Basics of Heap Exploitation

And why heap exploitation isn’t that scary

Overview

Prerequisite knowledge

What is Heap?

Key functions

Malloc_chunk

Malloc_state

Bins and chunks

Basic techniques

Prerequisite knowledge

Basic PWN

What is the Heap?

The heap is dynamically sized blocks of memory.

You use the heap when the amount of memory you want is unknown before runtime.

Whenever you require memory, you can `request` memory from the heap, and it’ll return a pointer to the memory it carved out of the global heap segment for you.

Key Functions

void* malloc(size_t bytes)

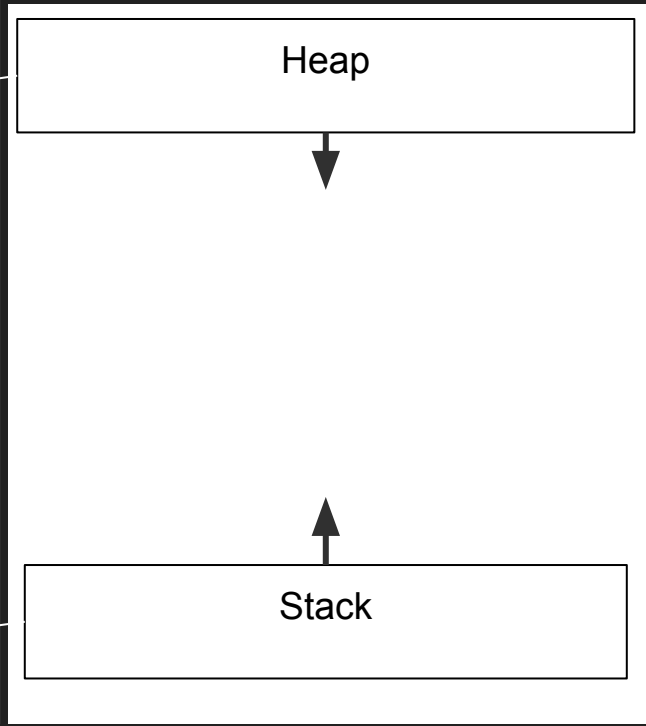
This returns a pointer to heap memory. It expects the programmer to know the size of the buffer that is returned, and not to write past the end of that buffer (lest bad things happen).

void free(void* ptr)

This tells the heap that it’s free to do whatever it wants to the chunk referenced by this pointer. Usage of a free’d pointer results in undefined behavior.

Where does the memory come from?

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x555555554000     0x555555556000 r-xp     2000 0      /ctf/hitcon/babytcache/binary
    0x555555755000     0x555555756000 r--p     1000 1000   /ctf/hitcon/babytcache/binary
    0x555555756000     0x555555757000 rw-p     1000 2000   /ctf/hitcon/babytcache/binary
    0x555555757000     0x555555778000 rw-p    21000 0      [heap]
    0x7ffff7dfe000     0x7ffff7e20000 r--p    22000 0      /usr/lib64/libc-2.28.so
    0x7ffff7e20000     0x7ffff7f6d000 r-xp   14d000 22000  /usr/lib64/libc-2.28.so
    0x7ffff7f6d000     0x7ffff7fb9000 r--p    4c000 16f000 /usr/lib64/libc-2.28.so
    0x7ffff7fb9000     0x7ffff7fba000 ---p     1000 1bb000 /usr/lib64/libc-2.28.so
    0x7ffff7fba000     0x7ffff7fbe000 r--p     4000 1bb000 /usr/lib64/libc-2.28.so
    0x7ffff7fbe000     0x7ffff7fc0000 rw-p     2000 1bf000 /usr/lib64/libc-2.28.so
    0x7ffff7fc0000     0x7ffff7fc6000 rw-p     6000 0
    0x7ffff7fcd000     0x7ffff7fd0000 r--p     3000 0      [vvar]
    0x7ffff7fd0000     0x7ffff7fd2000 r-xp     2000 0      [vdso]
    0x7ffff7fd2000     0x7ffff7fd3000 r--p     1000 0      /usr/lib64/ld-2.28.so
    0x7ffff7fd3000     0x7ffff7ff3000 r-xp    20000 1000   /usr/lib64/ld-2.28.so
    0x7ffff7ff3000     0x7ffff7ffb000 r--p     8000 21000  /usr/lib64/ld-2.28.so
    0x7ffff7ffc000     0x7ffff7ffd000 r--p     1000 29000  /usr/lib64/ld-2.28.so
    0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000 2a000  /usr/lib64/ld-2.28.so
    0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000 0
    0x7ffffffde000     0x7ffffffff000 rw-p    21000 0      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp     1000 0      [vsyscall]
```



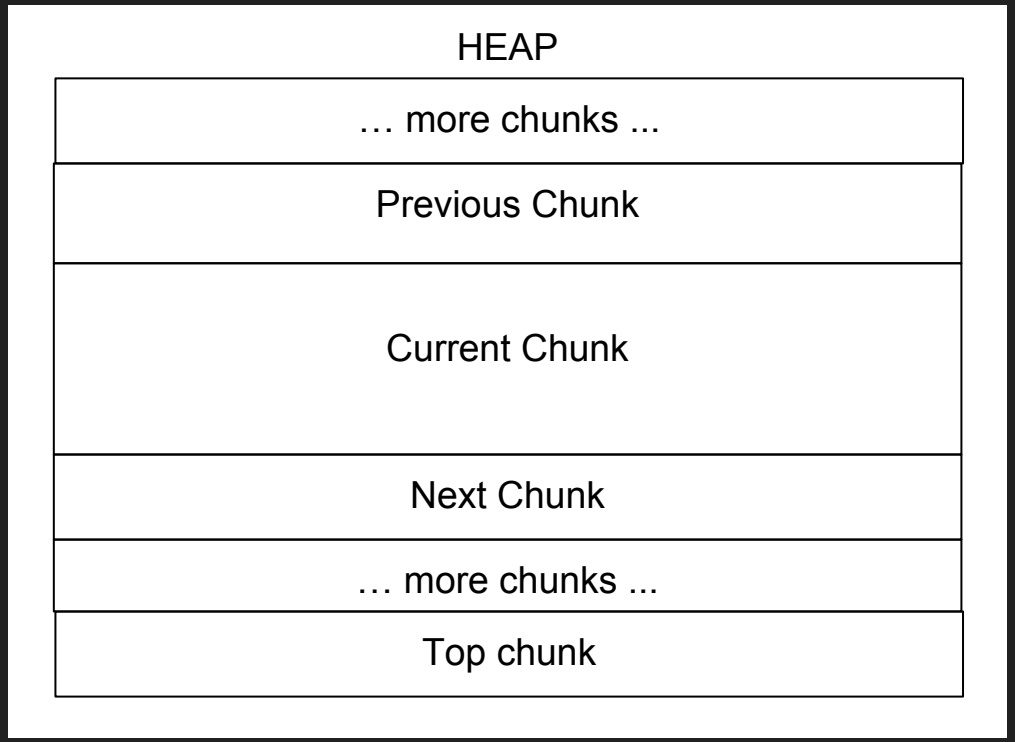
Lets see the heap in action

```
void *prev = malloc(0xf8);
void *curr = malloc(0xf8);
void* next = malloc(0xf8);
void* guard = malloc(0x78);

