FORMAT STRING VULNERABILITY LAB

NAME : JASHWANTH REDDY GANGULA

SUID: 646254141

2.1 Task1 : Exploit the Vulnerability

/\* 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 here.\*/

/\* The secret value is stored on the heap \*/

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);

printf("The variable secret's value is 0x%8x (on 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 from user \*/

printf("Please enter a string\n");

scanf("%s", user\_input); /\* getting a string from user \*/

/\* Vulnerable place \*/

printf(user\_input);

printf("\n");

/\* Verify whether your attack is successful \*/

printf("The original secrets: 0x%x -- 0x%x\n", SECRET1, SECRET2);

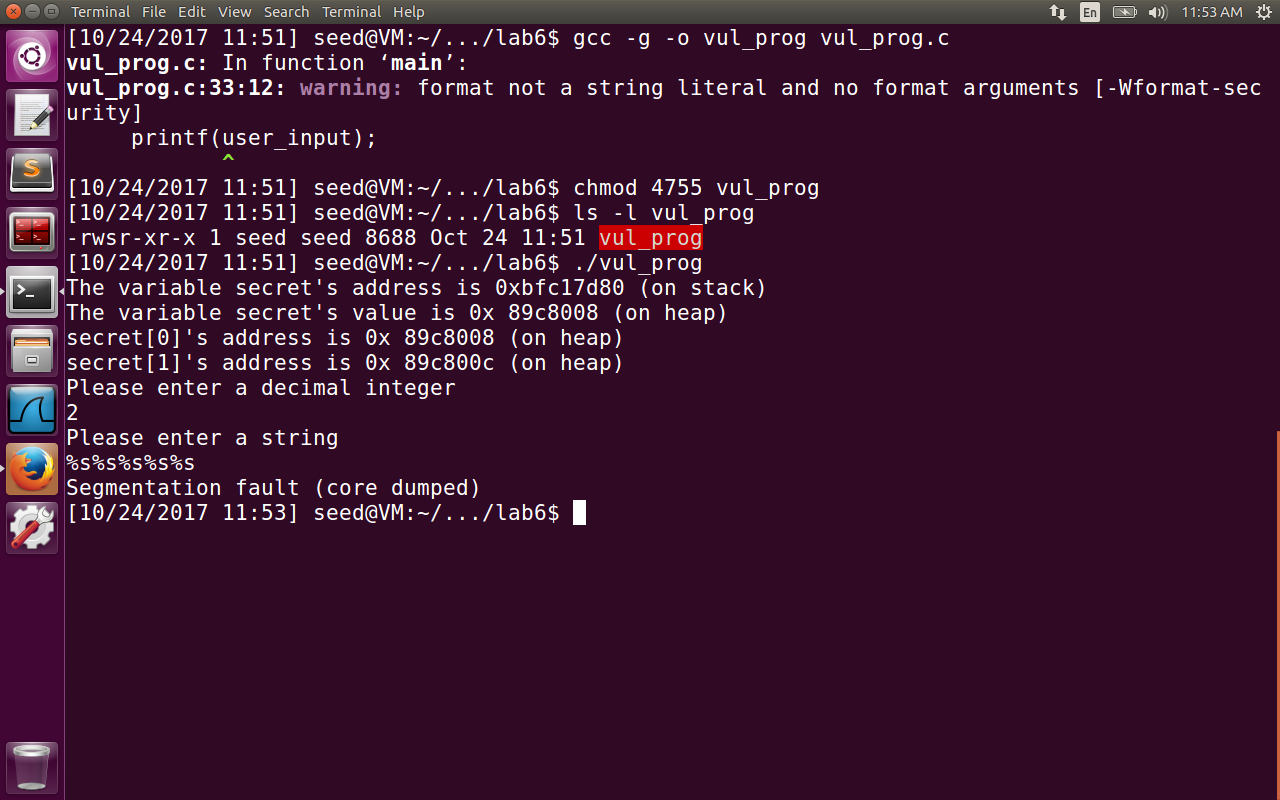
printf("The new secrets: 0x%x -- 0x%x\n", secret[0], secret[1]);

return 0;

