CIS 643 Computer Security

Lab 2 Format String Vulnerability Lab

**Instructor: Wenliang (Kevin) Du**

Contents

Task 1: Exploit the vulnerability 2

Crash the program 3

Print out the secret[1] value 3

Modify the secret[1] value 4

Modify the secret[1] value to a pre-determined value 4

Task 2: Memory randomization 5

Print out the secret[1] value 7

Modify the secret[1] value 7

## Task 1: Exploit the vulnerability

The following codes are for testing string format vulnerability.

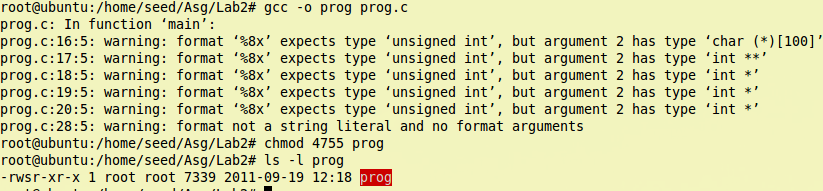


To make the input easy, I add one line to the program:

printf("secret[1]'s address is %u (on heap)\n", &secret[1]);

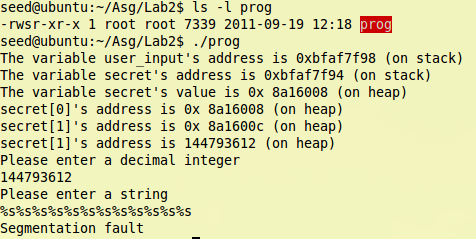
This code will output the address in decimal format.

Compile the codes and set the program as Set-UID program.



When compiling the program, it output some warnings. Those just are conversion problem; we would ignore them at current.

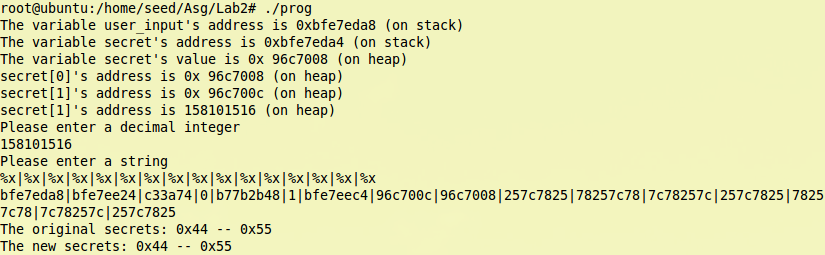
### Crash the program

To crash the program, we just need to input enough “%s” to make the program meet ‘\0’

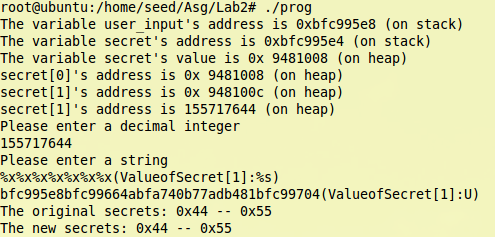
### Print out the secret[1] value

To hack the address of secret[1], we need to let int\_input to contain the address of secret[1], then using %x to move the pointer back when the program run call printf to try to output user\_input.

Following inputs are used to find where is the int\_input located, i.e. how many “%x” do we need to make pointer move to the int\_input position.



From the result, we see that 8th “%x” is output the value of int\_input.

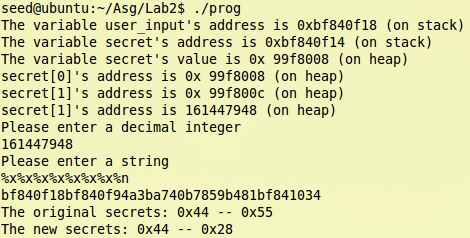
So we need “%s” after 7 “%x” to print the value of secret[1]

The print result is ‘U’, because the original value of secrete[1] is ‘0x44’, which corresponds to ASCII ‘U’.

### Modify the secret[1] value

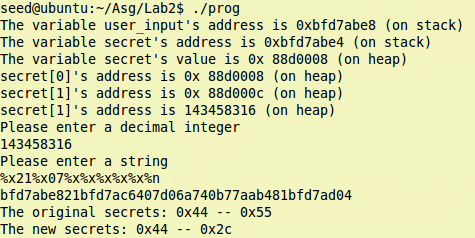
Usually, printf function could not set value to variable, but when format string contains “%n”, it will write the number of the string that written to the variable that address point to.

So we change the last “%s” of the format string that used in last requirement to “%n”, then we could set the secret[1] value as the number of the string that written.



We could see that the new value of secret[1] has been changed to 0x28.

