M1522.000800 System Programming
Fall 2018

System Programming
Buflab Report

Lee Young Jae
2016-11988

1. <lab> Buffer Lab

<short project description>
The goal of the buffer lab is to learn von Neumann architecture. In von Neumann architecture, the .text section and stack shares the machine code. In this architecture, we can execute the code in the stack area.

2. Implementation

<describe your design, implementation, way to solve the lab>

The buffer lab consists of five phases: Smoke, Sparkler, Firecracker, Dynamite and Nitroglycerin. To solve the problems, I use disassembler generated by 'objdump -d filename' and gdb debugger. In gdb debugger, the command 'break function_name' makes the process pauses in the function_name, and 'info registers' command shows current register information. Thanks to disassemler, I could find the address of function and the location of variables (especially for Sparkler phase). Thanks to gdb debugger, I could find the address of register information, which is essential to solve Dynamite.

2.1 Candle

<explain what you did to solve this phase in detail>
The goal of Candle phase is to call Smoke function. To call a Smoke function, I manipulated the return address of getbuf function. To manipulate the return address of getbuf function, I put the string in Get function with long string.
In the Get function, it receives 40 chars. However, if the user(Me) put the string more then 40 characters, then Get function touch other data section. More precisely, the stack frame of getbuf function is like below:

Memory Address	Value	Description
%ebx	%eax	val will be stored here
	Return address	Where we will go next
%ebp(0x55683340)	Old %ebp	
(0x5568333c)	Old %ebx	
	...	
	...	
%eax(0x55683310)		Get function starts from here
%esp(0x556832f8)	%eax	The value of %esp points where Get function starts

Each cell in the table contains 4 bytes(4 characters). Thus, to manipulate return address, we have to insert 56 characters in Get function. What we need is just call Smoke function(0x08048be8). We

don’t care anything else. The only thing to do is put the string with last four characters are 08048be8 with little endian. Therefore, we have to put the string
00 times 52 + e8 8b 04 08. In 00 parts, it does matter what it is.

2.2 Sparkler

<explain what you did to solve this phase in detail>

The goal of Sparkler phase it so call fizz function with appropriate arguments. The arguments are stored just before return address. To solve Sparkler phase, go to fizz function(0x08048c12) first, then insert appropriate arguments. In the fizz function, 0x55683350 stores ret function, and 0x55683354, 0x55683358 stores two arguments. I inserted val for 0 and val2 for (~val << 8) & cookie = 2fffe400.Therefore, my string in this phase is:
00 times 52 + 12 8c 04 08 00 00 00 00 00 00 00 00 e4 ff 2f

2.3 Firecracker

<explain what you did to solve this phase in detail>

The goal of Firecracker phase is to call bang function and change global variable. The only way to change global variable value is by machine code instruction. That is, we have to inject machine code by buffer overflow. We use von Neumann architecture to inject code. In the previous phases, we put address of the function to return to another function. However, in this phases, we put the address of the stack. If the PC goes to the stack memory, it executes the machine codes stored in the stack. In the stack, we use machine code changed by assembler to modify global variable and then return to the original bang function. By disassembling the code of bufbomb file, we can know that that the address of global variable is 0x0804d120, and the address of bang function is 0x08048c81. To change this 0x0804d120 address and go back to bang, we have to insert:

```
movl $0x0000040e, 0x0804d120
push $0x08048c81
ret
```

It is translated to filename.o by command ‘gcc -m32 -c filename.s’. We can read the real machine code of filename.o by command ‘objdump -d filename.o’. The machine code result without comments are:

```
c7 05 20 d1 04 08 0e
04 00 00
68 81 8c 04 08
c3
```

Now, only we have to do is inserting zero padding and return address (stack memory = 0x55683310). Therefore, my string in this phase is:
c7 05 20 d1 04 08 0e
04 00 00
68 81 8c 04 08
c3
+ 00 times 36
+ 10 33 68 55