memset(prev, 0x41, 0xf8);
memset(curr, 0x42, 0xf8);
memset(next, 0x43, 0xf8);
```

This is the debug output of the heap if we make a few allocations →

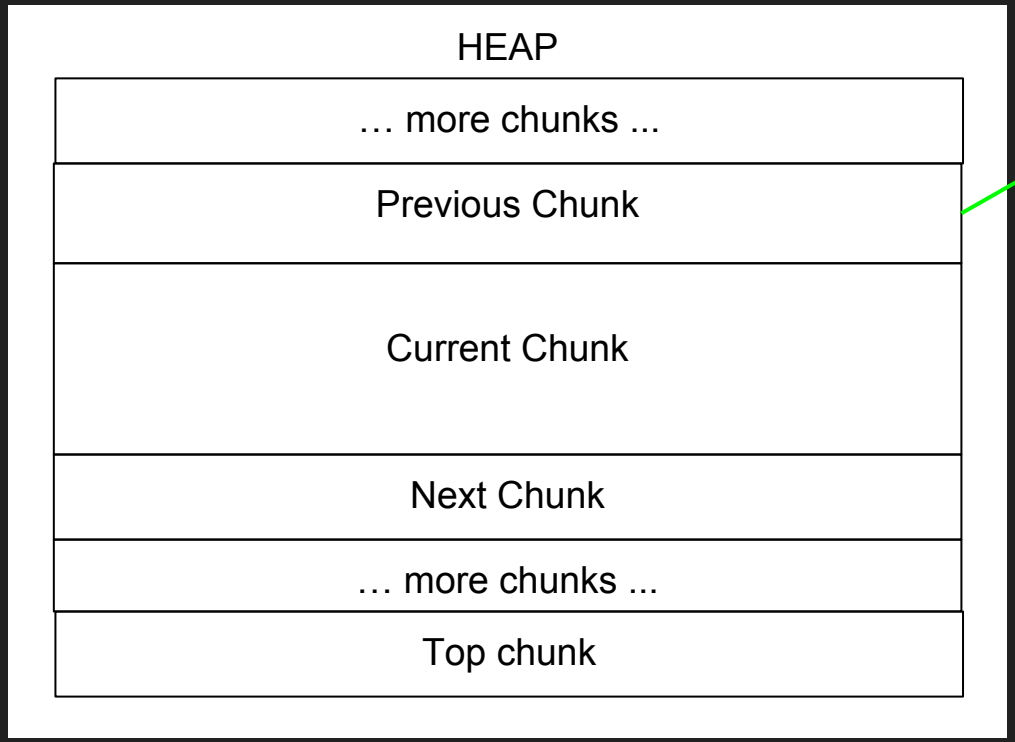
```
pwndbg> x/100xg 0x555555756000
0x555555756000: 0x0000000000000000      0x0000000000000101
0x555555756010: 0x4141414141414141      0x4141414141414141
0x555555756020: 0x4141414141414141      0x4141414141414141
0x555555756030: 0x4141414141414141      0x4141414141414141
0x555555756040: 0x4141414141414141      0x4141414141414141
0x555555756050: 0x4141414141414141      0x4141414141414141
0x555555756060: 0x4141414141414141      0x4141414141414141
0x555555756070: 0x4141414141414141      0x4141414141414141
0x555555756080: 0x4141414141414141      0x4141414141414141
0x555555756090: 0x4141414141414141      0x4141414141414141
0x5555557560a0: 0x4141414141414141      0x4141414141414141
0x5555557560b0: 0x4141414141414141      0x4141414141414141
0x5555557560c0: 0x4141414141414141      0x4141414141414141
0x5555557560d0: 0x4141414141414141      0x4141414141414141
0x5555557560e0: 0x4141414141414141      0x4141414141414141
0x5555557560f0: 0x4141414141414141      0x4141414141414141
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
0x555555756210: 0x4343434343434343      0x4343434343434343
0x555555756220: 0x4343434343434343      0x4343434343434343
0x555555756230: 0x4343434343434343      0x4343434343434343
0x555555756240: 0x4343434343434343      0x4343434343434343
0x555555756250: 0x4343434343434343      0x4343434343434343
0x555555756260: 0x4343434343434343      0x4343434343434343
0x555555756270: 0x4343434343434343      0x4343434343434343
0x555555756280: 0x4343434343434343      0x4343434343434343
0x555555756290: 0x4343434343434343      0x4343434343434343
0x5555557562a0: 0x4343434343434343      0x4343434343434343
0x5555557562b0: 0x4343434343434343      0x4343434343434343
0x5555557562c0: 0x4343434343434343      0x4343434343434343
0x5555557562d0: 0x4343434343434343      0x4343434343434343
0x5555557562e0: 0x4343434343434343      0x4343434343434343
0x5555557562f0: 0x4343434343434343      0x4343434343434343
0x555555756300: 0x4343434343434343      0x0000000000000081
0x555555756310: 0x0000000000000000      0x0000000000000000
```

Lets see the heap in action



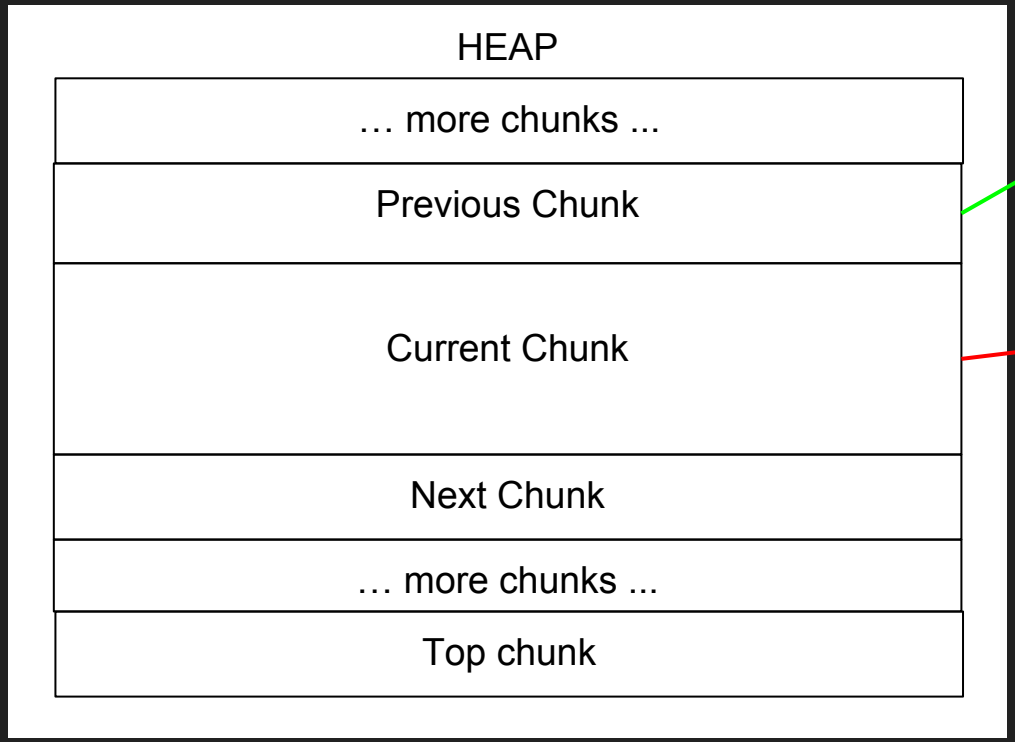

```
pwndbg> x/100xg 0x555555756000
0x555555756000: 0x0000000000000000      0x0000000000000101
0x555555756010: 0x4141414141414141      0x4141414141414141
0x555555756020: 0x4141414141414141      0x4141414141414141
0x555555756030: 0x4141414141414141      0x4141414141414141
0x555555756040: 0x4141414141414141      0x4141414141414141
0x555555756050: 0x4141414141414141      0x4141414141414141
0x555555756060: 0x4141414141414141      0x4141414141414141
0x555555756070: 0x4141414141414141      0x4141414141414141
0x555555756080: 0x4141414141414141      0x4141414141414141
0x555555756090: 0x4141414141414141      0x4141414141414141
0x5555557560a0: 0x4141414141414141      0x4141414141414141
0x5555557560b0: 0x4141414141414141      0x4141414141414141
0x5555557560c0: 0x4141414141414141      0x4141414141414141
0x5555557560d0: 0x4141414141414141      0x4141414141414141
0x5555557560e0: 0x4141414141414141      0x4141414141414141
0x5555557560f0: 0x4141414141414141      0x4141414141414141
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
0x555555756210: 0x4343434343434343      0x4343434343434343
0x555555756220: 0x4343434343434343      0x4343434343434343
0x555555756230: 0x4343434343434343      0x4343434343434343
0x555555756240: 0x4343434343434343      0x4343434343434343
0x555555756250: 0x4343434343434343      0x4343434343434343
0x555555756260: 0x4343434343434343      0x4343434343434343
0x555555756270: 0x4343434343434343      0x4343434343434343
0x555555756280: 0x4343434343434343      0x4343434343434343
0x555555756290: 0x4343434343434343      0x4343434343434343
0x5555557562a0: 0x4343434343434343      0x4343434343434343
0x5555557562b0: 0x4343434343434343      0x4343434343434343
0x5555557562c0: 0x4343434343434343      0x4343434343434343
0x5555557562d0: 0x4343434343434343      0x4343434343434343
0x5555557562e0: 0x4343434343434343      0x4343434343434343
0x5555557562f0: 0x4343434343434343      0x4343434343434343
0x555555756300: 0x4343434343434343      0x0000000000000081
0x555555756310: 0x0000000000000000      0x0000000000000000
```

Lets see the heap in action



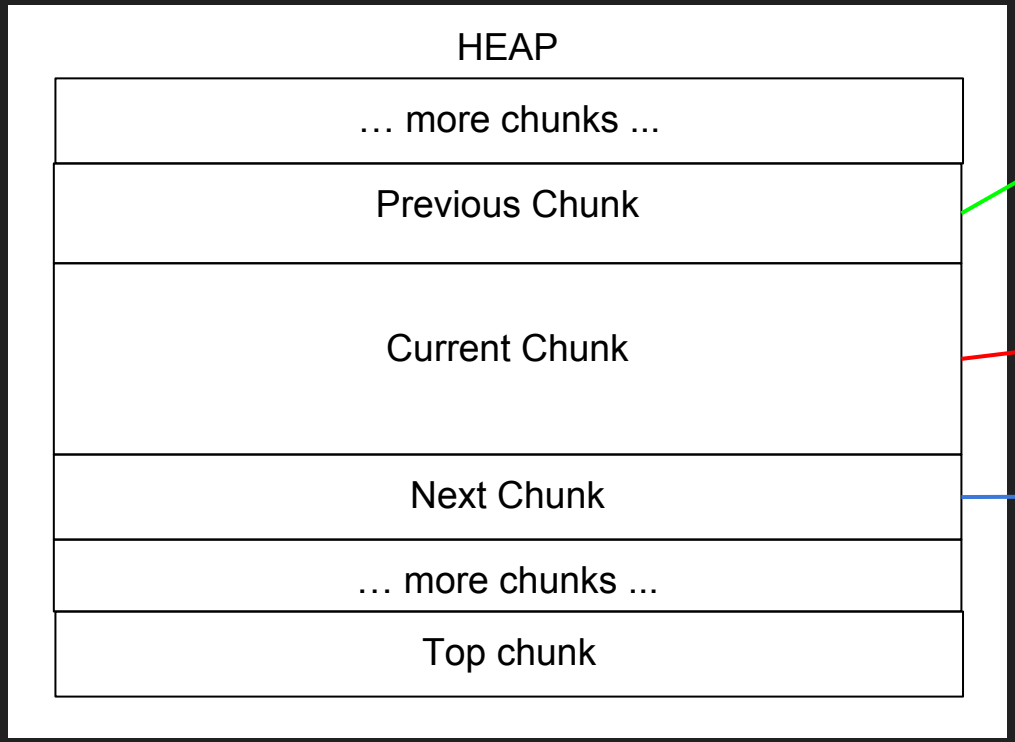

```
pwndbg> x/100xg 0x555555756000
0x555555756000: 0x0000000000000000      0x0000000000000101
0x555555756010: 0x4141414141414141      0x4141414141414141
0x555555756020: 0x4141414141414141      0x4141414141414141
0x555555756030: 0x4141414141414141      0x4141414141414141
0x555555756040: 0x4141414141414141      0x4141414141414141
0x555555756050: 0x4141414141414141      0x4141414141414141
0x555555756060: 0x4141414141414141      0x4141414141414141
0x555555756070: 0x4141414141414141      0x4141414141414141
0x555555756080: 0x4141414141414141      0x4141414141414141
0x555555756090: 0x4141414141414141      0x4141414141414141
0x5555557560a0: 0x4141414141414141      0x4141414141414141
0x5555557560b0: 0x4141414141414141      0x4141414141414141
0x5555557560c0: 0x4141414141414141      0x4141414141414141
0x5555557560d0: 0x4141414141414141      0x4141414141414141
0x5555557560e0: 0x4141414141414141      0x4141414141414141
0x5555557560f0: 0x4141414141414141      0x4141414141414141
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
0x555555756210: 0x4343434343434343      0x4343434343434343
0x555555756220: 0x4343434343434343      0x4343434343434343
0x555555756230: 0x4343434343434343      0x4343434343434343
0x555555756240: 0x4343434343434343      0x4343434343434343
0x555555756250: 0x4343434343434343      0x4343434343434343
0x555555756260: 0x4343434343434343      0x4343434343434343
0x555555756270: 0x4343434343434343      0x4343434343434343
0x555555756280: 0x4343434343434343      0x4343434343434343
0x555555756290: 0x4343434343434343      0x4343434343434343
0x5555557562a0: 0x4343434343434343      0x4343434343434343
0x5555557562b0: 0x4343434343434343      0x4343434343434343
0x5555557562c0: 0x4343434343434343      0x4343434343434343
0x5555557562d0: 0x4343434343434343      0x4343434343434343
0x5555557562e0: 0x4343434343434343      0x4343434343434343
0x5555557562f0: 0x4343434343434343      0x4343434343434343
0x555555756300: 0x4343434343434343      0x0000000000000081
0x555555756310: 0x0000000000000000      0x0000000000000000
```

Lets see the heap in action




