■Q3

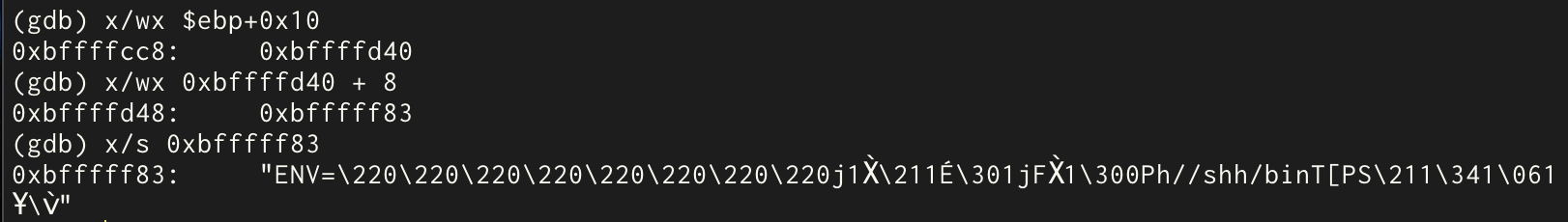
The program has an Off by one vulnerability. At line 9, it checks i<=64 which allows an attacker to write to buf[64] which is the least significant byte of saved frame pointer.

for (i = 0; i < n && i <= 64; ++i)

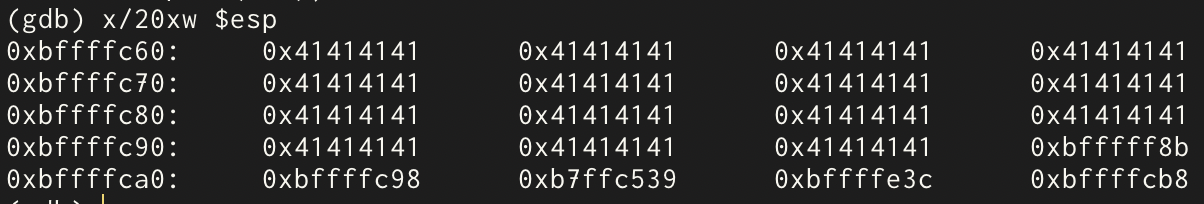
buf[i] = input[i] ^ (1u << 5);

An attacker can forge the sfp to point into the buffer and by taking advantage of the function epilogues, an attacker can make the program execute the shellcode.

First, I used the environmental variable ENV to store the shellcode. Using gdb, I found that ENV is placed in env[2] at 0xbfffff8b.



In order to place 0xbfffff8b into the buffer, I need to take xor with (1u << 5) for every byte. If I flip the 6th bit of each byte, it will give a string “\xab\xdf\xdf\x9f”. Now I can place this string from buf[60] - buf[63], then the sfp needs to point to buf[56] which is ebp - 8 = 0xbffffca0 - 8 = 0xbffffc98. Since the current sfp is 0xbffffcac, I just need to modify the least significant byte which we can modify because sfp is placed right above buf which means buf[64] is the least significant byte of sfp. In order to store 0xbffffc98 in sfp, I took 0x98 xor (1u << 5) = b8, and stored in buf[64]. I can fill in from buf[0] to buf[59] with any junk. This will give a string "a" \* 60 + "\xab\xdf\xdf\x9f" + "\xb8", and it will look as follows:



Here is the description about how it will lead to the execution of the shellcode.

Function epilogues: mov %ebp, %esp - (1), pop %ebp - (2), pop %eip - (3)

In the function epilogues for the invoke function, after (1) is executed, the ebp and the esp point to the sfp. After (2), the ebp points to 0xbffffc98 and the esp points to the rip. After (3), the esp points to the top of dispatch’s stack frame. Next, we have the function epilogues for the dispatch function. After (1), the esp and the ebp point to 0xbffffc98. After (2), the esp is incremented by 4 bytes and point to 0xbffffc9c which is buf[60]. After (3), the eip is overwritten with 0xbfffff8b which is the address where the shellcode is stored. The program will execute the shellcode next.

■Q4

At line 8, the program has a string format vulnerability. Because the program allows an attacker to pass any arguments to the printf function, even if we have the stack canary, an attacker can overwrite the return address without overwriting the stack canary, which allows the program to execute the shellcode.

First, using the following string, I printed out the contents of the stack above printf and I identified that string[0] is at the 7th argument to the printf:

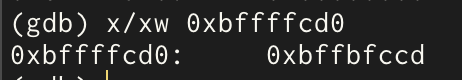
print "aaaa" + " %08x %08x %08x %08x %08x %08x %08x"

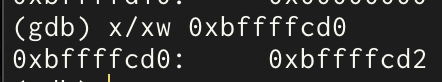


Since the address where the rip is stored is 0xbffffd0c, upper 2 bytes are stored at 0xbffffd0e. In order to overwrite the upper 2 bytes of the rip of oracle(), I need to store 0xbffffd0e into string[0] - string[3] and move the print()’s internal pointer by 6 times using %x, then use %hn to overwrite the upper half of rip. Next, in order to overwrite the lower 2 bytes of rip, I need to first increment the number of characters printed using %<some number>x, and use %hn to write to the target. This means I need to store the address of rip (0xbffffd0c) into string[8] - string[11] because %15571x will increment the printf()’s internal pointer. We can fill out string[4] - string[7] with “%x%x” because that would reduce the total number of characters needed inside the string buffer. Now, the only thing I need to figure out is where the shellcode is stored so that I can overwrite the rip with the address of the shellcode. With the following string, I figured out that the shellcode starts at 0xbffffcd2:

"\x0e\xfd\xff\xbf" + "%x%x" + "\x0c\xfd\xff\xbf"+ "%x" \* 3 + "%49130x" + "hhn"+ "%10000x" + "hhn" + "aaaaaaaaaaaaaaaaaaaaaaaaa"



In order to store 0xbfff which is 49151 in decimal into the upper 2 bytes, since I already have 25 characters, I need to pad out 49151 - 25 = 49126 more characters. And in order to store fcd2 (64722), I need 64722 - 49154 = 15570 more characters. I tried writing it to 0xbffffcd0 so that I can see if I’m writing the correct value. And it turns out it was a little off:

After some trial and error, I got the value 0xbffffcd2:

And by changing the target address to rip, and adding the shellcode at the end of the string, I got the final solution:

"\x0e\xfd\xff\xbf" + "%x%x" + "\x0c\xfd\xff\xbf"+ "%x" \* 3 + "%49130x" + "%hn"+ "%15571x" + "%hn" + shellcode