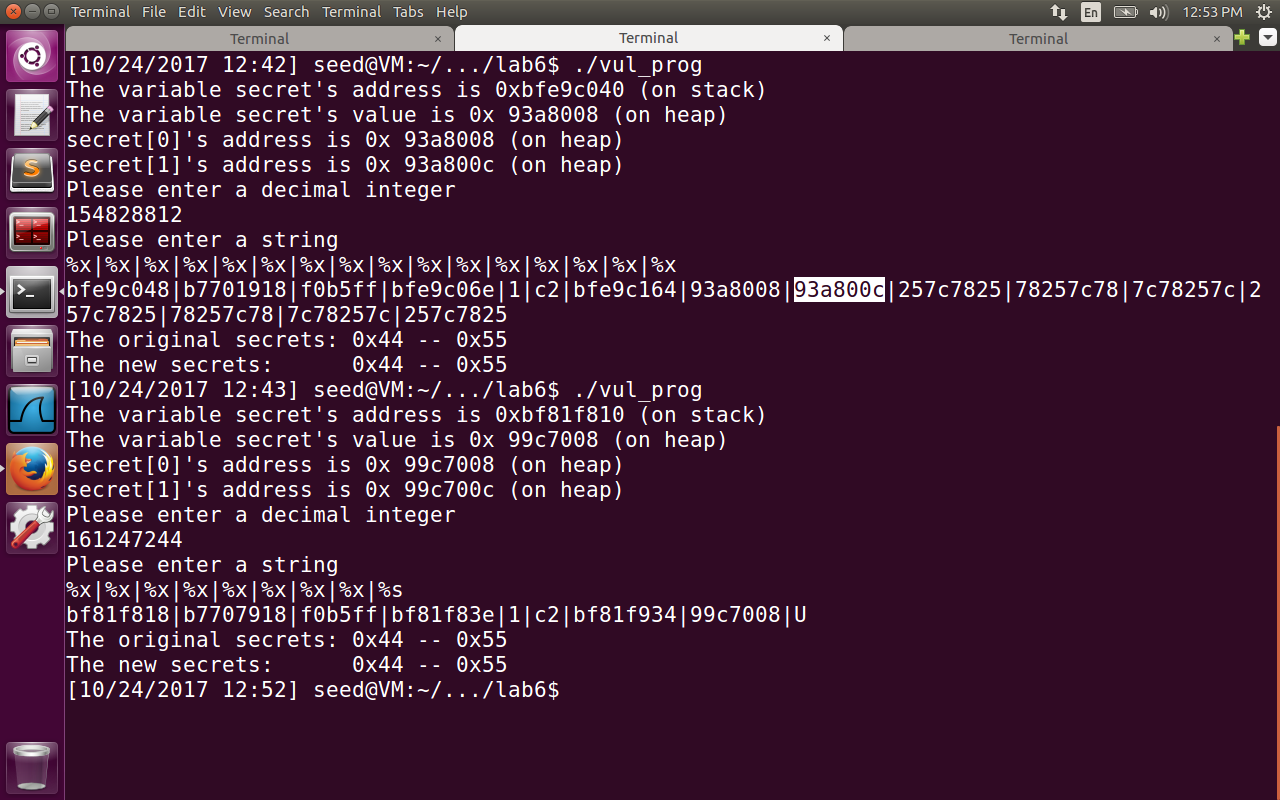
}

a) Crash the program

To crash the program, we need to use %s which points to invalid address like NULL. So we use trial and error to print many %s to crash the program.