```
pwndbg> x/100xg 0x555555756000
0x555555756000: 0x0000000000000000      0x0000000000000101
0x555555756010: 0x4141414141414141      0x4141414141414141
0x555555756020: 0x4141414141414141      0x4141414141414141
0x555555756030: 0x4141414141414141      0x4141414141414141
0x555555756040: 0x4141414141414141      0x4141414141414141
0x555555756050: 0x4141414141414141      0x4141414141414141
0x555555756060: 0x4141414141414141      0x4141414141414141
0x555555756070: 0x4141414141414141      0x4141414141414141
0x555555756080: 0x4141414141414141      0x4141414141414141
0x555555756090: 0x4141414141414141      0x4141414141414141
0x5555557560a0: 0x4141414141414141      0x4141414141414141
0x5555557560b0: 0x4141414141414141      0x4141414141414141
0x5555557560c0: 0x4141414141414141      0x4141414141414141
0x5555557560d0: 0x4141414141414141      0x4141414141414141
0x5555557560e0: 0x4141414141414141      0x4141414141414141
0x5555557560f0: 0x4141414141414141      0x4141414141414141
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
0x555555756210: 0x4343434343434343      0x4343434343434343
0x555555756220: 0x4343434343434343      0x4343434343434343
0x555555756230: 0x4343434343434343      0x4343434343434343
0x555555756240: 0x4343434343434343      0x4343434343434343
0x555555756250: 0x4343434343434343      0x4343434343434343
0x555555756260: 0x4343434343434343      0x4343434343434343
0x555555756270: 0x4343434343434343      0x4343434343434343
0x555555756280: 0x4343434343434343      0x4343434343434343
0x555555756290: 0x4343434343434343      0x4343434343434343
0x5555557562a0: 0x4343434343434343      0x4343434343434343
0x5555557562b0: 0x4343434343434343      0x4343434343434343
0x5555557562c0: 0x4343434343434343      0x4343434343434343
0x5555557562d0: 0x4343434343434343      0x4343434343434343
0x5555557562e0: 0x4343434343434343      0x4343434343434343
0x5555557562f0: 0x4343434343434343      0x4343434343434343
0x555555756300: 0x4343434343434343      0x0000000000000081
0x555555756310: 0x0000000000000000      0x0000000000000000
```

Lets see the heap in action



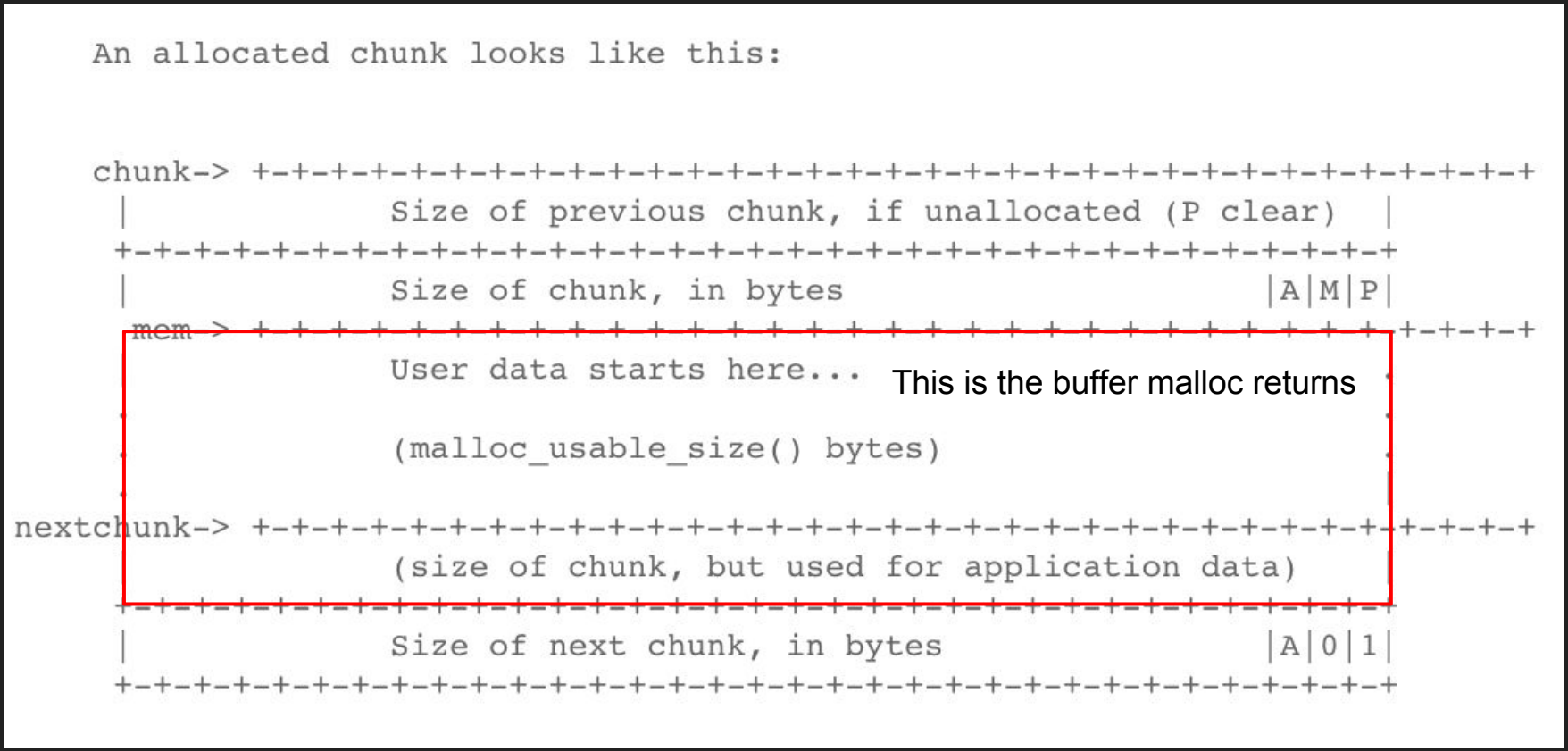

```
pwndbg> x/100xg 0x555555756000
0x555555756000: 0x0000000000000000      0x0000000000000101
0x555555756010: 0x4141414141414141      0x4141414141414141
0x555555756020: 0x4141414141414141      0x4141414141414141
0x555555756030: 0x4141414141414141      0x4141414141414141
0x555555756040: 0x4141414141414141      0x4141414141414141
0x555555756050: 0x4141414141414141      0x4141414141414141
0x555555756060: 0x4141414141414141      0x4141414141414141
0x555555756070: 0x4141414141414141      0x4141414141414141
0x555555756080: 0x4141414141414141      0x4141414141414141
0x555555756090: 0x4141414141414141      0x4141414141414141
0x5555557560a0: 0x4141414141414141      0x4141414141414141
0x5555557560b0: 0x4141414141414141      0x4141414141414141
0x5555557560c0: 0x4141414141414141      0x4141414141414141
0x5555557560d0: 0x4141414141414141      0x4141414141414141
0x5555557560e0: 0x4141414141414141      0x4141414141414141
0x5555557560f0: 0x4141414141414141      0x4141414141414141
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
0x555555756210: 0x4343434343434343      0x4343434343434343
0x555555756220: 0x4343434343434343      0x4343434343434343
0x555555756230: 0x4343434343434343      0x4343434343434343
0x555555756240: 0x4343434343434343      0x4343434343434343
0x555555756250: 0x4343434343434343      0x4343434343434343
0x555555756260: 0x4343434343434343      0x4343434343434343
0x555555756270: 0x4343434343434343      0x4343434343434343
0x555555756280: 0x4343434343434343      0x4343434343434343
0x555555756290: 0x4343434343434343      0x4343434343434343
0x5555557562a0: 0x4343434343434343      0x4343434343434343
0x5555557562b0: 0x4343434343434343      0x4343434343434343
0x5555557562c0: 0x4343434343434343      0x4343434343434343
0x5555557562d0: 0x4343434343434343      0x4343434343434343
0x5555557562e0: 0x4343434343434343      0x4343434343434343
0x5555557562f0: 0x4343434343434343      0x4343434343434343
0x555555756300: 0x4343434343434343      0x0000000000000081
0x555555756310: 0x0000000000000000      0x0000000000000000
```

malloc_chunk

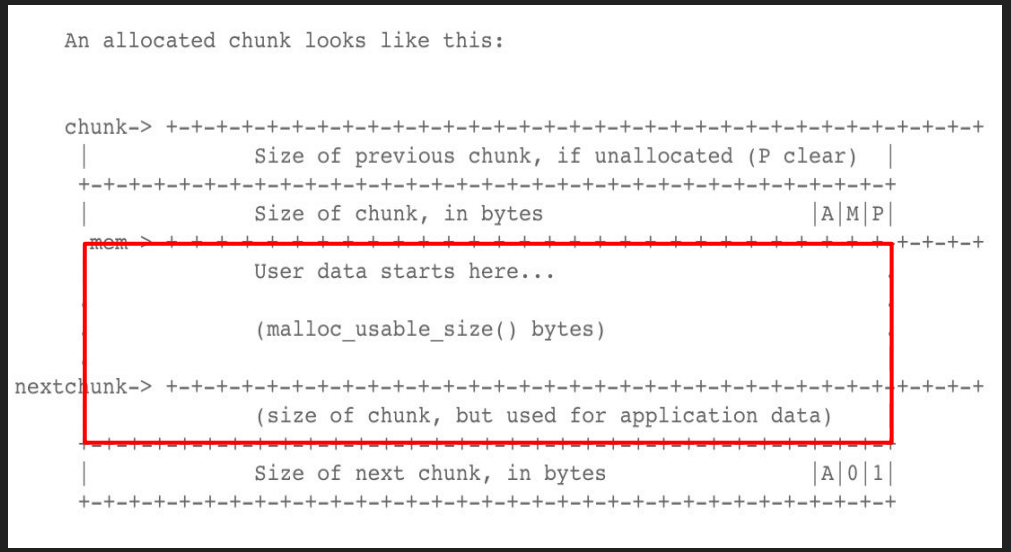
This is a comment from libc

