more formal definition would call this a ”denial-of-service” vulnerability, due to an incorrect input being intentionally used to cause the program to crash. We will aim to find a string that causes this error using AFL.

2 Run AFL on it and demonstrate how AFL identifies the vulnerability

We will first compile the vulnerable program using AFL instrumentation,

```
afl-gcc vuln.c -o vuln_afl
```

This allows AFL to include instructions that will make tracing the program easier.

Once the program has been compiled, we can create a seed. This seed will be kept in a directory named `in`, of which it will include the inputs `10, 1, 2`. This seed is not an input that will crash the program, but is adjacent to it (when `a > b`, `a > 10`, and `b = 0`).

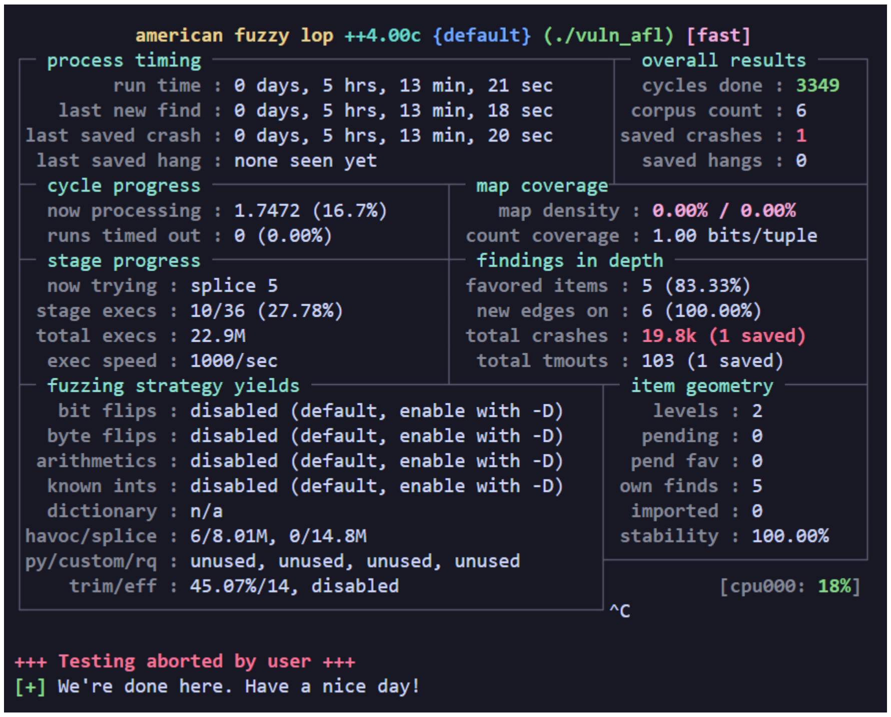
Finally, AFL can be run with the following command,

```
afl-fuzz -i in -o out -- ./vuln_afl
```

This runs the fuzzing portion of the AFL suite. Breaking down this command,

- `-i in` - Provides the seek file.
- `-o out` - Outputs the inputs that cause hangs and crashes.
- `./vuln_afl` - Uses the previously compiled `vuln_afl` binary.