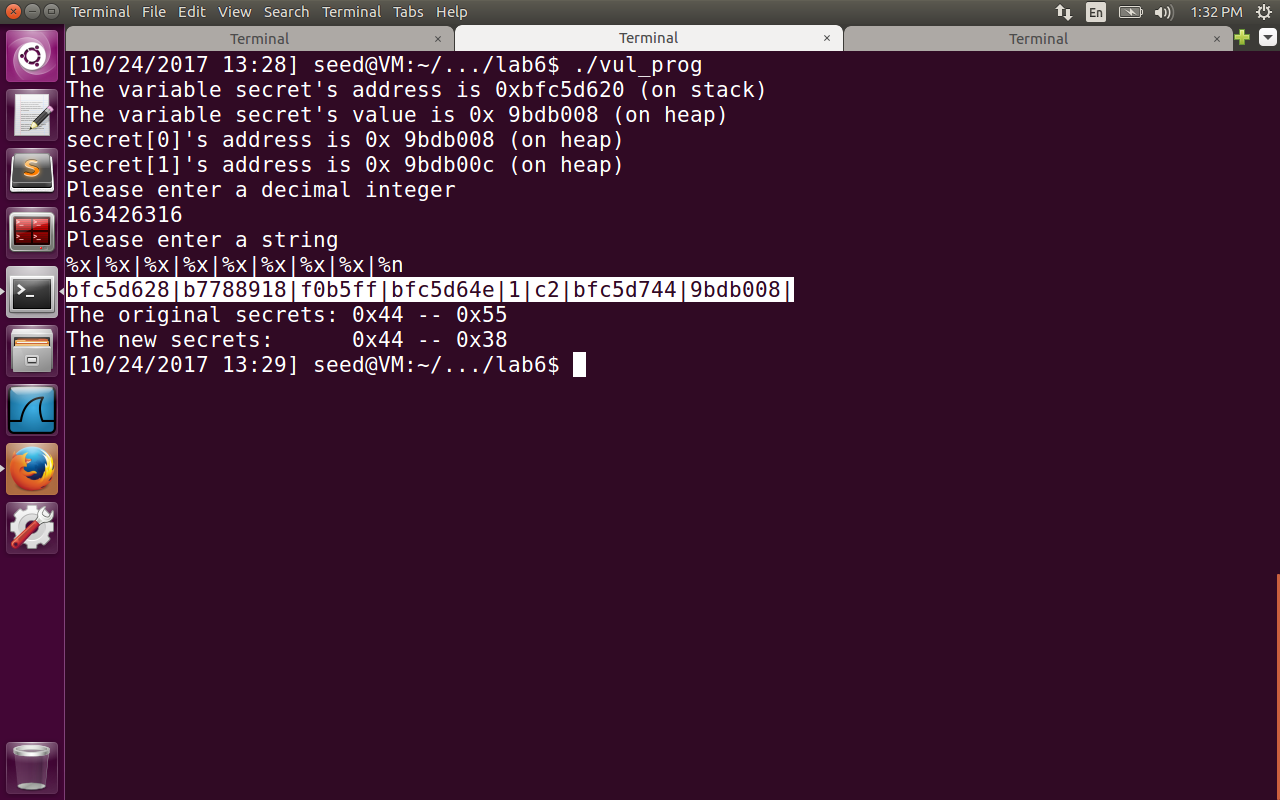
b) Print out secret[1] value



From the program, we know that addresses secret[0] and secret[1] are stored on heap. To print their values, we need to bring them onto stack. Once they are in the stack, we have to know what to specify in the format specifier using %x’s (how far is it offset from the current printf statement), so that we advance to that address, we can use %s at that address to print the secret value. In the current program it is at 9th %x from the output in the image. We put the stack[1] address in the int\_input by converting it into decimal, and using %s to print it. 0x55 = U when converted to ASCII and the output is as expected.

c) Modify the secret[1] value

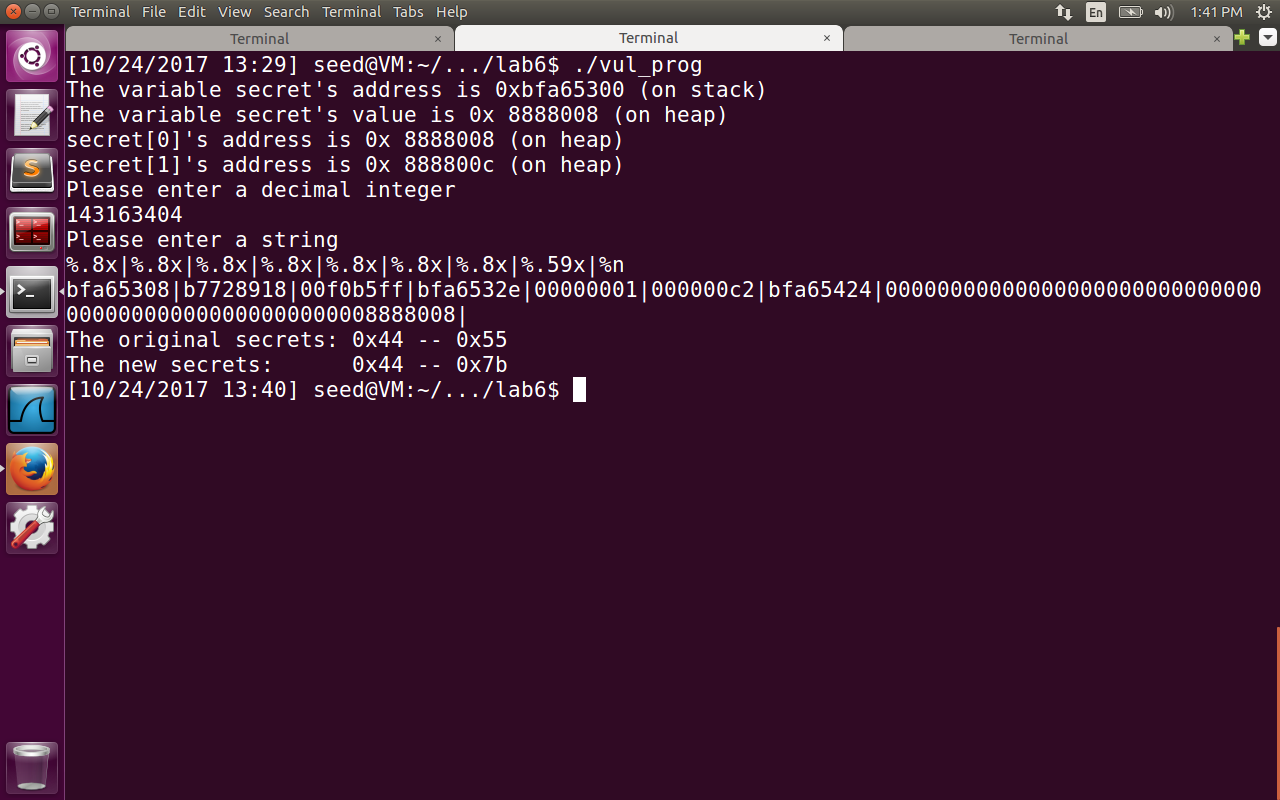
We can use %n too write to some address. It writes the number of bytes written so far to the address.



