Daniel Oliveros

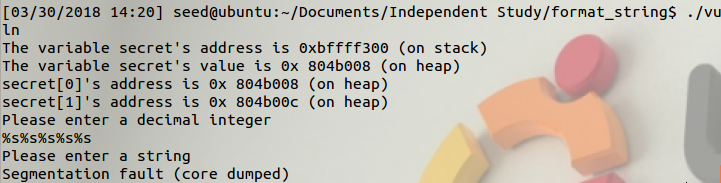
Garrett Bogart

Spring 2018 – Independent Study

Format String Vulnerability Lab

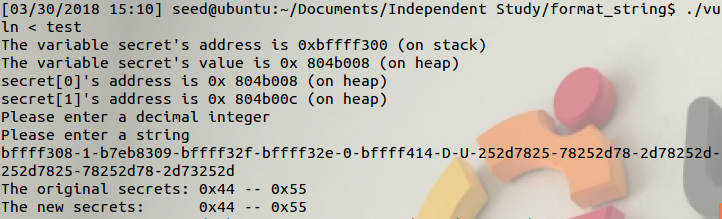
**Task 1:**

* **Crashing the program:** An easy way to crash the program is to input a series of %s, this makes it so the function searches through memory looking for these strings until it finds memory blocks it’s not allowed to access, at which point the program crashes.



* **Printing from stack:** We determined that the hexadecimal values 44 and 55 were being stored in the stack. We set up a file named test and put the following values in it, we then passed this file into our program.

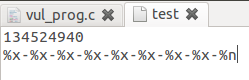


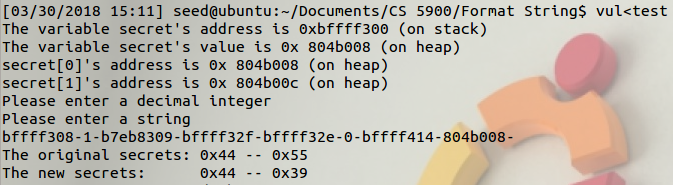


We determined how many %x values we needed to place before the %s through testing, the letter D is tied to the hexadecimal ascii value of 44, and the letter U is tied to the value 55, we printed these letters in the terminal, showing this memory can be printed using this exploit

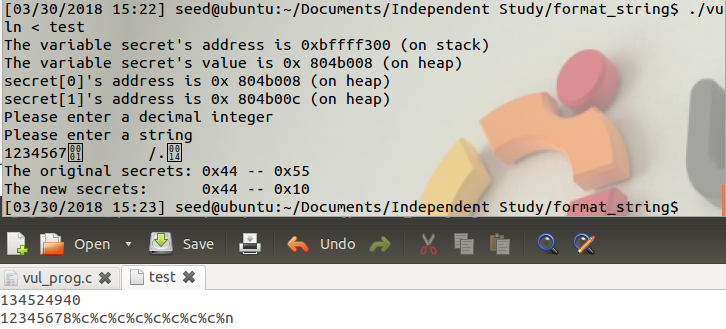
* **Changing the value of secret[1]:** There are two parts to solving this part. The first one is getting the address we want to change into the heap. An easy solution to this is to copy the location for secret[1] given to us in the terminal output (in this case 0x804b00c) and convert it to a decimal number (134524940), this number’s hexadecimal representation will be stored into memory, which is exactly what we want considering we’ll use it later to access a location in memory.

The second part is changing the value in that location. Since we can access it now, we can use the command %n to change the data in that location. Using the following file as an input we can change the value of secret[1]





* **Writing an arbitrary value:** %n keeps count of the number of things output before reaching it, by using %c we can make it so each value that came before the location is only counted as 1. Once we do that, we can then just simply add a bunch of numbers at the start of the second input to make the program print all of them out. In this example, we added 8 numbers before our 8 %c calls, so the program wrote 16 (10 in hexadecimal) to the value of secret[1]



**Task 2.**

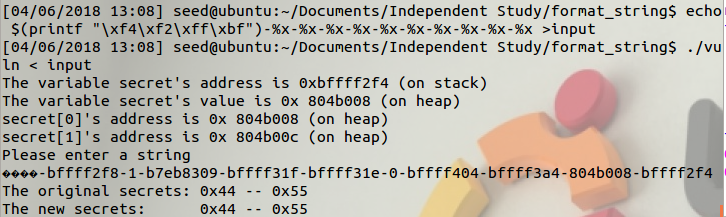
It is easy to exploit the program when address randomization is disabled. First we have to get rid of the integer input from the file.



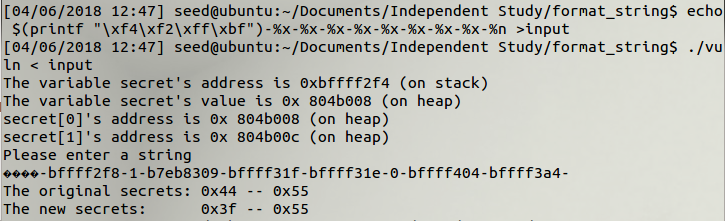
Now we have to figure out a new way of getting the address onto the stack. To do this we have to use command substitution. We first add the memory address to our stack by using printf so, we can later change the value.

echo $(printf "\xf4\xf2\xff\xbf")-%x-%x-%x-%x-%x-%x-%x-%x-%x-%x > input

As it can be seen our memory address is the final value printed out.



Now we just have to modify our input value slightly and, we can change the value at the memory address.



**Feedback:**

* This lab was written early and so it is unclear on a lot of things. Running it without memory randomization should be considered the default option to make testing code easier.
* It felt similar to the SQL injection lab in its execution, we’d recommend going with that one instead since it generally just feels more cohesive