Machbooks

WCTF 2020

Balsn

@_how2hack_

Outline

- Challenge info
- Vulnerability
- Exploit

Machbooks

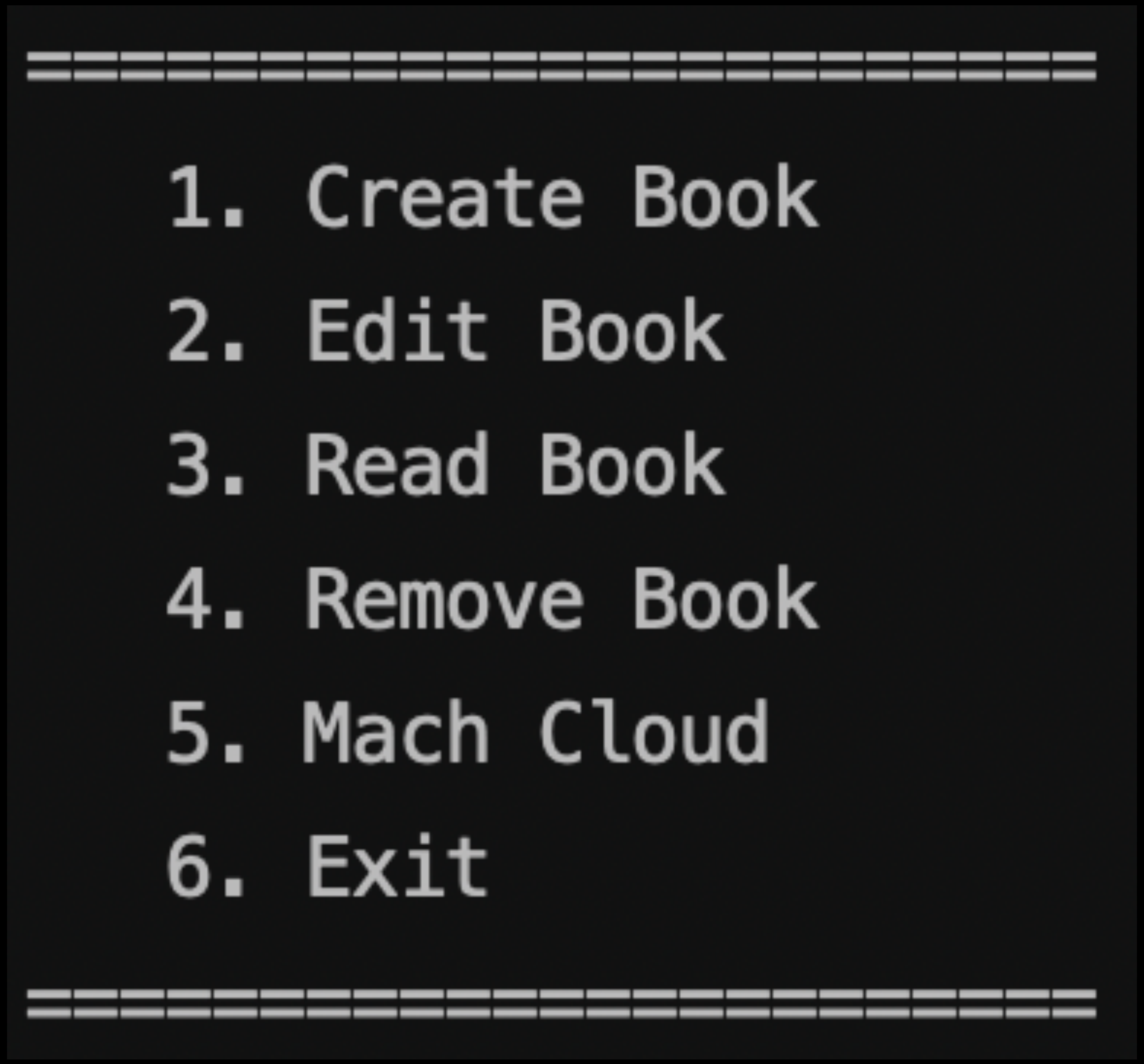
- Pure Pwn challenge
- Mach-O 64-bit executable x86_64
- macOS Catalina 10.15.7