To modify the value at the address (0x9bdb00c), we brought it to stack using int\_input variable, which is at offset of 8 %x’s followed by %n to print the bytes written so far to stack[1]. In this case output is 0x38 = 56. If we count the characters in the highlighted text, they are 56. The output is as expected, and we modified the secret[1] value.

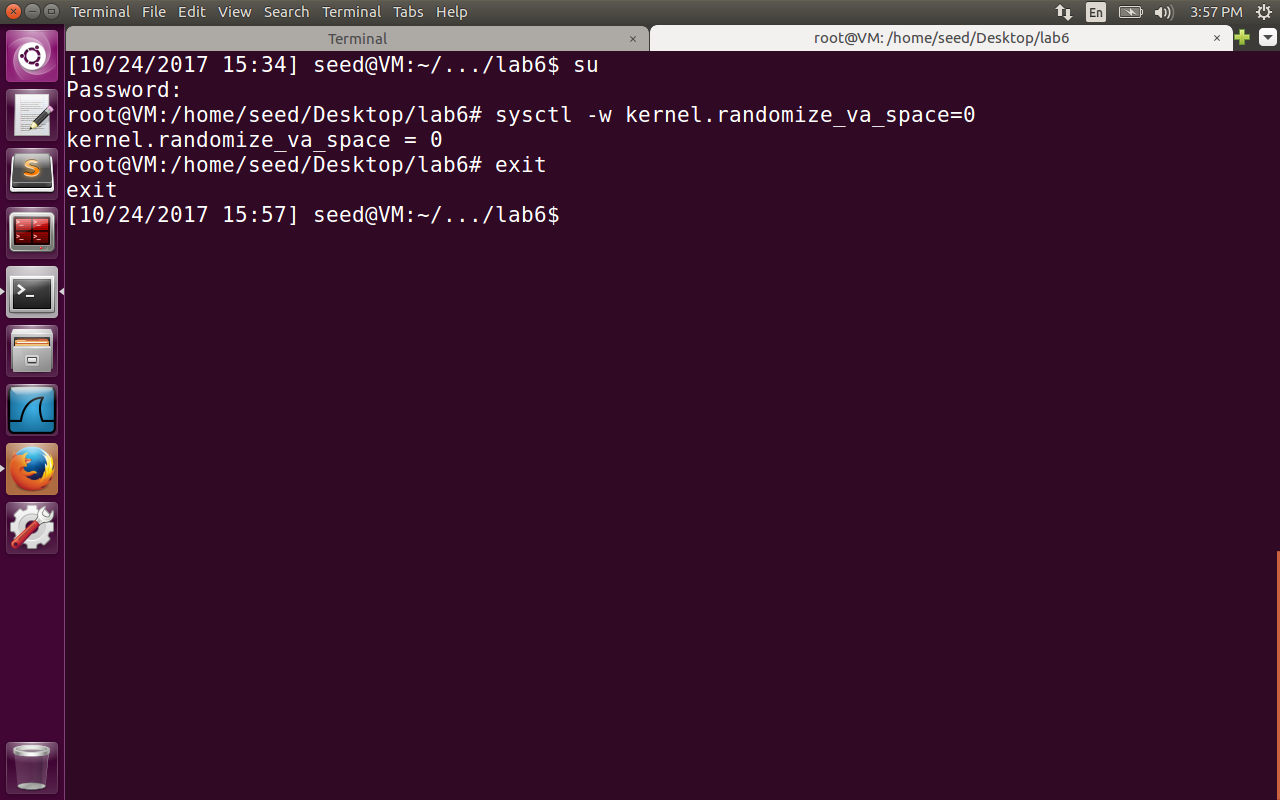
d) Modify to pre-determined value:

Let the predetermined value = 123 = { (ASCII 7B) value.

