#### ****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.

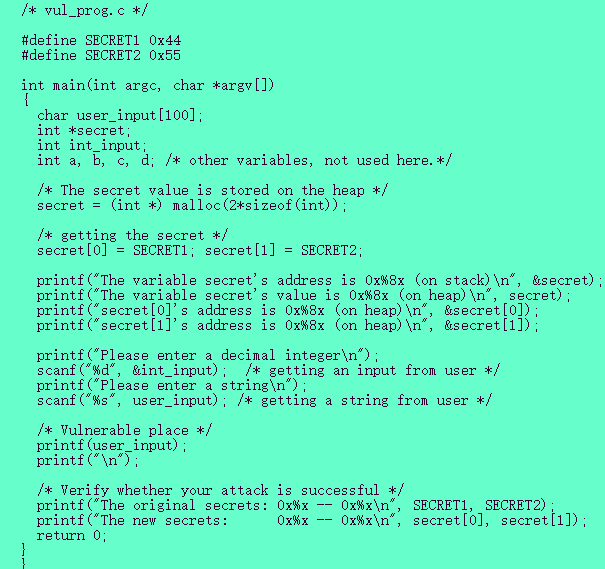
In the following program, you will be asked to provide an input, which will be saved in a buffer called user input. The program then prints out the buffer using printf. The program is a Set-UID program (the owner is root), i.e., it runs with the root privilege. Unfortunately, there is a format-string 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.

The program has two secret values stored in its memory, and you are interested in these secret values. However, the secret values are unknown to you, nor can you find them from reading the binary code (for the sake of simplicity, we hardcode the secrets using constants 0x44 and 0x55). Although you do not know the secret values, in practice, it is not so difficult to find out the memory address (the range or the exact value) of them (they are in consecutive addresses), because for many operating systems, the addresses are exactly the same anytime you run the program. In this lab, we just assume that you have already known the exact addresses. To achieve this, the program "intentionally" prints out the addresses for you. With such knowledge, your goal is to achieve the followings (not necessarily at the same time):

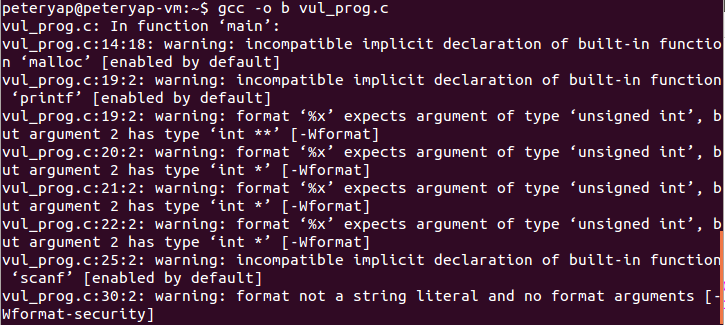
* Crash the program named "vul\_prog.c".
* Print out the secret[1] value.
* Modify the secret[1] value.
* Modify the secret[1] value to a pre-determined value.

Steps :

1. 按照要求把对应的代码写入c文件中并进行编译运行。

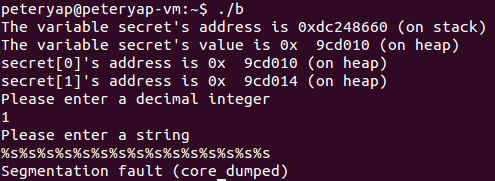


1. 编译运行vul\_prog.c，出现了很多Warning，先不管warning.

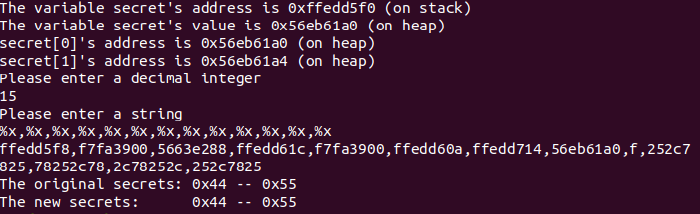


本程序的原理：通过malloc函数生成的secret[0] 和secret[1] 是被存放在堆中的，而用户需要输入的数字和string，以及指向secret的指针被存放在栈中，printf函数在处理的时候需要先将参数从右到左压入堆栈，然后遍历格式化的字符串，每次碰到%的格式化输出就需要从栈中弹出一个元素来与之对应，如果用户输入的带%的格式化输出参数的个数大于世纪传入的参数，就会出现内存泄漏的问题

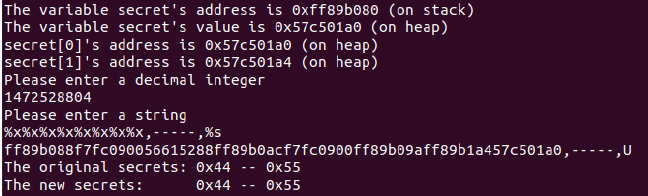
1. 问题1：可以通过输入比较多的%s%s%s%s%s来不断进行pop操作，使程序崩溃



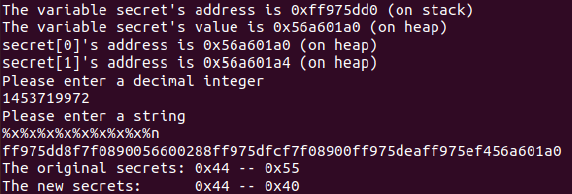
1. 问题2：我们发现int\_input在栈中的地址应该低于user\_input，因此可以先输入一个数字，通过%x不断机械能栈的pop操作，观察这个数字使第几个参数，实验过程中我发现，输入的15被放在第几个参数



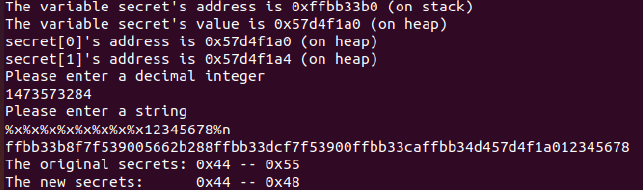
因此可以把输入secret[1]的地址作为int\_input输入%x%x%x%x%x%x%x%x,-----,%s来弹出前八个字符串，然后通过%s来读取int\_input指向的secret[1]的值，得到的结果是U



1. 问题3：用%n可以将已经输入的字符个数复制给传入的参数，∴输入%x%x%x%x%x%x%x%x%n把输入的字符个数写道对应的secret[1]的地址中，修改了内容



1. 问题4：计算前面的8个%x一共输出了0x40个字符的内容，也就是一个%x输出8个字符，因此想要更改secret[1]的值，只需要控制这个字符串输入的内容即可执行，比如在原有的基础上额外输入12345678就会把遍历改为0x48(72)



1. Lab2.4实验完成。