Machbooks

- machbooks.sb
  - Flag path: /Users/ctf/machbooks/flag (relative path ../../flag)
  - Can read write on tmp directory
  - Can only execute the binary machbooks
    - Can’t get shell
    - Final exploit should be ORW

Machbooks

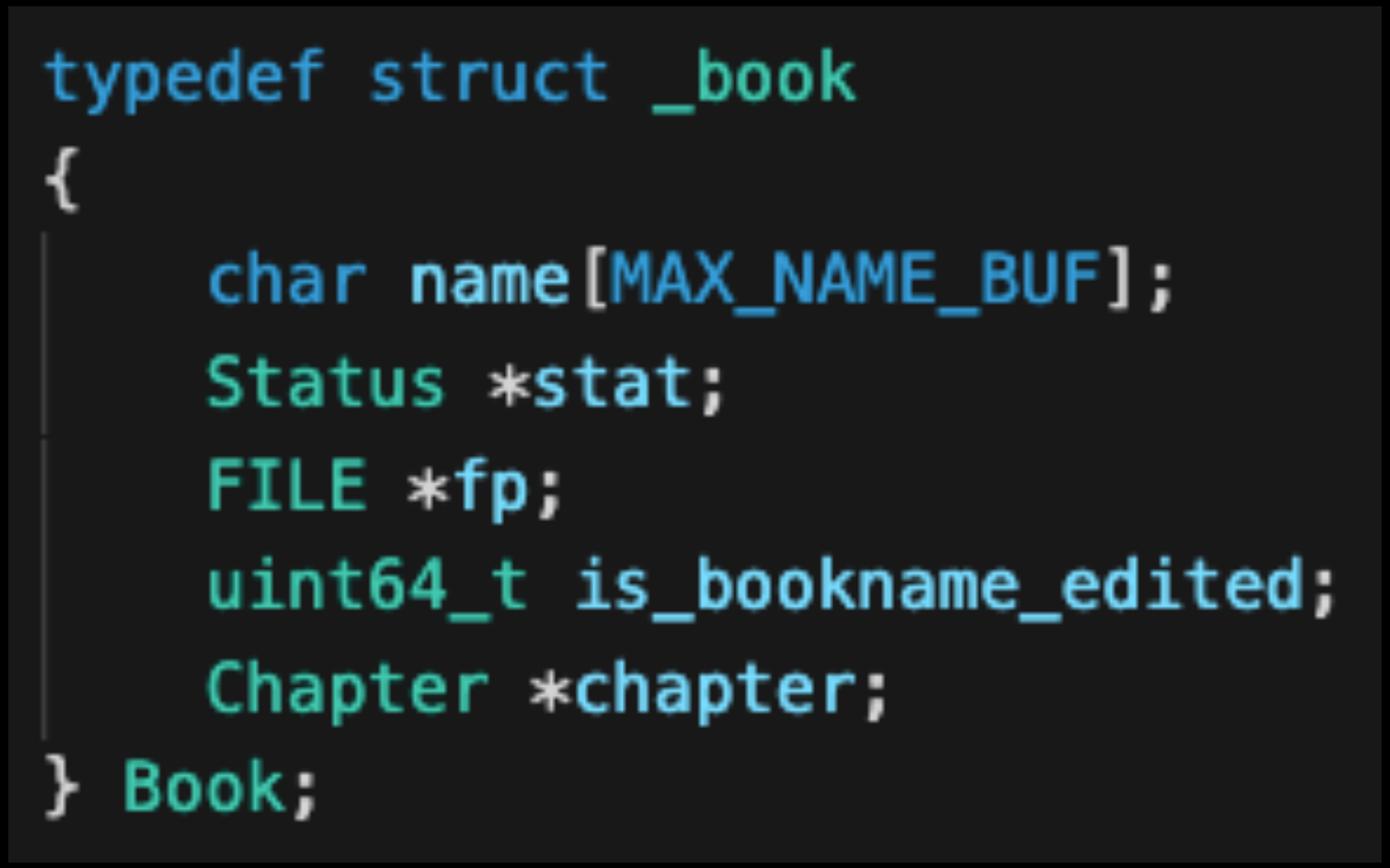
- Create / Read / Remove books
- Edit book name & Add new chapter to book
- Store/Load books to/from “cloud”



```
==========================

   1. Create Book
   2. Edit Book
   3. Read Book
   4. Remove Book
   5. Mach Cloud
   6. Exit

==========================
```