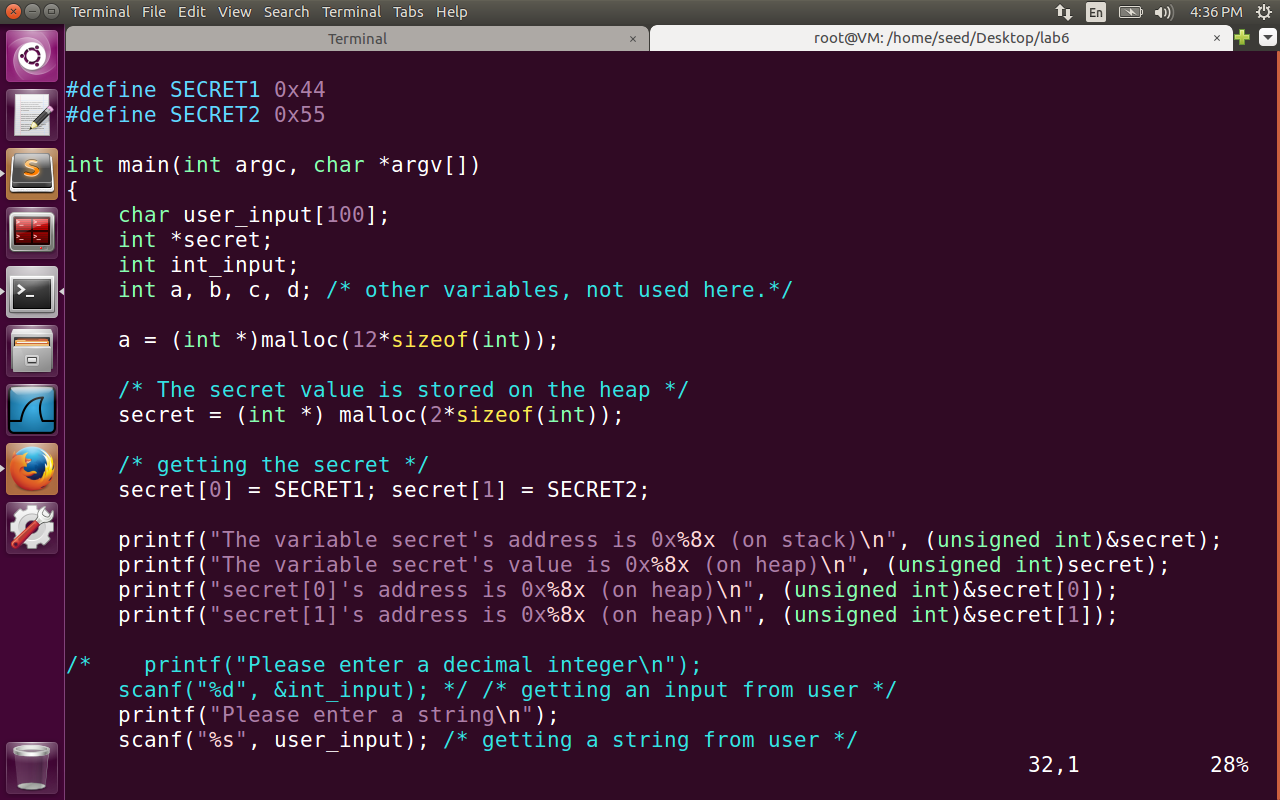


The new secret In the output is 7B as expected. To print 123 characters, we use %.8x to adjust each address to print 0’s in case it is less than 8 bytes. In the last specifier, we used %.59x so that total characters written will become 123.