Once run, a GUI showing the search will be displayed,



```
            american fuzzy lop ++4.00c {default} (./vuln_afl) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 5 hrs, 13 min, 21 sec    │  cycles done : 3349│
│   last new find : 0 days, 5 hrs, 13 min, 18 sec    │ corpus count : 6   │
│last saved crash : 0 days, 5 hrs, 13 min, 20 sec    │saved crashes : 1   │
│ last saved hang : none seen yet                    │  saved hangs : 0   │
├─ cycle progress ─────────────────────┬─ map coverage┴──────────────────┤
│  now processing : 1.7472 (16.7%)     │    map density : 0.00% / 0.00%  │
│  runs timed out : 0 (0.00%)          │ count coverage : 1.00 bits/tuple│
├─ stage progress ─────────────────────┼─ findings in depth ─────────────┤
│  now trying : splice 5               │ favored items : 5 (83.33%)      │
│ stage execs : 10/36 (27.78%)         │  new edges on : 6 (100.00%)     │
│ total execs : 22.9M                  │ total crashes : 19.8k (1 saved) │
│  exec speed : 1000/sec               │  total tmouts : 103 (1 saved)   │
├─ fuzzing strategy yields ────────────┴─────────────┬─ item geometry ───┤
│   bit flips : disabled (default, enable with -D)   │    levels : 2     │
│  byte flips : disabled (default, enable with -D)   │   pending : 0     │
│ arithmetics : disabled (default, enable with -D)   │  pend fav : 0     │
│  known ints : disabled (default, enable with -D)   │ own finds : 5     │
│  dictionary : n/a                                  │  imported : 0     │
│havoc/splice : 6/8.01M, 0/14.8M                     │ stability : 100.00%│
│py/custom/rq : unused, unused, unused, unused       ├───────────────────┘
│    trim/eff : 45.07%/14, disabled                  │        [cpu000: 18%]
└────────────────────────────────────────────────────┘^C

+++ Testing aborted by user +++
[+] We're done here. Have a nice day!
```

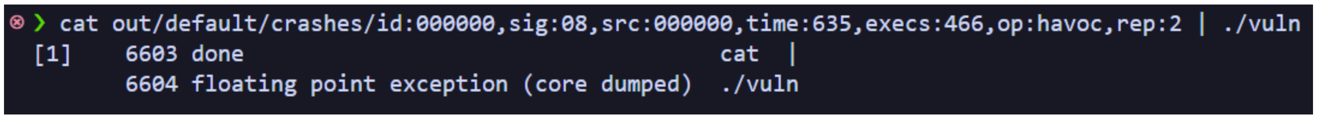
After the program's execution, all inputs that cause crashes are stored in `out/default/crashes/`. In this instance, there was only one set of inputs that caused a crash, which was stored in,

```
id:000000,sig:08,src:000000,time:635,execs:466,op:havoc,rep:2
```

Viewing the contents of this file, the input that caused a crash was "11 00". We can now use this to cause the program to crash purposefully.

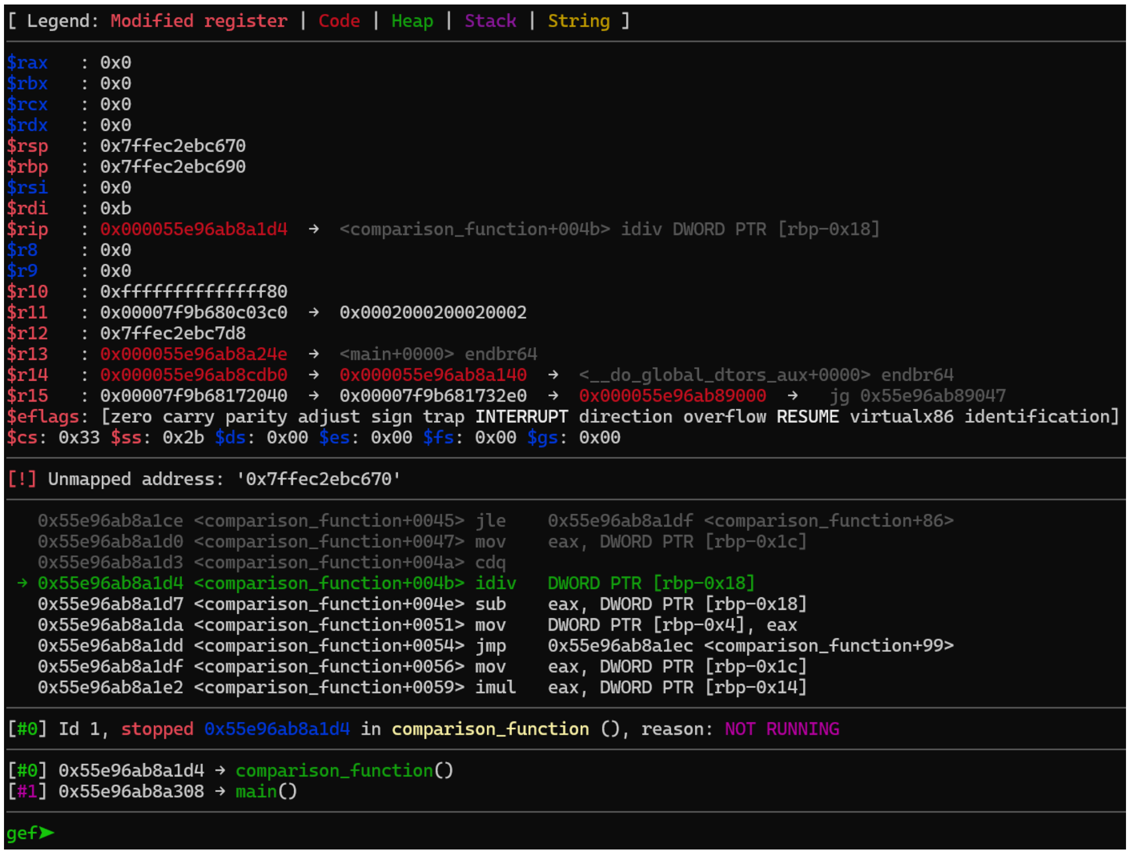
3 Show how to exploit the identified vulnerability