Machbooks

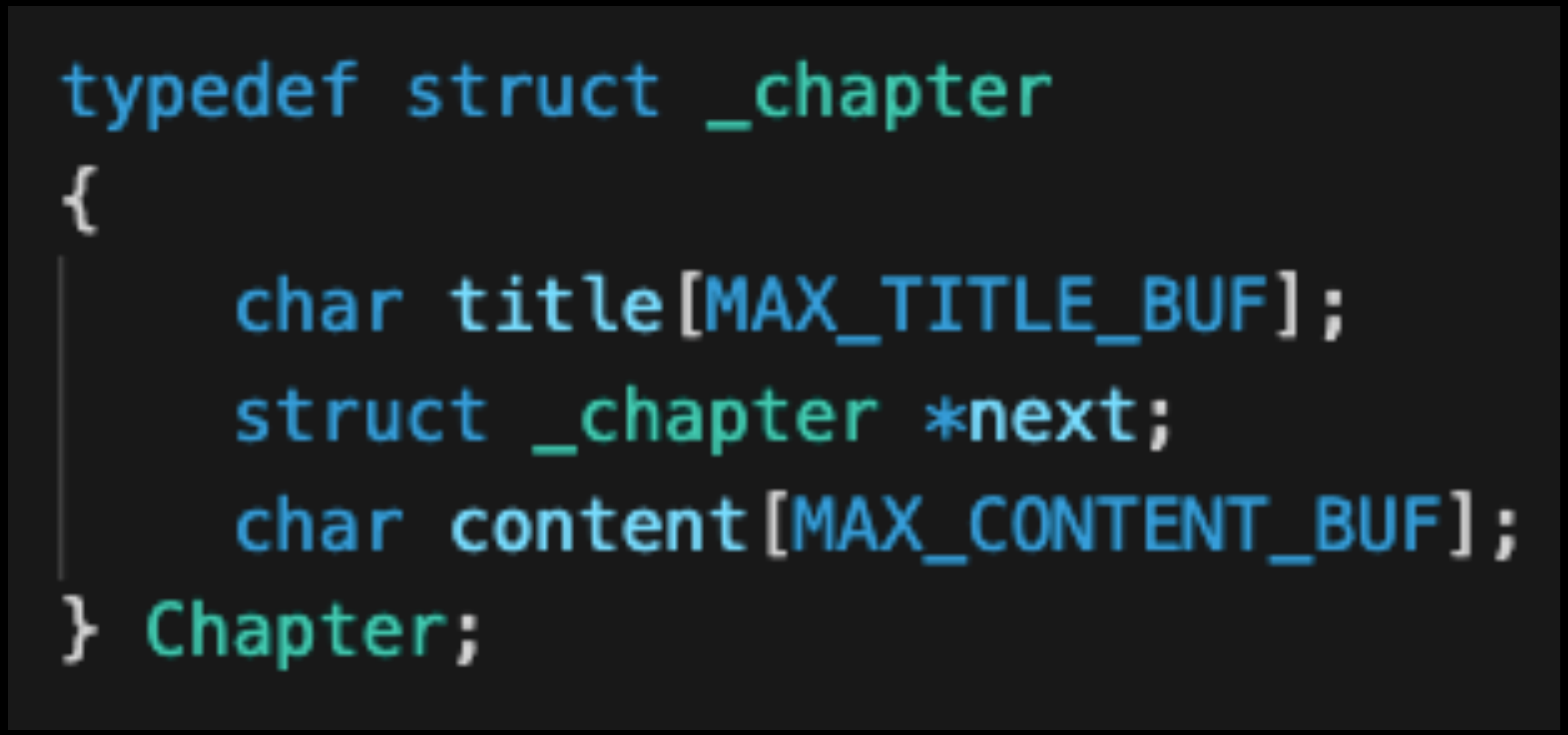
- Book
  - MAX_NAME_BUF = 0x20
  - Can only edit bookname once
  - fp is used only when book is loaded from cloud



```c
typedef struct _book
{
    char name[MAX_NAME_BUF];
    Status *stat;
    FILE *fp;
    uint64_t is_bookname_edited;
    Chapter *chapter;
} Book;
```