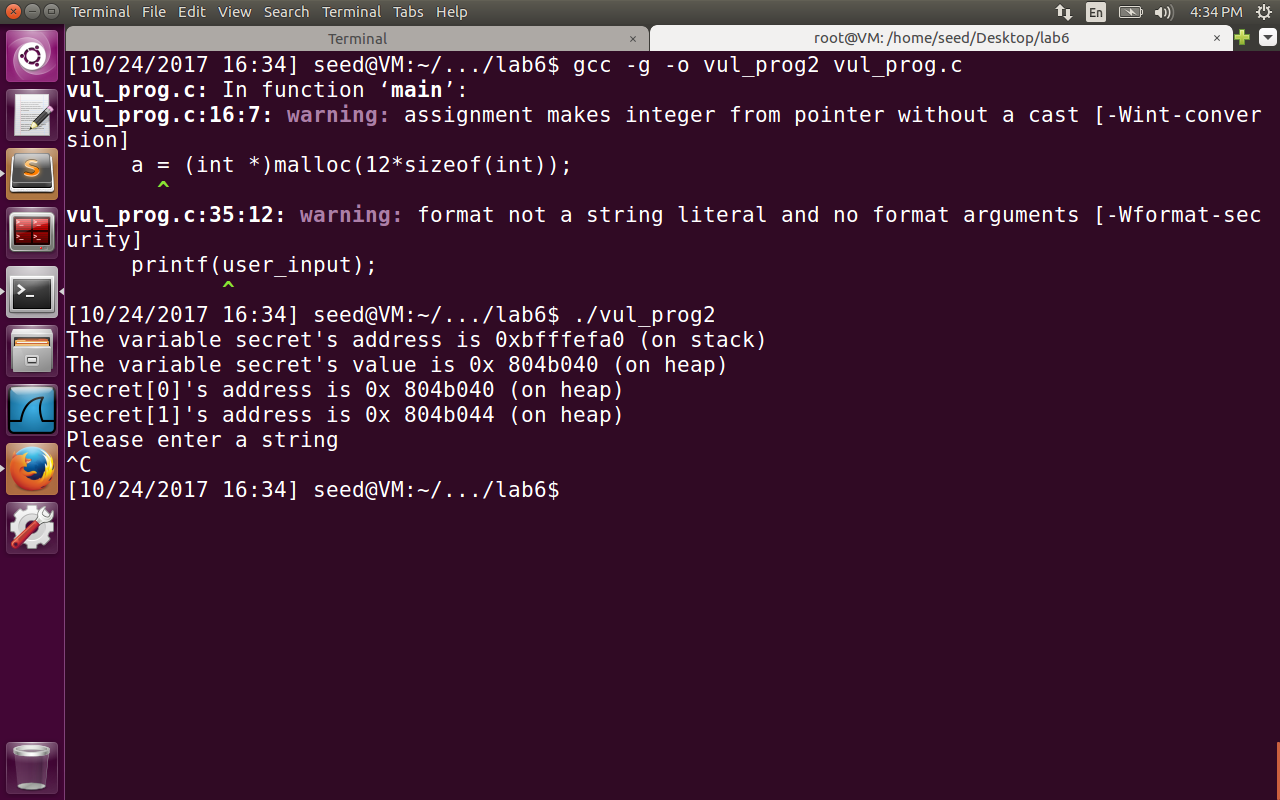
**2.2 Task2: Memory Randomization**



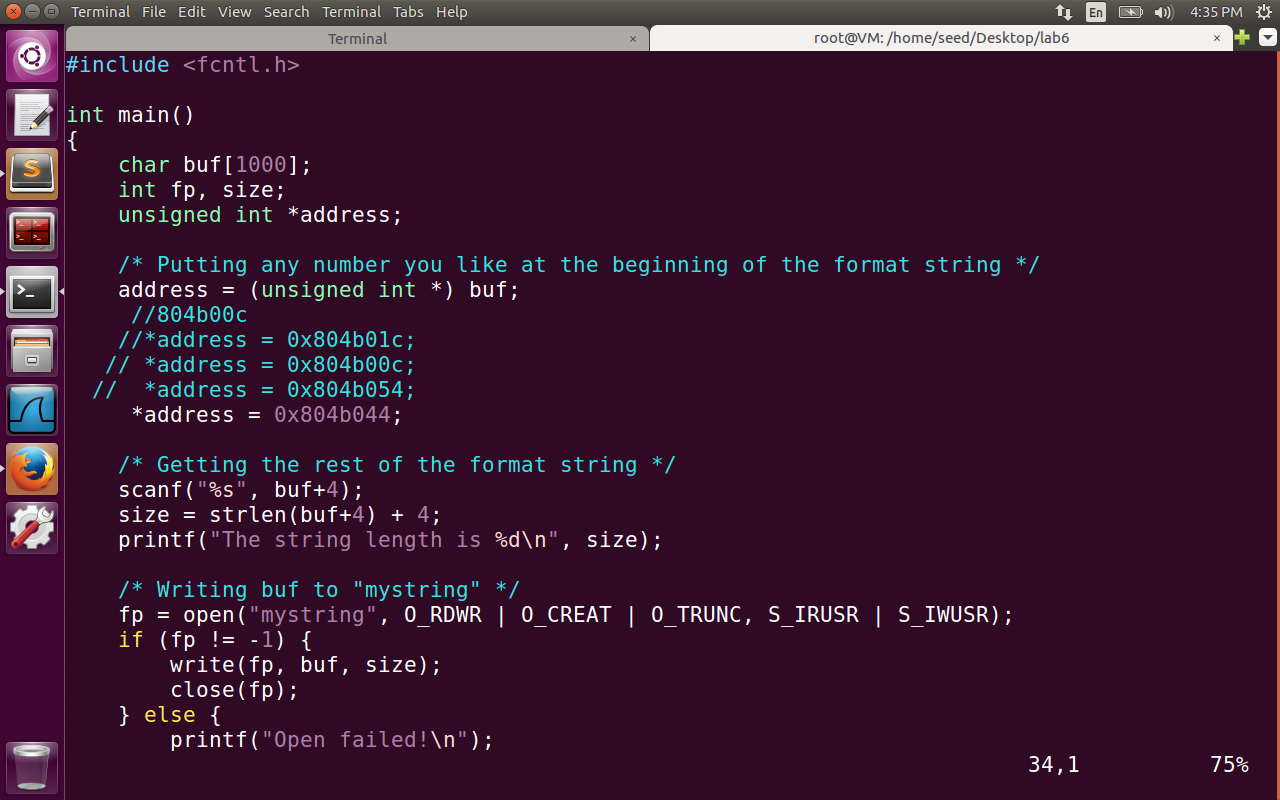
804b00C - This is the address we get using malloc for secret after disabling address space randomization. Scanf stops scanning if it encounters special characters. Here 0c is a special