```
    An allocated chunk looks like this:


    chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             Size of previous chunk, if unallocated (P clear)  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             Size of chunk, in bytes                     |A|M|P|
      mem-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             User data starts here...                          .
      .                                                               .
      .             (malloc_usable_size() bytes)                      .
      .                                                               |
nextchunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             (size of chunk, but used for application data)    |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |             Size of next chunk, in bytes                |A|0|1|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

malloc_chunk



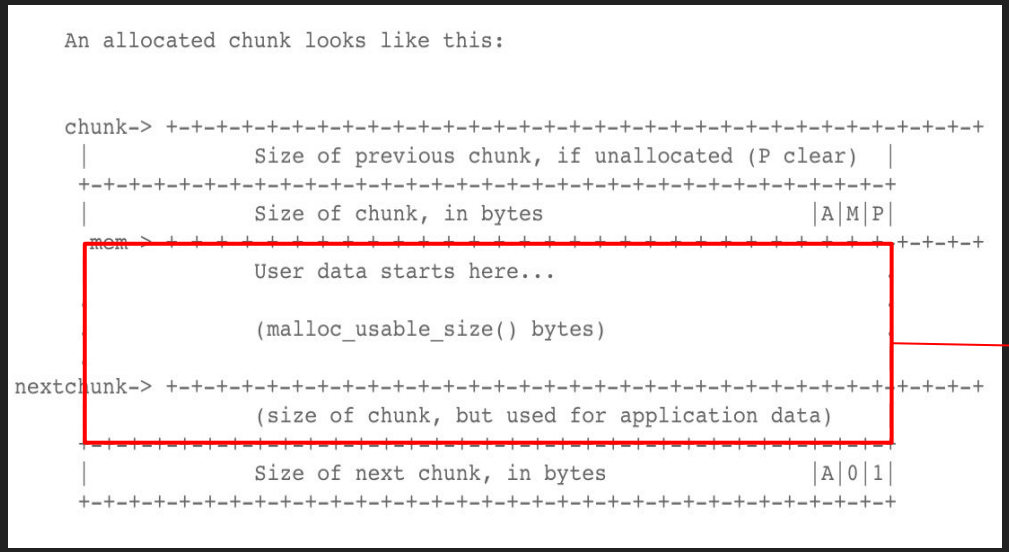
malloc_chunk



This is the current chunk cut out of the debug output so we can analyze it closer

```
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
```

malloc_chunk

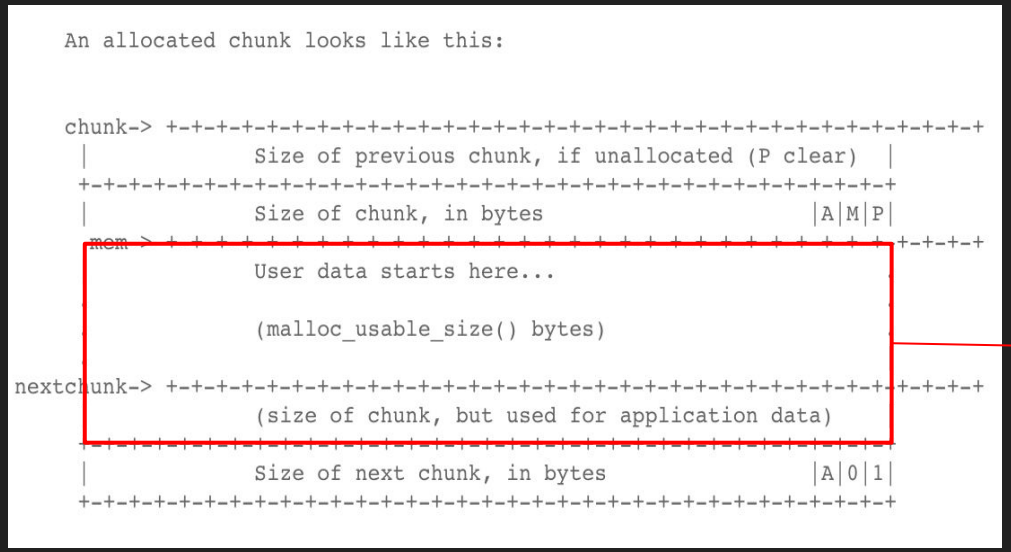


Remember when we memset the buffer to all 0x42 characters?

```
0x555555756100: 0x4141414141414141  0x0000000000000101
0x555555756110: 0x4242424242424242  0x4242424242424242
0x555555756120: 0x4242424242424242  0x4242424242424242
0x555555756130: 0x4242424242424242  0x4242424242424242
0x555555756140: 0x4242424242424242  0x4242424242424242
0x555555756150: 0x4242424242424242  0x4242424242424242
0x555555756160: 0x4242424242424242  0x4242424242424242
0x555555756170: 0x4242424242424242  0x4242424242424242
0x555555756180: 0x4242424242424242  0x4242424242424242
0x555555756190: 0x4242424242424242  0x4242424242424242
0x5555557561a0: 0x4242424242424242  0x4242424242424242
0x5555557561b0: 0x4242424242424242  0x4242424242424242
0x5555557561c0: 0x4242424242424242  0x4242424242424242
0x5555557561d0: 0x4242424242424242  0x4242424242424242
0x5555557561e0: 0x4242424242424242  0x4242424242424242
0x5555557561f0: 0x4242424242424242  0x4242424242424242
0x555555756200: 0x4242424242424242  0x0000000000000101
```

malloc_chunk

Let’s take a look at the size of the chunk now

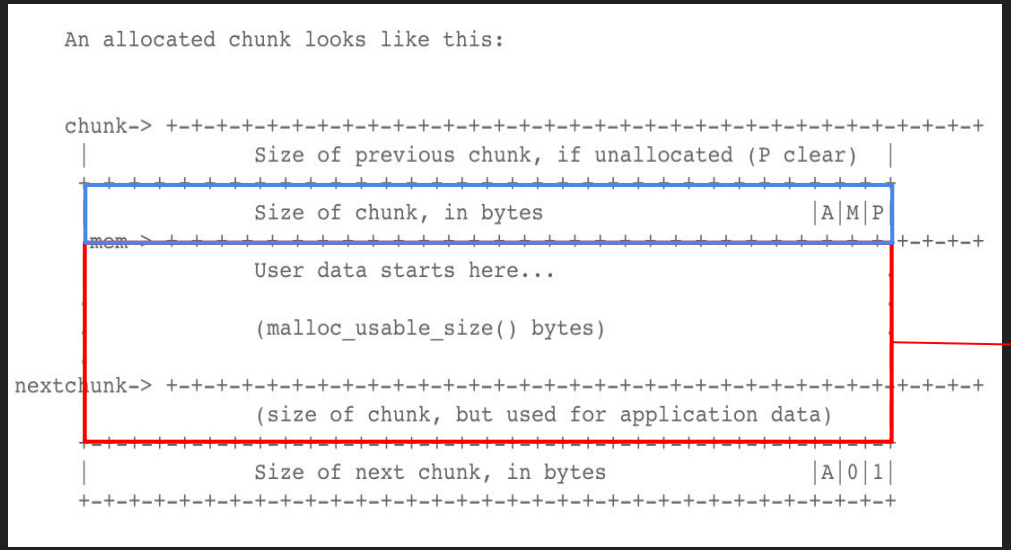



```
void *curr = malloc(0xf8);
```

```
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
```

An allocation of 0xf8 returns a chunksize of?

malloc_chunk

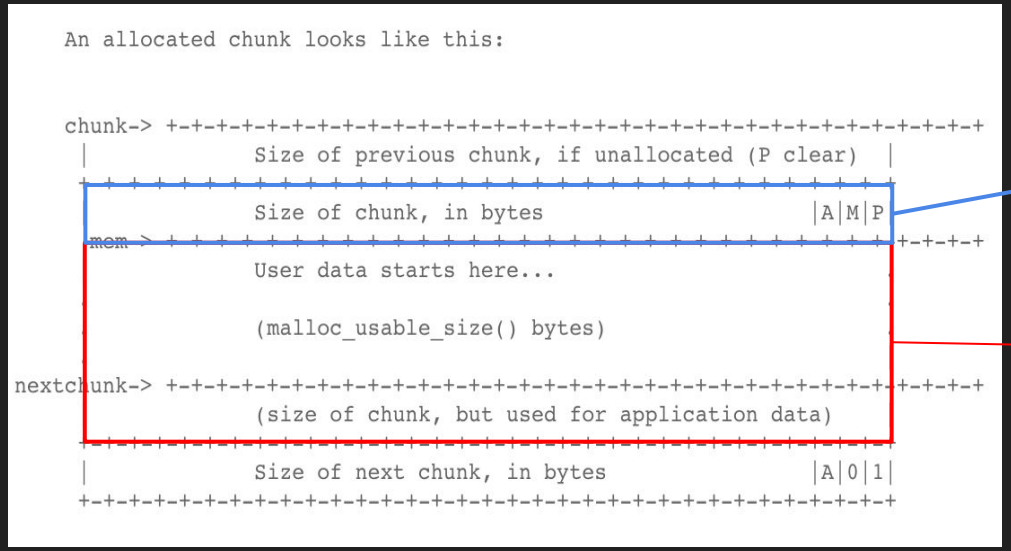



```
void *curr = malloc(0xf8);
```

```
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
```

An allocation of 0xf8 returns a chunksize of?