Machbooks

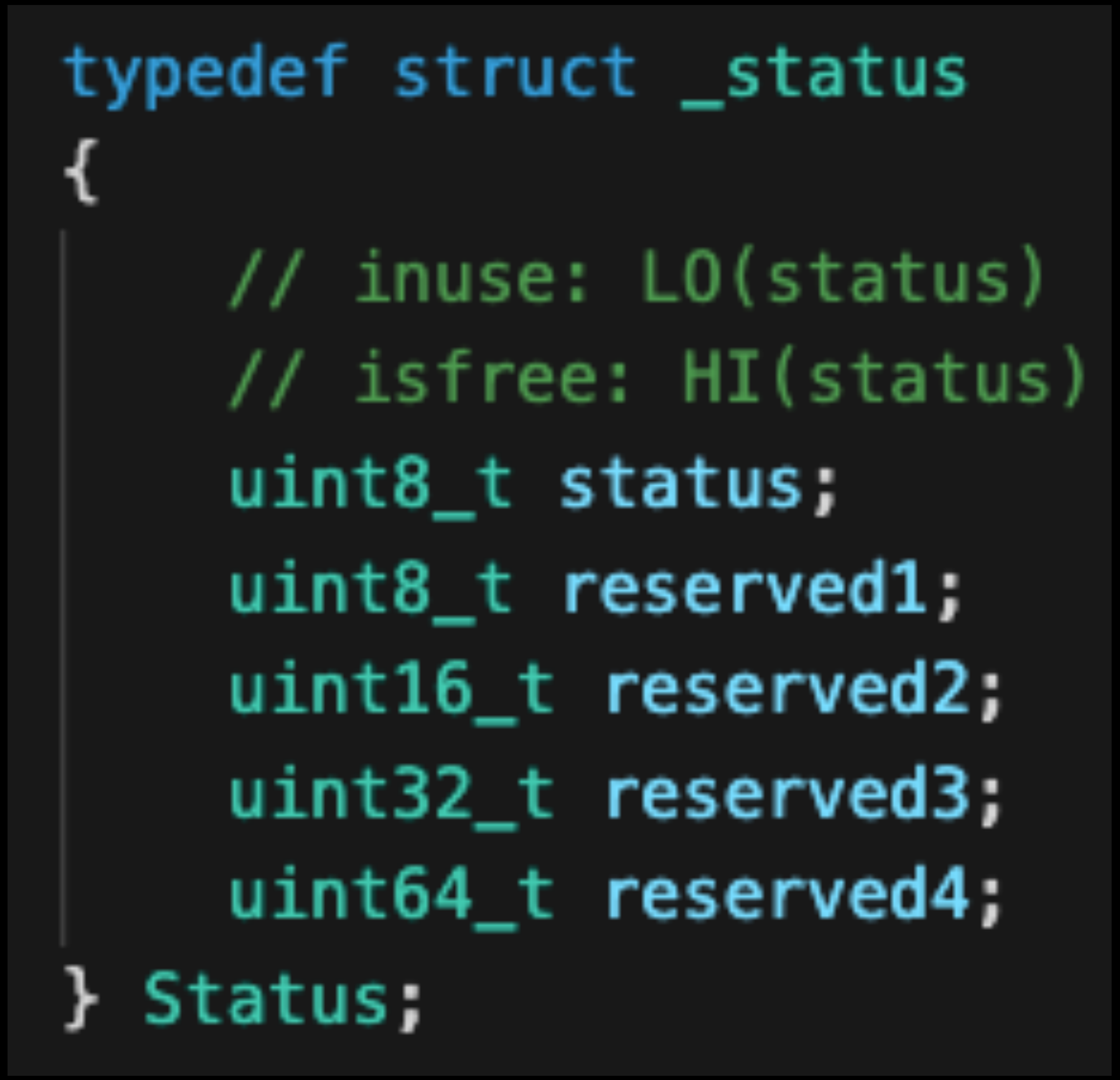
- Chapter [0x528] (Small Heap)
  - MAX_TITLE_BUF = 0x20
  - MAX_CONTENT_BUF = 0x500



```c
typedef struct _chapter
{
    char title[MAX_TITLE_BUF];
    struct _chapter *next;
    char content[MAX_CONTENT_BUF];
} Chapter;
```