character. (form feed character). To get rid of this address, we add some malloc statements before this, so that the address does not contain special characters.



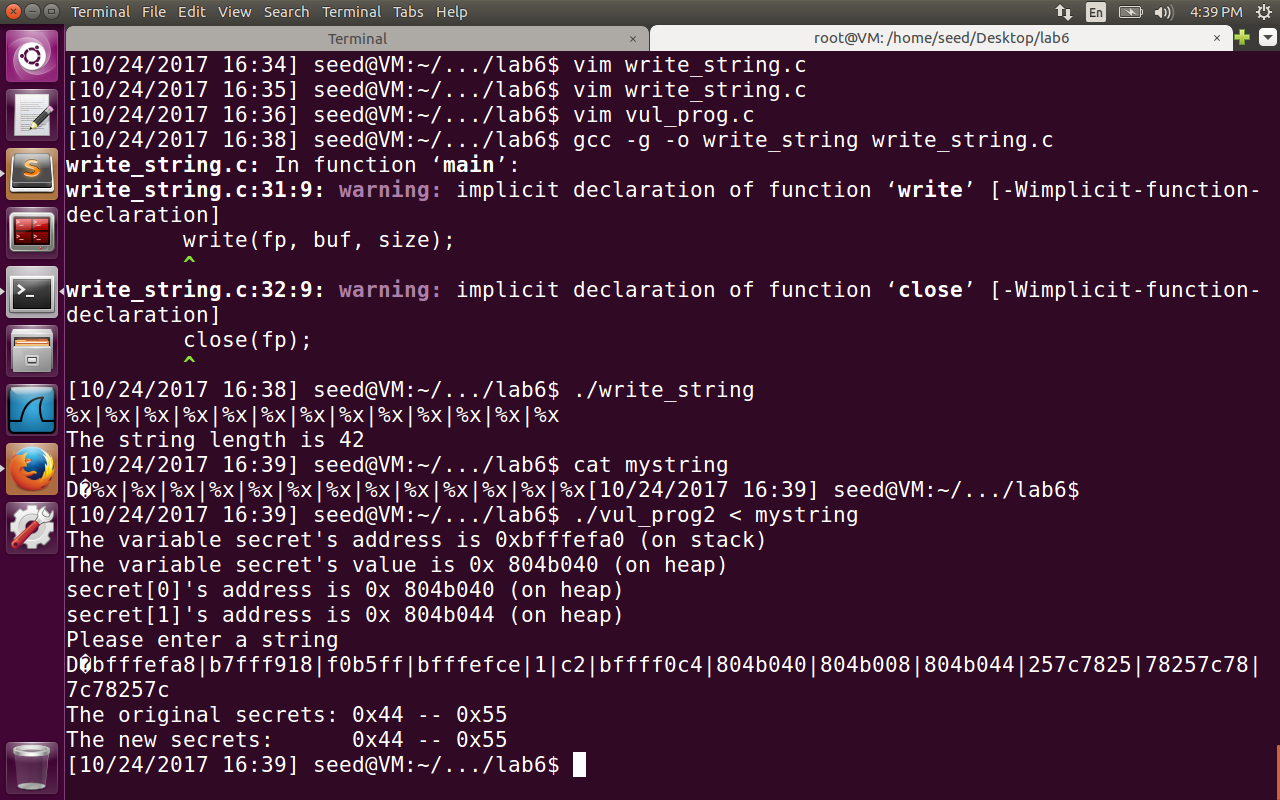
Here a = (int \***)**malloc(12\*sizeof(int)) is added before secret is allocated memory**.**