malloc_chunk



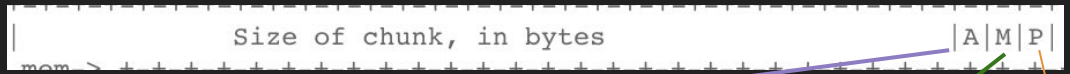

```
void *curr = malloc(0xf8);
```

```
0x555555756100: 0x4141414141414141  0x0000000000000101
0x555555756110: 0x4242424242424242  0x4242424242424242
0x555555756120: 0x4242424242424242  0x4242424242424242
0x555555756130: 0x4242424242424242  0x4242424242424242
0x555555756140: 0x4242424242424242  0x4242424242424242
0x555555756150: 0x4242424242424242  0x4242424242424242
0x555555756160: 0x4242424242424242  0x4242424242424242
0x555555756170: 0x4242424242424242  0x4242424242424242
0x555555756180: 0x4242424242424242  0x4242424242424242
0x555555756190: 0x4242424242424242  0x4242424242424242
0x5555557561a0: 0x4242424242424242  0x4242424242424242
0x5555557561b0: 0x4242424242424242  0x4242424242424242
0x5555557561c0: 0x4242424242424242  0x4242424242424242
0x5555557561d0: 0x4242424242424242  0x4242424242424242
0x5555557561e0: 0x4242424242424242  0x4242424242424242
0x5555557561f0: 0x4242424242424242  0x4242424242424242
0x555555756200: 0x4242424242424242  0x0000000000000101
```

An allocation of 0xf8 returns a chunksize of: 0x101

malloc_chunk

Let’s take a closer look at this size field.



A for ARENA
0: main_arena ptr
1: non-main_arena

There is functionality for multiple arenas to exist at once.

M for Mmapped
0: is not mmapped
1: is mmapped

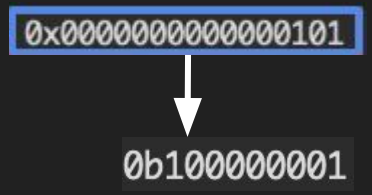
(being mmapped implies that the heap doesn’t need to manage it, and instead to hand it off to the system)

P for PREV_IN_USE
0: prev chunk NOT in use
1: prev chunk in use

PREV here meaning the chunk directly above it in the heap segment, not the chunk referred to by the chunk’s back ptr.

malloc_chunk

Notice that this size field is in hex. We only want the 3 least significant bits.



P is on, indicating that the chunk directly previous to it is in use (which is true, because we just allocated it).

A for ARENA
0: main_arena ptr
1: non-main_arena

There is functionality for multiple arenas to exist at once.

M for Mmapped
0: is not mmapped
1: is mmapped

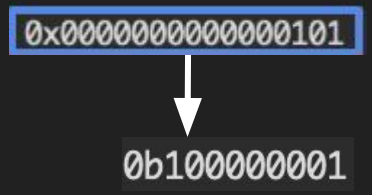
(being mmapped implies that the heap doesn't need to manage it, and instead to hand it off to the system)

P for PREV_IN_USE
0: prev chunk NOT in use
1: prev chunk in use

PREV here meaning the chunk directly above it in the heap segment, not the chunk referred to by the chunk's back ptr.

malloc_chunk

Notice that this size field is in hex. We only want the 3 least significant bits.



M is 0, which is correct because this is definitely NOT an mmapped chunk. (remember we allocated specifically 0xf8 to hit smallbin).

A for ARENA
0: main_arena ptr
1: non-main_arena

There is functionality for multiple arenas to exist at once.

M for Mmapped
0: is not mmapped
1: is mmapped

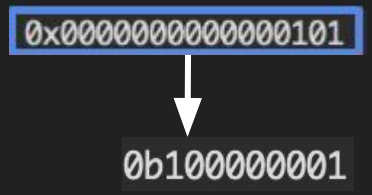
(being mmapped implies that the heap doesn't need to manage it, and instead to hand it off to the system)

P for PREV_IN_USE
0: prev chunk NOT in use
1: prev chunk in use

PREV here meaning the chunk directly above it in the heap segment, not the chunk referred to by the chunk's back ptr.

malloc_chunk

Notice that this size field is in hex. We only want the 3 least significant bits.



A is also 0, which tells us that this chunk still belongs to main_arena.

A for ARENA
0: main_arena ptr
1: non-main_arena

There is functionality for multiple arenas to exist at once.

M for Mmapped
0: is not mmapped
1: is mmapped

(being mmapped implies that the heap doesn't need to manage it, and instead to hand it off to the system)

P for PREV_IN_USE
0: prev chunk NOT in use
1: prev chunk in use

PREV here meaning the chunk directly above it in the heap segment, not the chunk referred to by the chunk's back ptr.

malloc_chunk

```
void *curr = malloc(0xf8);
```

Why was the size 0x100 and not 0xf8?

TL;DR the heap adds 0x8 bytes for the size_sz field, then pads so that the least significant nibble of the next chunk will be 0x0.

This is for alignment's sake.

When you are exploiting and need the size to be a particular number, make sure to subtract 0x8 before requesting the allocation.

This is the conversion at the top of _int_malloc in libc.

```
/*
   Convert request size to internal form by adding SIZE_SZ bytes
   overhead plus possibly more to obtain necessary alignment and/or
   to obtain a size of at least MINSIZE, the smallest allocatable
   size. Also, checked_request2size traps (returning 0) request sizes
   that are so large that they wrap around zero when padded and
   aligned.
 */

checked_request2size (bytes, nb);
```

```
/* pad request bytes into a usable size -- internal version */

#define request2size(req)                                         \
  (((req) + SIZE_SZ + MALLOC_ALIGN_MASK < MINSIZE)  ?             \
   MINSIZE :                                                      \
   ((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK)

/* Same, except also perform an argument and result check.  First, we check
   that the padding done by request2size didn't result in an integer
   overflow.  Then we check (using REQUEST_OUT_OF_RANGE) that the resulting
   size isn't so large that a later alignment would lead to another integer
   overflow.  */
#define checked_request2size(req, sz) \
({                                    \
  (sz) = request2size (req);          \
  if (((sz) < (req))                  \
      || REQUEST_OUT_OF_RANGE (sz)) \
    {                                 \
      __set_errno (ENOMEM);           \
      return 0;                       \
    }                                 \
})
```

I also pulled the actual fn

Back to the action

Now I’m going to free the current chunk.

These things will change in the current chunk:

FD ptr
BK ptr

These things will change in the next chunk:

prev_size field
PREV_IN_USE bit

```
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x4242424242424242      0x4242424242424242
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x4242424242424242      0x0000000000000101
```

```
free(curr);
```

Back to the action

Now I’m going to free the current chunk.

These things will change in the current chunk:

FD ptr
BK ptr

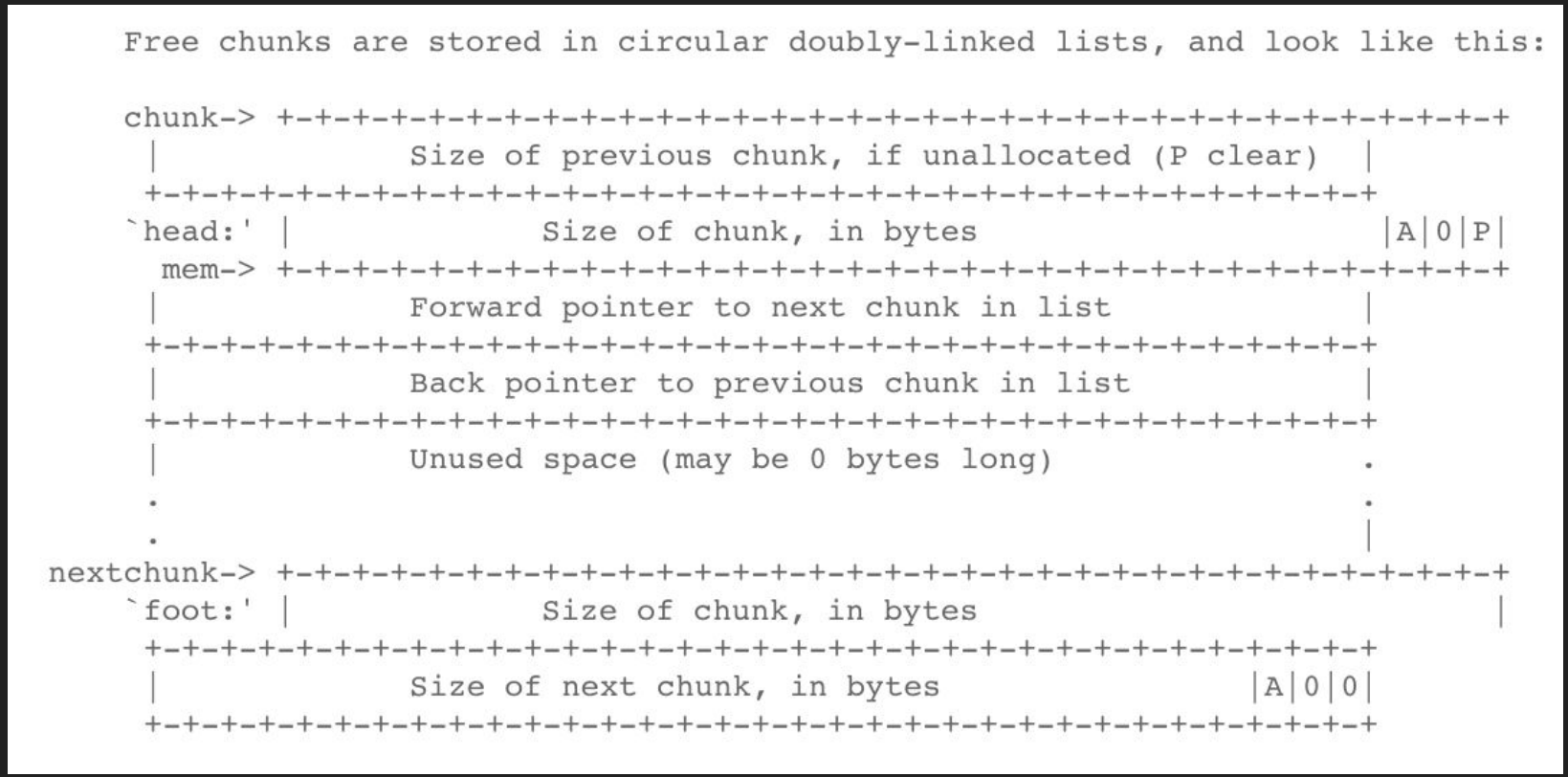
These things will change in the next chunk:

prev_size field
PREV_IN_USE bit

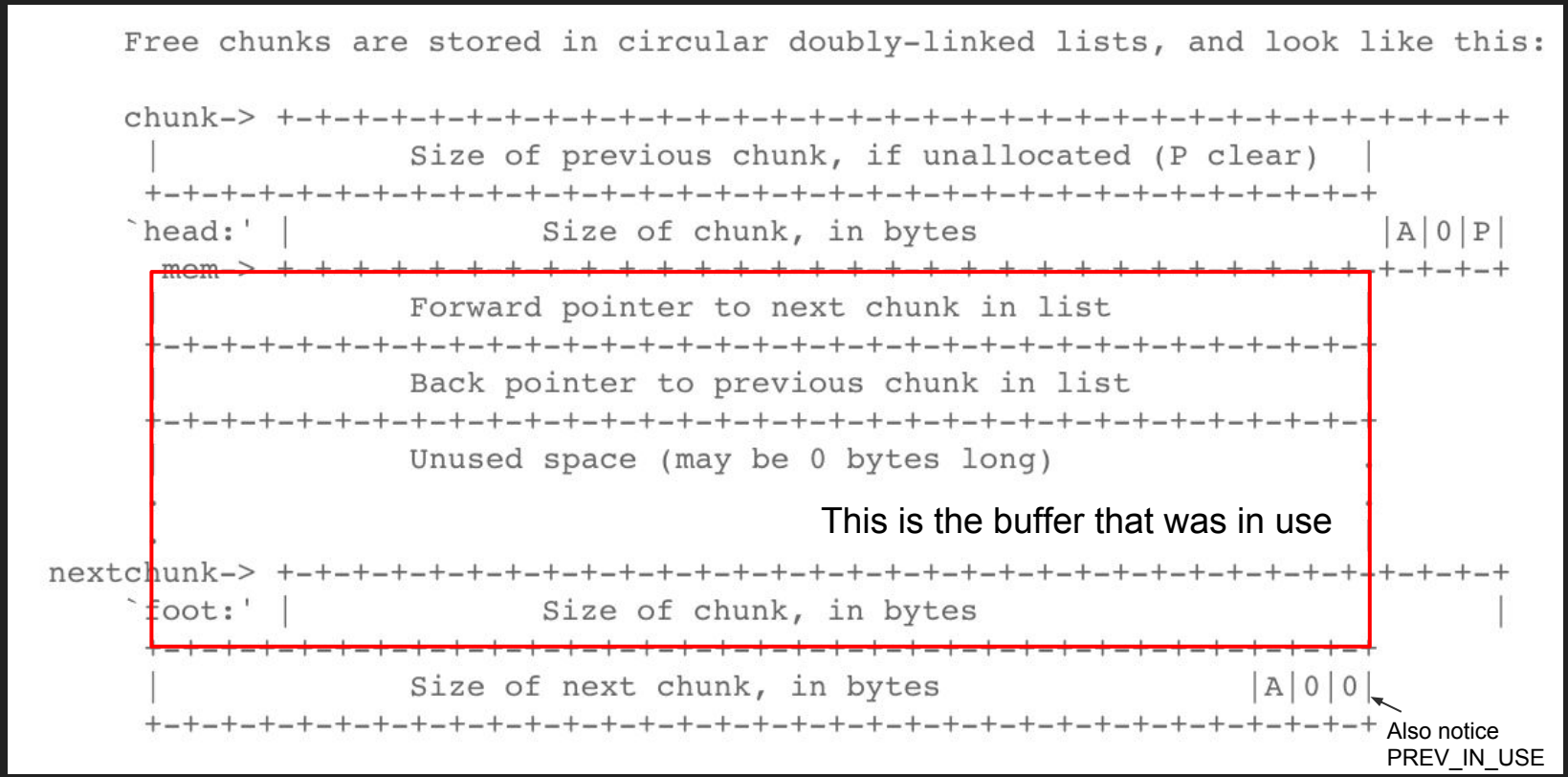
```
0x555555756100: 0x4141414141414141      0x0000000000000101
0x555555756110: 0x00007ffff7dd1b78      0x00007ffff7dd1b78
0x555555756120: 0x4242424242424242      0x4242424242424242
0x555555756130: 0x4242424242424242      0x4242424242424242
0x555555756140: 0x4242424242424242      0x4242424242424242
0x555555756150: 0x4242424242424242      0x4242424242424242
0x555555756160: 0x4242424242424242      0x4242424242424242
0x555555756170: 0x4242424242424242      0x4242424242424242
0x555555756180: 0x4242424242424242      0x4242424242424242
0x555555756190: 0x4242424242424242      0x4242424242424242
0x5555557561a0: 0x4242424242424242      0x4242424242424242
0x5555557561b0: 0x4242424242424242      0x4242424242424242
0x5555557561c0: 0x4242424242424242      0x4242424242424242
0x5555557561d0: 0x4242424242424242      0x4242424242424242
0x5555557561e0: 0x4242424242424242      0x4242424242424242
0x5555557561f0: 0x4242424242424242      0x4242424242424242
0x555555756200: 0x0000000000000100      0x0000000000000100
```

```
free(curr);
```

malloc_chunk



malloc_chunk



malloc_chunk

```
struct malloc_chunk {

  INTERNAL_SIZE_T      mchunk_prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      mchunk_size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

malloc_chunk

Malloc will return ptr here


```
struct malloc_chunk {

  INTERNAL_SIZE_T      mchunk_prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      mchunk_size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;                /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

mchunk_prev_size of next chunk

malloc_state

This structure holds the state of an Arena

The main thread's arena is a global called

`main_arena`

Other thread's arenas are stored in the heap itself. Non-main_arenas can have multiple malloc_state structs assigned to them.

```
/*
   have_fastchunks indicates that there are probably some fastbin chunks.
   It is set true on entering a chunk into any fastbin, and cleared early in
   malloc_consolidate.  The value is approximate since it may be set when there
   are no fastbin chunks, or it may be clear even if there are fastbin chunks
   available.  Given it's sole purpose is to reduce number of redundant calls to
   malloc_consolidate, it does not affect correctness.  As a result we can safely
   use relaxed atomic accesses.
 */


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 fastbinsY[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];

  /* Bitmap of bins */
  unsigned int binmap[BINMAPSIZE];

  /* Linked list */
  struct malloc_state *next;

  /* Linked list for free arenas.  Access to this field is serialized
     by free_list_lock in arena.c.  */
  struct malloc_state *next_free;

  /* Number of threads attached to this arena.  0 if the arena is on
     the free list.  Access to this field is serialized by
     free_list_lock in arena.c.  */
  INTERNAL_SIZE_T attached_threads;

  /* Memory allocated from the system in this arena.  */
  INTERNAL_SIZE_T system_mem;
  INTERNAL_SIZE_T max_system_mem;
};
```

Bins and Chunks

A bin is a linked-list (single for fast, double for others) of free chunks.

They are differentiated based on size:

Fastbin	0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80	10 of these
Smallbin	0x20, 0x30, 0x40, … 0x1f8	62 of these
Largebin	0x200 - 0x100000	63 of these
mmap’d	<= 0x100000	----------

Bins and Chunks

There is also one SPECIAL bin: the Unsorted bin.

This holds anything that just got freed, as well as whatever is left over when it chops up a chunk

```
/*
   Unsorted chunks

    All remainders from chunk splits, as well as all returned chunks,
    are first placed in the "unsorted" bin. They are then placed
    in regular bins after malloc gives them ONE chance to be used before
    binning. So, basically, the unsorted_chunks list acts as a queue,
    with chunks being placed on it in free (and malloc_consolidate),
    and taken off (to be either used or placed in bins) in malloc.

    The NON_MAIN_ARENA flag is never set for unsorted chunks, so it
    does not have to be taken into account in size comparisons.
 */
```

Fastbins

These are special because they only use singly-linked lists.

They do not consolidate with chunks around them until malloc_consolidate is called, which then consolidates ALL freed fastbins at once.

The back pointer field in these chunks is not touched by the heap.

```
/*
   Fastbins

    An array of lists holding recently freed small chunks.  Fastbins
    are not doubly linked.  It is faster to single-link them, and
    since chunks are never removed from the middles of these lists,
    double linking is not necessary. Also, unlike regular bins, they
    are not even processed in FIFO order (they use faster LIFO) since
    ordering doesn't much matter in the transient contexts in which
    fastbins are normally used.

    Chunks in fastbins keep their inuse bit set, so they cannot
    be consolidated with other free chunks. malloc_consolidate
    releases all chunks in fastbins and consolidates them with
    other free chunks.
 */
```

Basic Techniques

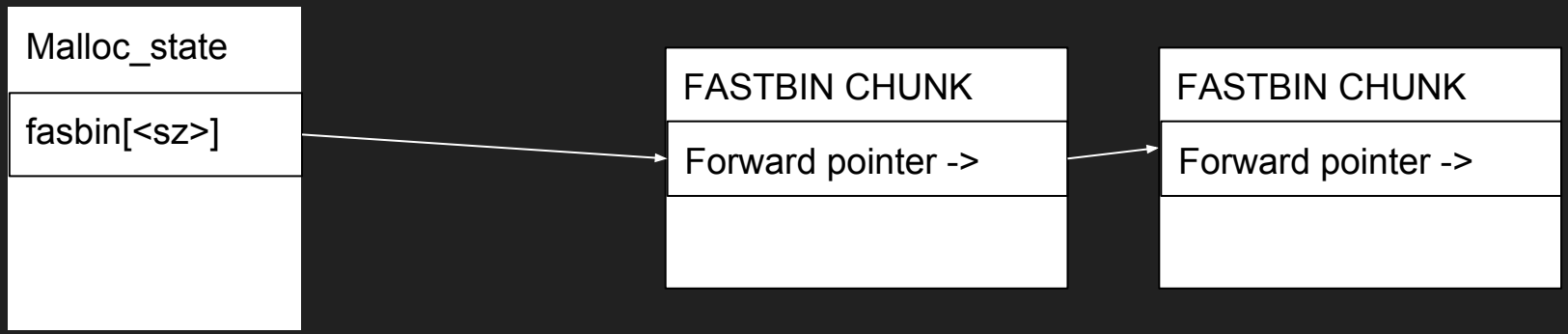
Ok it’s finally time to go over some basic heap exploitation techniques!