Machbooks

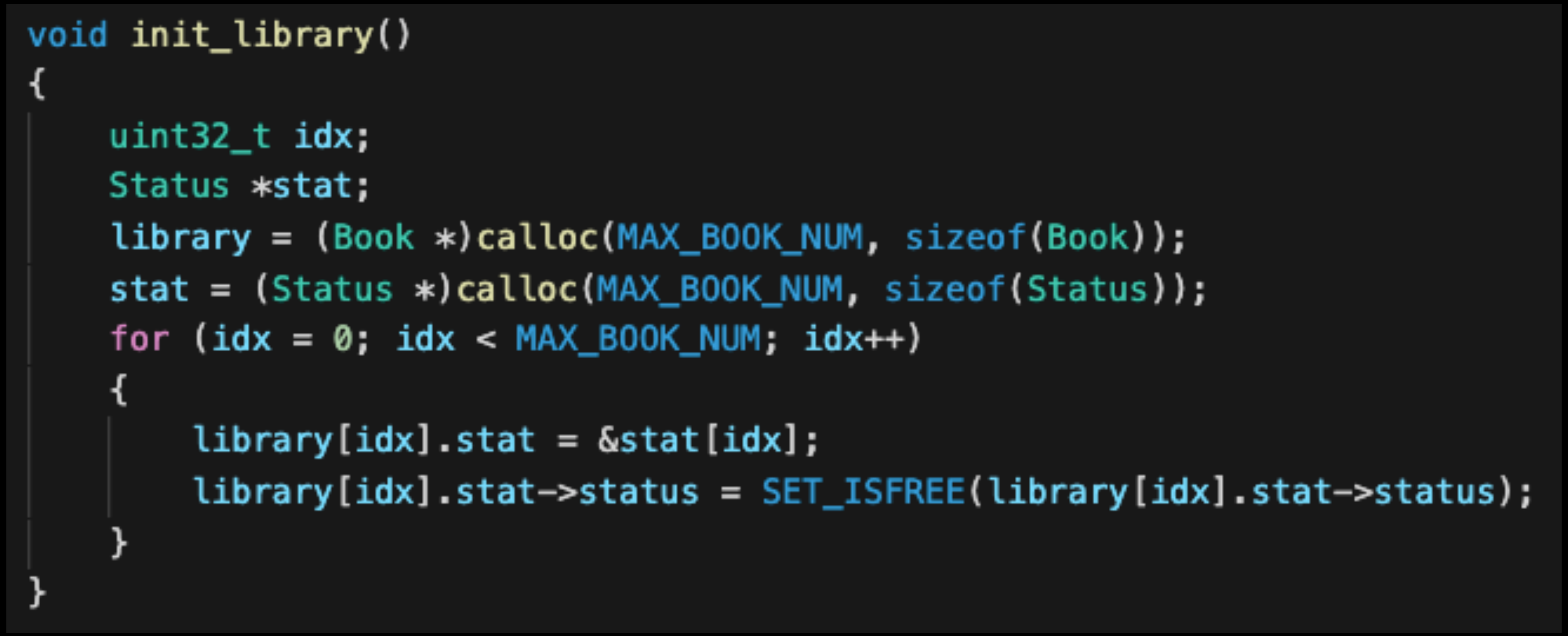
- Status
  - indicate inuse and isfree of a Book
  - inuse bit = LO(status)
  - isfree bit = HI(status)
  - 0x10 aligned



```c
typedef struct _status
{
    // inuse: LO(status)
    // isfree: HI(status)
    uint8_t status;
    uint8_t reserved1;
    uint16_t reserved2;
    uint32_t reserved3;
    uint64_t reserved4;
} Status;
```

Machbooks

- Initialization
  - Allocate [MAX_BOOK_NUM = 6] Book (Tiny Heap)
  - Allocate [MAX_BOOK_NUM = 6] Status (Tiny Heap)
  - Assign book[i].status to &stat[idx] and set isfree bit