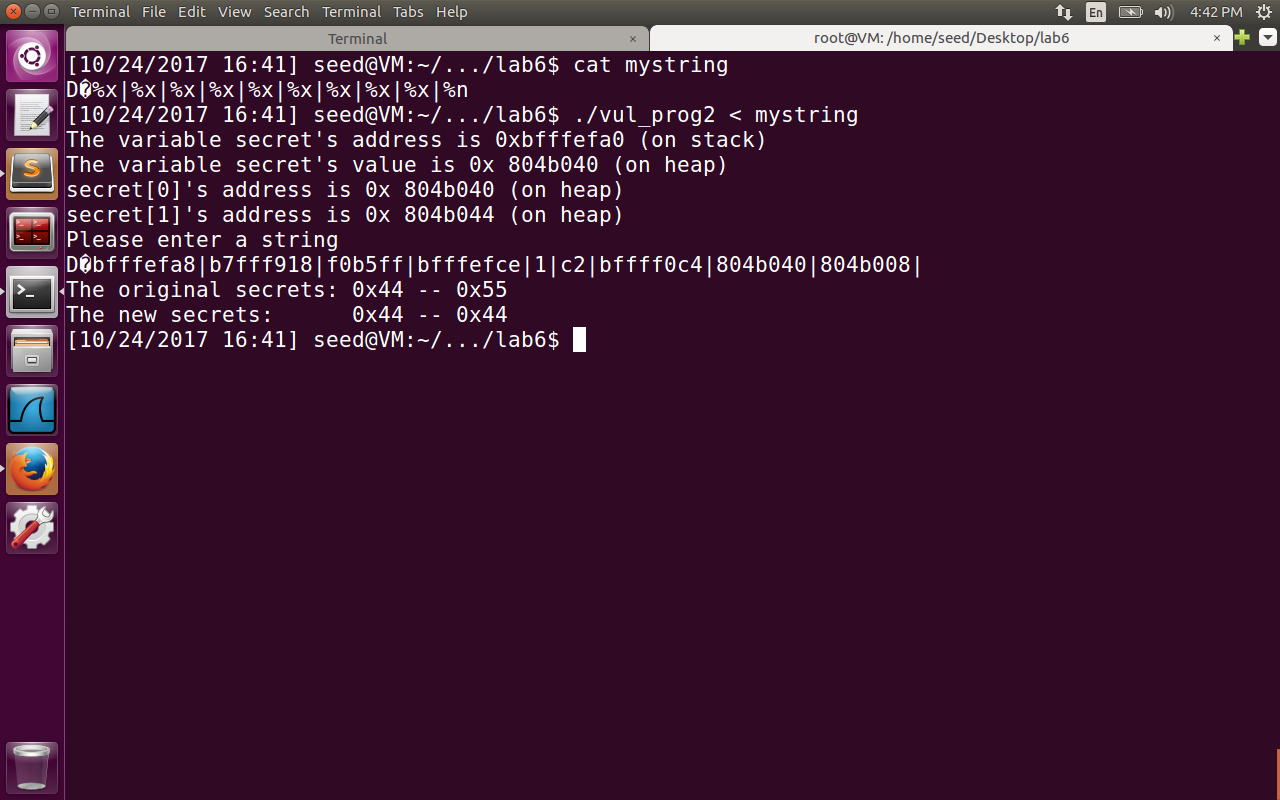
We got the address of secret[1] as 0x804b044. Add this address to the write\_string.c file.



Add some %x|%x|… to find the offset of secret[1] from the printf statement for the 1st iteration.



We observe that secret[1] is located 9 times after %x, so next time we add %n after 9 times %x to edit the secret value.



Because there are 68 characters printed, we got the value of secret[1] = 0x44 = 68(decimal).