fastbin_dup

fastbin_to_arbitrary_ptr

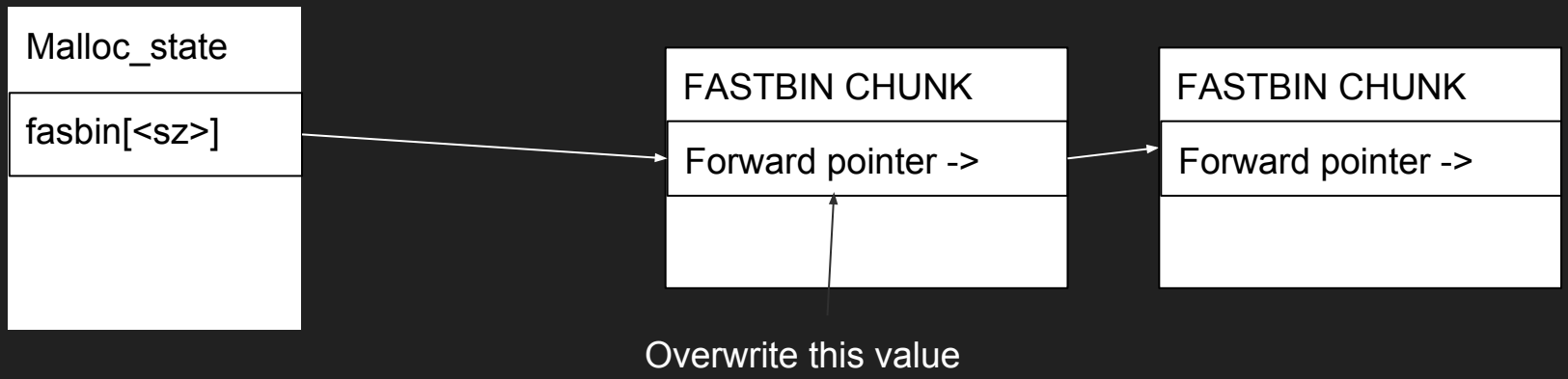
fastbin_to_arbitrary_ptr

This exploit relies on changing the contents of a chunk that is already on a fastbin freelist. It requires the attacker to be able to overwrite the FD pointer of a freed chunk, either through some use-after-free vulnerability or an arbitrary overwrite of the chunk above it.



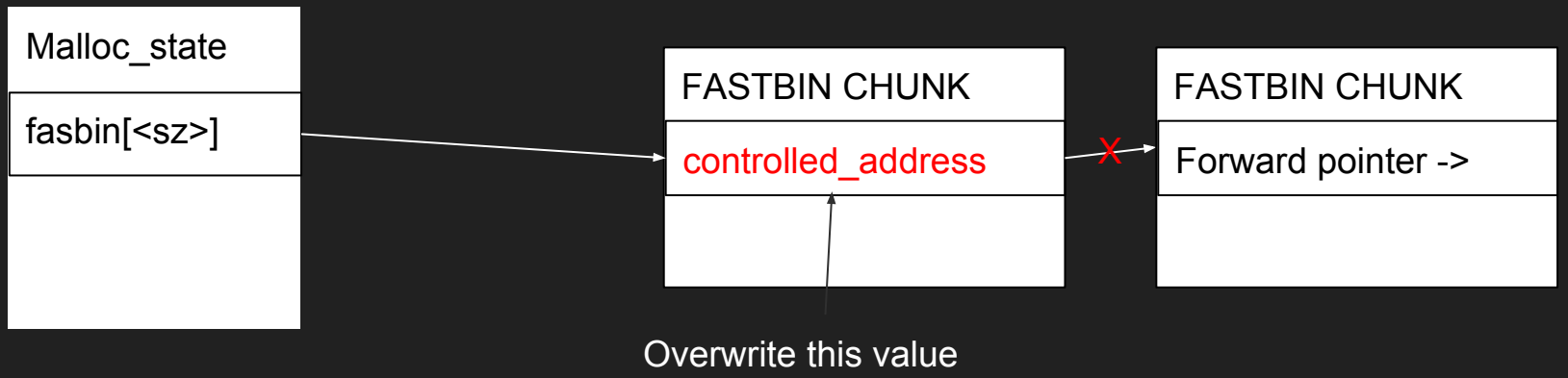
fastbin_to_arbitrary_ptr

This exploit relies on changing the contents of a chunk that is already on a fastbin freelist. It requires the attacker to be able to overwrite the FD pointer of a freed chunk, either through some use-after-free vulnerability or an arbitrary overwrite of the chunk above it.



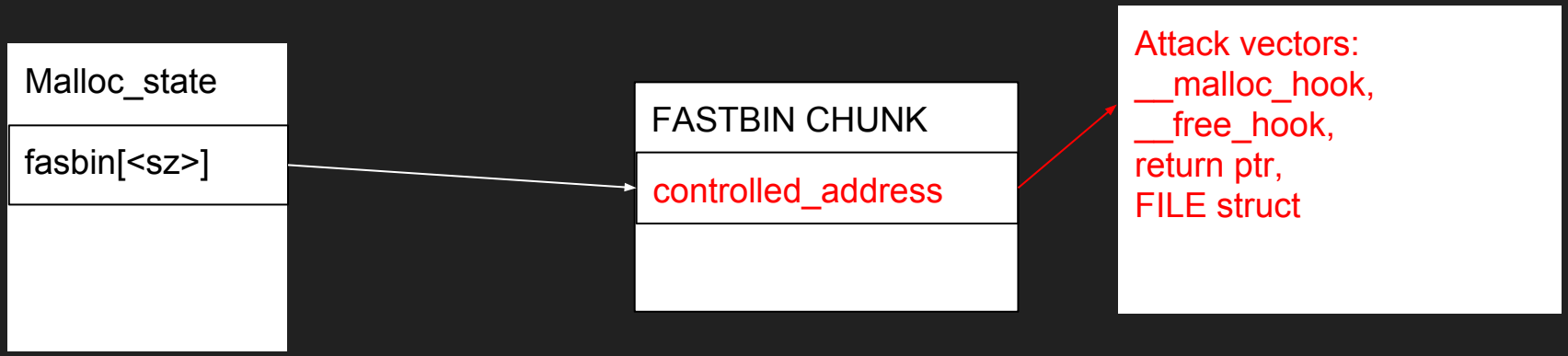
fastbin_to_arbitrary_ptr

This exploit relies on changing the contents of a chunk that is already on a fastbin freelist. It requires the attacker to be able to overwrite the FD pointer of a freed chunk, either through some use-after-free vulnerability or an arbitrary overwrite of the chunk above it.



fastbin_to_arbitrary_ptr

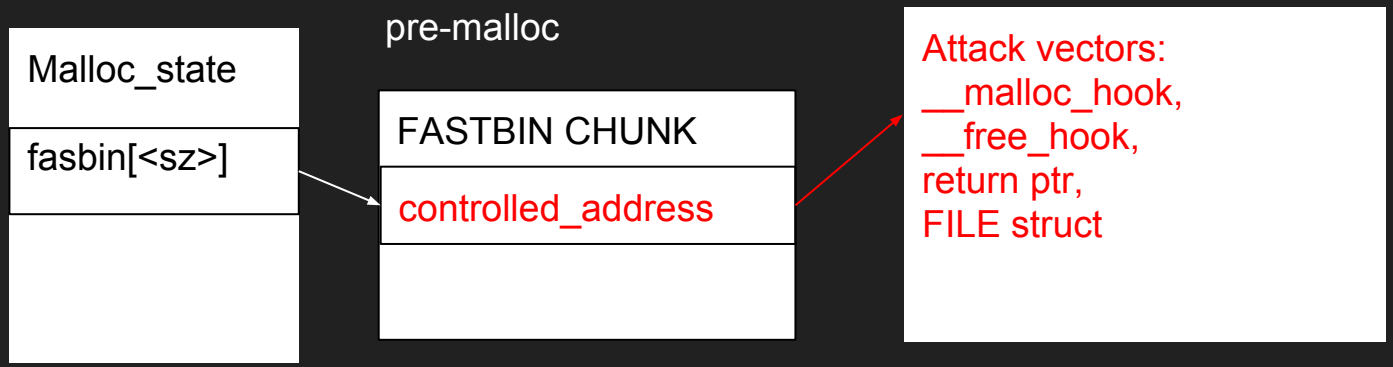
This exploit relies on changing the contents of a chunk that is already on a fastbin freelist. It requires the attacker to be able to overwrite the FD pointer of a freed chunk, either through some use-after-free vulnerability or an arbitrary overwrite of the chunk above it.



fastbin_to_arbitrary_ptr

This exploit relies on changing the contents of a chunk that is already on a fastbin freelist. It requires the attacker to be able to overwrite the FD pointer of a freed chunk, either through some use-after-free vulnerability or an arbitrary overwrite of the chunk above it.

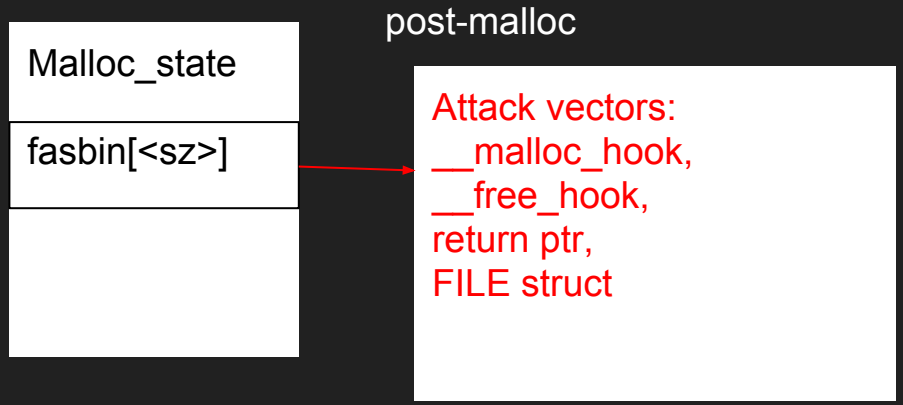
If we malloc(sz - 0x8) now, the fastbin will return the overflowed chunk.



fastbin_to_arbitrary_ptr

This exploit relies on changing the contents of a chunk that is already on a fastbin freelist. It requires the attacker to be able to overwrite the FD pointer of a freed chunk, either through some use-after-free vulnerability or an arbitrary overwrite of the chunk above it.