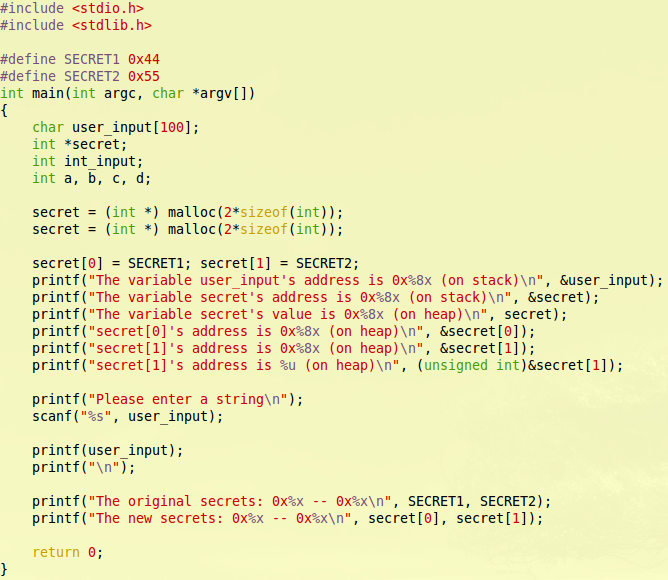
### Modify the secret[1] value to a pre-determined value

To make the value more flexible, we should add more characters in format string, so that the value assigned to secret[1] by “%n”

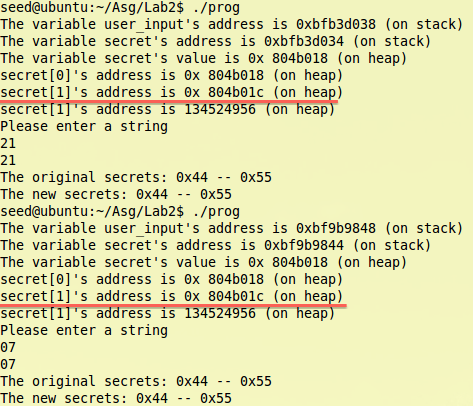
I add 4 numbers in format string, then the result of secret[1] become 0x2c, which is 4 bigger than value before: 0x28;

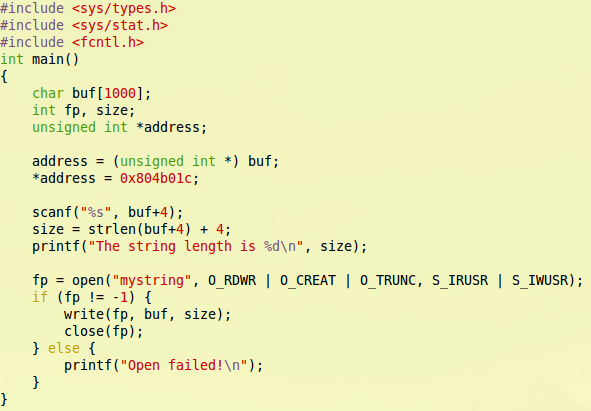
We could add any normal characters before “%n” except the format parameters.

## Task 2: Memory randomization

This time, the scanf for int\_input has been removed, so we need to add the address of screte[1] at the top of user\_input.(Because there is 0x0c in address, so I put two malloc function in the program.)

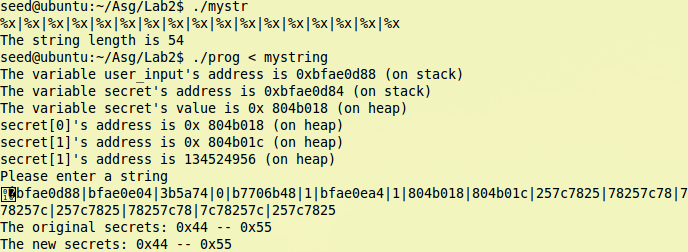
But random memory makes it hard to locate the program address, from the screenshot before, we could see that everytime we run the program the address of secrete[1] is different. So we need turn the randomization off.Macintosh HD:Users:Ider:Desktop:Screen Shot 2011-09-19 at 9.59.12 PM.png

Then run the program two times again, we could see the address is the same this time

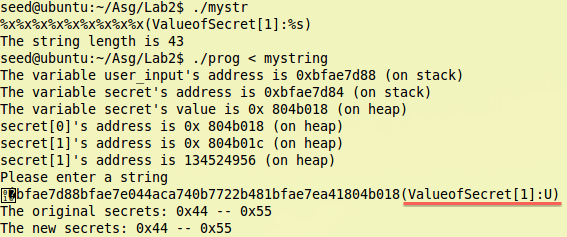
To pass the value to the program we need a help program

### Print out the secret[1] value

Test how many %x do we need to pass to program:



From the result output, we could know that we need 9 “%x” to make program pointer to the address that contain the address of secrete[1].

So, we can print the secret[1] now:

### Modify the secret[1] value

### Macintosh HD:Users:Ider:Desktop:Screen Shot 2011-09-19 at 11.22.08 PM.png