While not immediately beneficial, crashing the program can reveal sensitive information about it's memory and logic. We can purposefully cause the program to crash by supplying the previously found input,



```
⊗ ❯ cat out/default/crashes/id:000000,sig:08,src:000000,time:635,execs:466,op:havoc,rep:2 | ./vuln
  [1]    6603 done                                cat  |
         6604 floating point exception (core dumped)  ./vuln
```

In enabling core dumps, we can use gdb to inspect the memory of the program before it crashed.



```
[ Legend: Modified register | Code | Heap | Stack | String ]

$rax   : 0x0
$rbx   : 0x0
$rcx   : 0x0
$rdx   : 0x0
$rsp   : 0x7ffec2ebc670
$rbp   : 0x7ffec2ebc690
$rsi   : 0x0
$rdi   : 0xb
$rip   : 0x000055e96ab8a1d4  →  <comparison_function+004b> idiv DWORD PTR [rbp-0x18]
$r8    : 0x0
$r9    : 0x0
$r10   : 0xffffffffffffff80
$r11   : 0x00007f9b680c03c0  →  0x0002000200020002
$r12   : 0x7ffec2ebc7d8
$r13   : 0x000055e96ab8a24e  →  <main+0000> endbr64
$r14   : 0x000055e96ab8cdb0  →  0x000055e96ab8a140  →  <__do_global_dtors_aux+0000> endbr64
$r15   : 0x00007f9b68172040  →  0x00007f9b681732e0  →  0x000055e96ab89000  →  jg 0x55e96ab89047
$eflags: [zero carry parity adjust sign trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00

[!] Unmapped address: '0x7ffec2ebc670'

   0x55e96ab8a1ce <comparison_function+0045> jle    0x55e96ab8a1df <comparison_function+86>
   0x55e96ab8a1d0 <comparison_function+0047> mov    eax, DWORD PTR [rbp-0x1c]
   0x55e96ab8a1d3 <comparison_function+004a> cdq
 → 0x55e96ab8a1d4 <comparison_function+004b> idiv   DWORD PTR [rbp-0x18]
   0x55e96ab8a1d7 <comparison_function+004e> sub    eax, DWORD PTR [rbp-0x18]
   0x55e96ab8a1da <comparison_function+0051> mov    DWORD PTR [rbp-0x4], eax
   0x55e96ab8a1dd <comparison_function+0054> jmp    0x55e96ab8a1ec <comparison_function+99>
   0x55e96ab8a1df <comparison_function+0056> mov    eax, DWORD PTR [rbp-0x1c]
   0x55e96ab8a1e2 <comparison_function+0059> imul   eax, DWORD PTR [rbp-0x14]

[#0] Id 1, stopped 0x55e96ab8a1d4 in comparison_function (), reason: NOT RUNNING

[#0] 0x55e96ab8a1d4 → comparison_function()
[#1] 0x55e96ab8a308 → main()

gef➤
```

As seen here, we have exploited the vulnerability in order to gain sensitive information about the workings of the program.