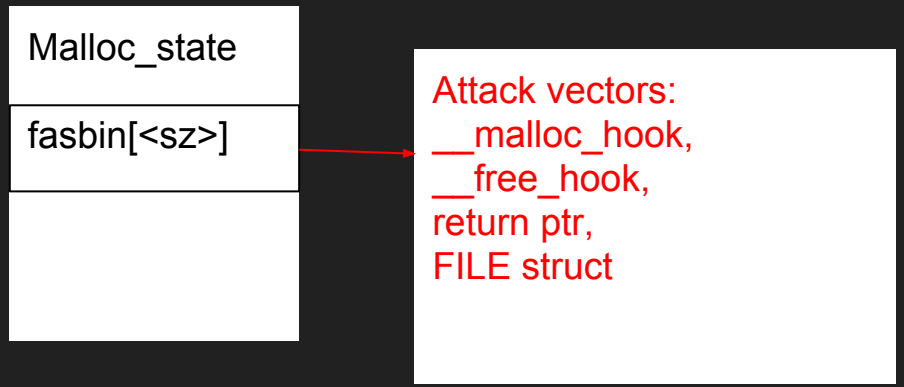
Now the top of the freelist holds the value we overwrote the FD pointer with.



fastbin_to_arbitrary_ptr

Once the chunk being exploited is overwritten, because of how fastbin works, once the overwritten chunk becomes top of the freelist, we can malloc again to set the bin to point at our arbitrary pointer.

We malloc again to receive a pointer to the attack vector



fastbin_dup

This exploit relies on the existence of a double free - where the program frees the same pointer more than once.

There exists a security check for this - when freeing fastbins, free will check if the top of the freelist is identical to the one being freed.

This prevents silly bugs such as this one →

However does not prevent a malicious attacker from using a non-nulled pointer as an exploitable vulnerability.

```
free(curr);

/* ... */

free(curr);
```

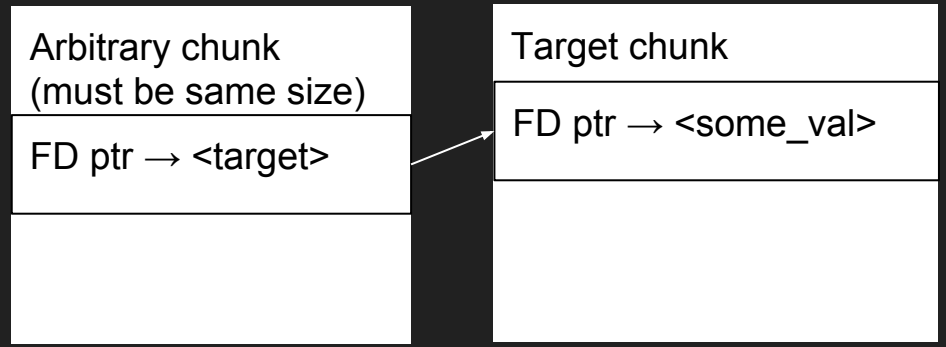
fastbin_dup

This exploit requires that the attacker has the ability to line up 3 frees in a row.

The first on the target chunk, the second on an arbitrary chunk of the same size, and the third on the target chunk once more (hence the *double* in double free).

This bypasses the security check for fastbins.

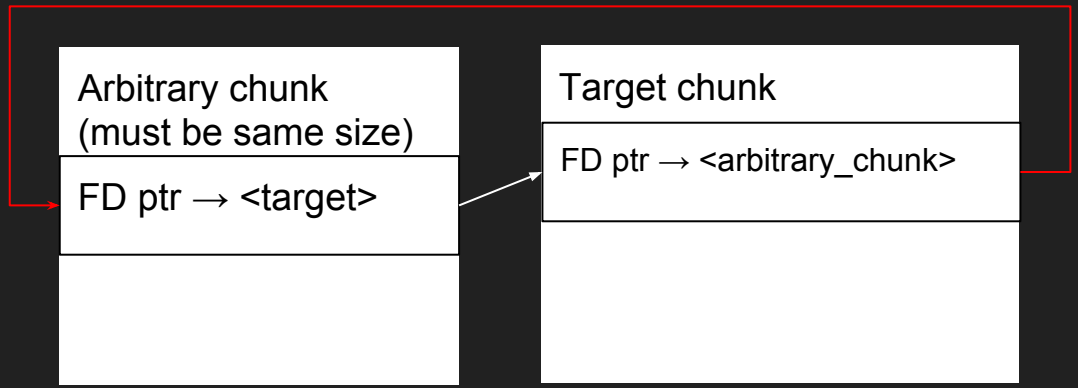
```
free(target);
free(arb);
```
WE ARE HERE
```
free(target);
```



fastbin_dup

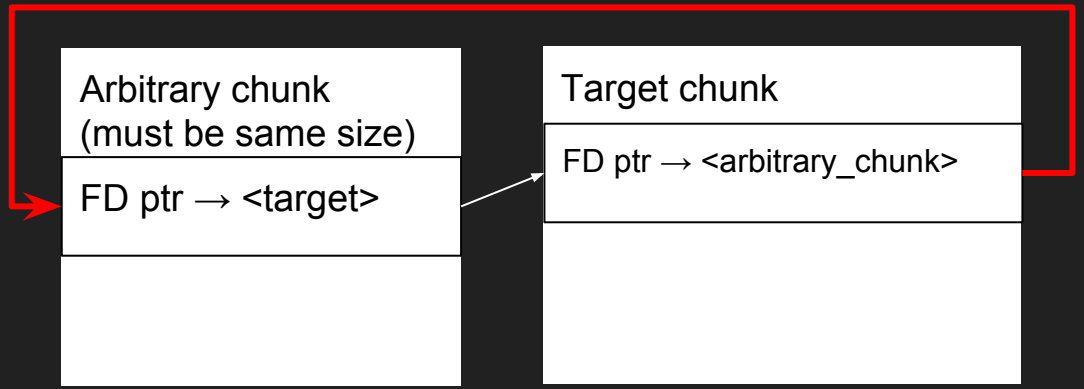
```
free(target);
free(arb);

free(target);
```

When we free the target chunk again, we write a new FD ptr to the chunk and set top of the fastbin to point at target chunk.



fastbin_dup

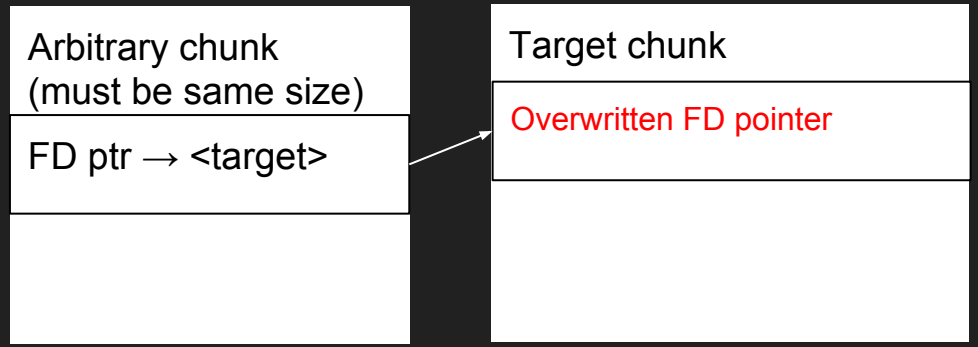
This is useful when exploiting because programs usually only allow for reading into malloc'd space.



fastbin_dup

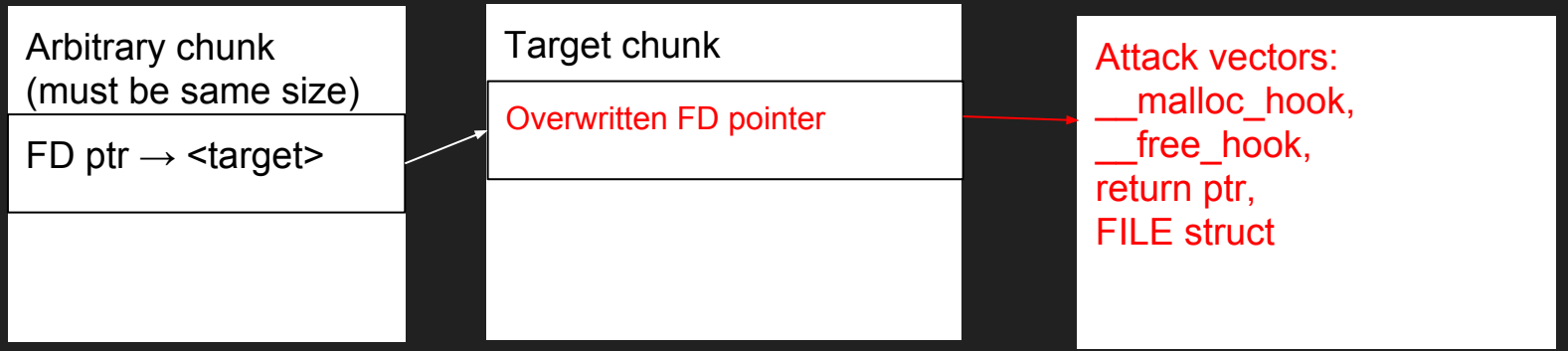
We can convince the program to write to a freed chunk now by malloc'ing again.

This will return the target chunk, and allow us to write into it.



fastbin_dup

This allows us to chain fastbin_dup to a fastbin_to_arbitrary_ptr!



Leon Chou
OSIRIS Lab
NYUSEC

References:

https://heap-exploitation.dhavalkapil.com/

https://github.com/shellphish/how2heap

Good resources to learn more:

http://blog.angelboy.tw/

Related exploits:

https://dhavalkapil.com/blogs/FILE-Structure-Exploitation/