Lab 5

Documentation

December 6, 2022

By Kevin Yu 20203451

And Raatik Sharma 20120770

Problem Statement

The purpose of this lab was to look at an exploit that deals with a buffer overflow. Code would be written to compromise the server and get all environment variables where one of them would be a password hash. With the password hash retrieved from the server, a match would be found through trial and error in the terminal by checking different combinations of a single word followed by a single number and seeing which one of those guesses would make the same hash to the hash received from the server. The words/numbers would be in some way related to the person in question via social media.

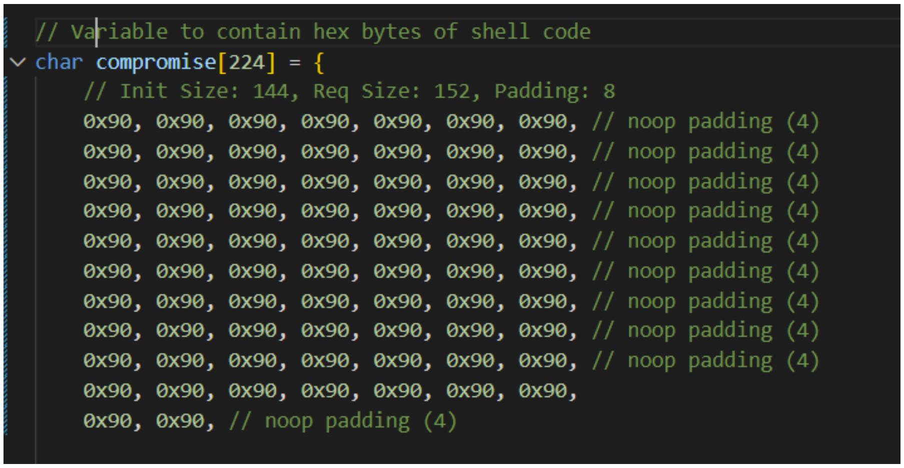
How the Attack Works

The attack works by overflowing the character buffer “compromise” and overwriting the return address to point the instruction pointer to the beginning of our malicious code to be executed. In the buffer, a sequence of hex numbers are stored which correspond to the malicious code. The return address was determined by calculating the difference between the stack pointer and the size of the buffer allowing the instruction pointer to start at the beginning of the malicious code. The length of the string used to compromise the server was 224 bytes long and the location of the return address on the stack was 0x7fffffffe168.

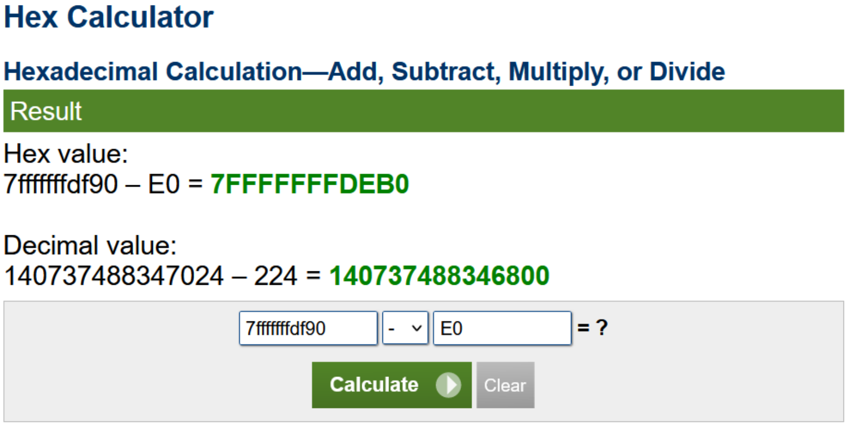
Location of the Return Address

```
#0  0x000055555555696a in ?? ()
(gdb) info registers
rax            0x7fffffffdfa0      140737488347040
rbx            0x555555556bd0      93824992242640
rcx            0x0                 0
rdx            0x7fffffffe06d      140737488347245
rsi            0x7fffffffdf80      140737488347008
rdi            0x7fffffffdfa0      140737488347040
rbp            0x5a595857504f4e4d  0x5a595857504f4e4d
rsp            0x7fffffffdf88      0x7fffffffdf88
r8             0xfefefefefefefeff  -72340172838076673
r9             0xfefeffe077777777  -72339204449273993
r10            0x7ffff7fc2810      140737353885712
r11            0x7fffffffdf81      140737488347009
r12            0x555555556460      93824992240736
r13            0x7fffffffe1a0      140737488347552
r14            0x0                 0
--Type <RET> for more, q to quit, c to continue without pagi
```

Size of the buffer



```
// Variable to contain hex bytes of shell code
char compromise[224] = {
    // Init Size: 144, Req Size: 152, Padding: 8
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // noop padding (4)
    0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90,
    0x90, 0x90, // noop padding (4)
```



Hex Calculator

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:
7fffffffdf90 – E0 = **7FFFFFFFDEB0**

Decimal value:
140737488347024 – 224 = **140737488346800**

7fffffffdf90 - E0 = ?

NASM, Selfcomp.c and Client.c Source Code

Please refer to Lab 5 in GitLabs to see the source code.