2.4 Dynamite

<explain what you did to solve this phase in detail>

The goal of Dynamite phase is to change val and return to next line of the getbuf function. By disassembling the code, we can know that val is stored in 0x55683364, and the next line of the getbuf function is 08048e65. Therefore, we have to inject

```
movl $0x2fffe42e, 0x55683364
push $0x0804e865
ret
```

However, it is not sufficient. When we perform with this code, we encounter segmentation fault. It is because %ebp address. By zero padding, we make the value of %ebp to zero. To execute the function correctly, we have to make the value of %esp is equal to the value of %ebp. In the previous phases, there is an instruction ‘movl %esp, %ebp’ to make that the two values are same. However, in this phase, there is no such instruction since we insert out code into the middle of the function. Hence, we have to manually modify %ebp value. Thus, we insert

```
movl $0x55683370, %ebp
```

into the assembly code. Just as I did in Firecracker mode, we compile it and disassemble it. We get toe machine code

```
bd 70 33 68 55
c7 05 64 33 68 55 2e
e4 ff 2f
68 65 8e 04 08
c3
```

Therefore, my string in this phase is
bd 70 33 68 55
c7 05 64 33 68 55 2e
e4 ff 2f
68 65 8e 04 08
c3
+ 00 times 31
+ 10 68 33 55

2.5 Nitroglycerin

<explain what you did to solve this phase in detail>

The goal of Nitroglycerin phase is to change val value and return to the next line of the getbufn function. Unlike the previous phase, we can modify the value of val by just changing %eax value. In the previous phase, we have to find the address of the val since %eax does not store the value of val in getbuf. Only we have to do is changing the value of the %ebp without fixed stack memory address. In this phase, In getbufn function, it stores 512 bytes of the characters with allocating random amount of the stack. Hence we cannot modify the value of %ebp with fixed value. We have to modify the value of %ebp with relative value. In the disassembled file, we can know that the difference between %ebp and %esp is 0x28, hence we can change ‘movl MemAdd, %ebp’ by ‘leal 0x28(%esp), %ebp’. Therefore, we have to inject

```
leal 0x28(%esp), %ebp
movl $0x2fffe42e, %eax
push $0x08048dd1
ret
```

Moreover, since we don’t know where to start, but only know where to end, I inserted nop operation as padding at the start of injected code. Therefore, my string in this phase is:
90 times 509
+ 8b 6c 24 28

+ bd 2e e4 ff 2f
+ 68 d1 8d 04 08
+ c3
+ a8 31 68 55.

The return address 0x556831a8 is not the unique solution. I found that we can change return address from min: 0x556831a8 to max: 0x556832a8.

3. Conclusion

<explain what you have learn in this lab. What was difficult, what was surprising, and so on>

In fact, I didn't know how buffer overflow works in practice. I didn't know how to return to the address, how to inject machine code. Also, I didn't know how %ebp and %esp make segmentation faults. Now, I've learned them by executing the codes.

The most difficult thing was dynamite mode. There are two big obstacles to me. One is injection code. In fact, I make INVALID result in phase two. I return to the middle of the fizz function right after the if-else statement. I think my solution was correct, but later I realized that it is invalid method when I struggle with dynamite mode.

The other obstacle is matching %ebp. I didn't know why my code makes segmentation faults. In the previous phases, I don't have to modify %ebp since there was a 'movl %esp, %ebp' command. With many gdb trials, I knew that %ebp address is something strange. Finally, I can modify %ebp value after 40 hours.

The most surprising thing was the hardness of hacking. If there is a slot like this example, I though that we can manipulate the value (but I didn't know how to). However, to modify the value, we have to inject many codes. If I were a programmer, I would not make those too many slots to insert string. Now, I knew that hacking by buffer overflow is difficult and space limited method.

Tips:

- do not copy-paste screenshots of your code. Format it properly as text

- length: do not exceed 5 pages. There is no lower limit, but one page is probably not detailed enough.