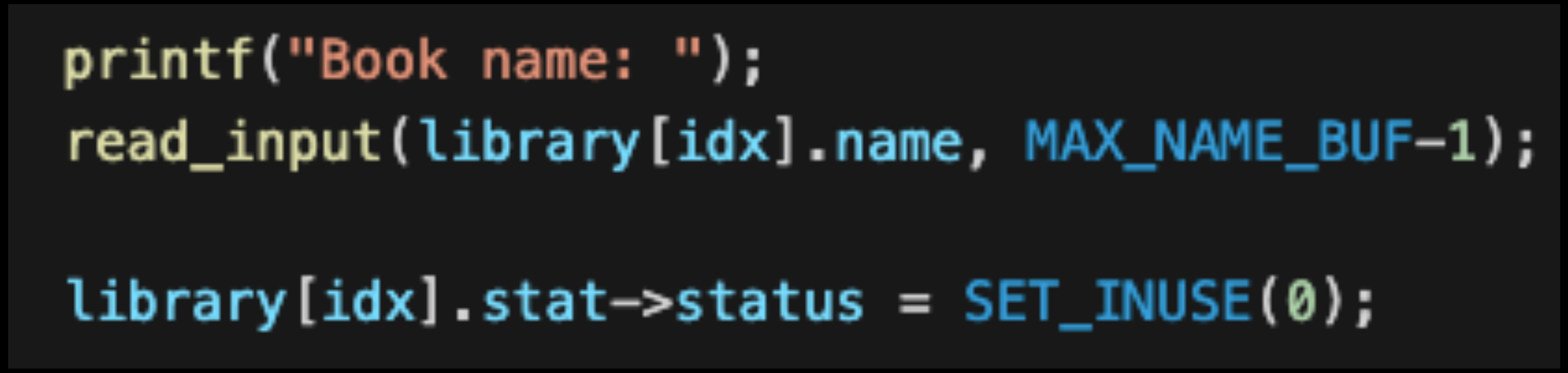
```
void init_library()
{
    uint32_t idx;
    Status *stat;
    library = (Book *)calloc(MAX_BOOK_NUM, sizeof(Book));
    stat = (Status *)calloc(MAX_BOOK_NUM, sizeof(Status));
    for (idx = 0; idx < MAX_BOOK_NUM; idx++)
    {
        library[idx].stat = &stat[idx];
        library[idx].stat->status = SET_ISFREE(library[idx].stat->status);
    }
}
```

Outline

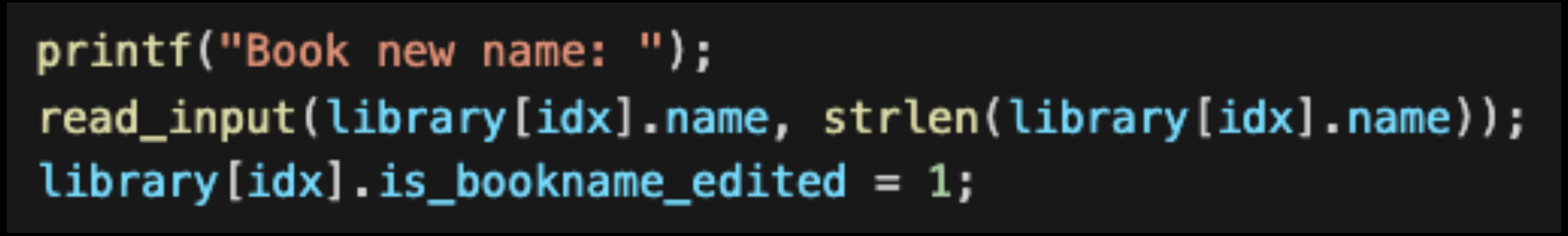
- Challenge info
- Vulnerability
- Exploit

Vulnerability

- Overflow in **Edit Bookname** -> Information Leak
  - **Edit Bookname** uses strlen() to determine the read size
  - **Create Book** only read (MAX_NAME_BUF - 1) length
  - However, **Load Book** uses fgets(), so newline will be appended to buf
  - Leak status address (tiny heap address)



```
printf("Book name: ");
read_input(library[idx].name, MAX_NAME_BUF-1);

library[idx].stat->status = SET_INUSE(0);
```



```
printf("Book new name: ");
read_input(library[idx].name, strlen(library[idx].name));
library[idx].is_bookname_edited = 1;
```

Vulnerability

- Overflow in **Edit Bookname** -> Selective HI bit or LO bit overwrite
  - Overwrite status to any address
  - If LO(*address) == 0
    - Create a book so that LO(*address) == inuse bit is set
  - If HI(*address) == 0
    - Remove a book so that HI(*address) == isfree bit is set

Outline

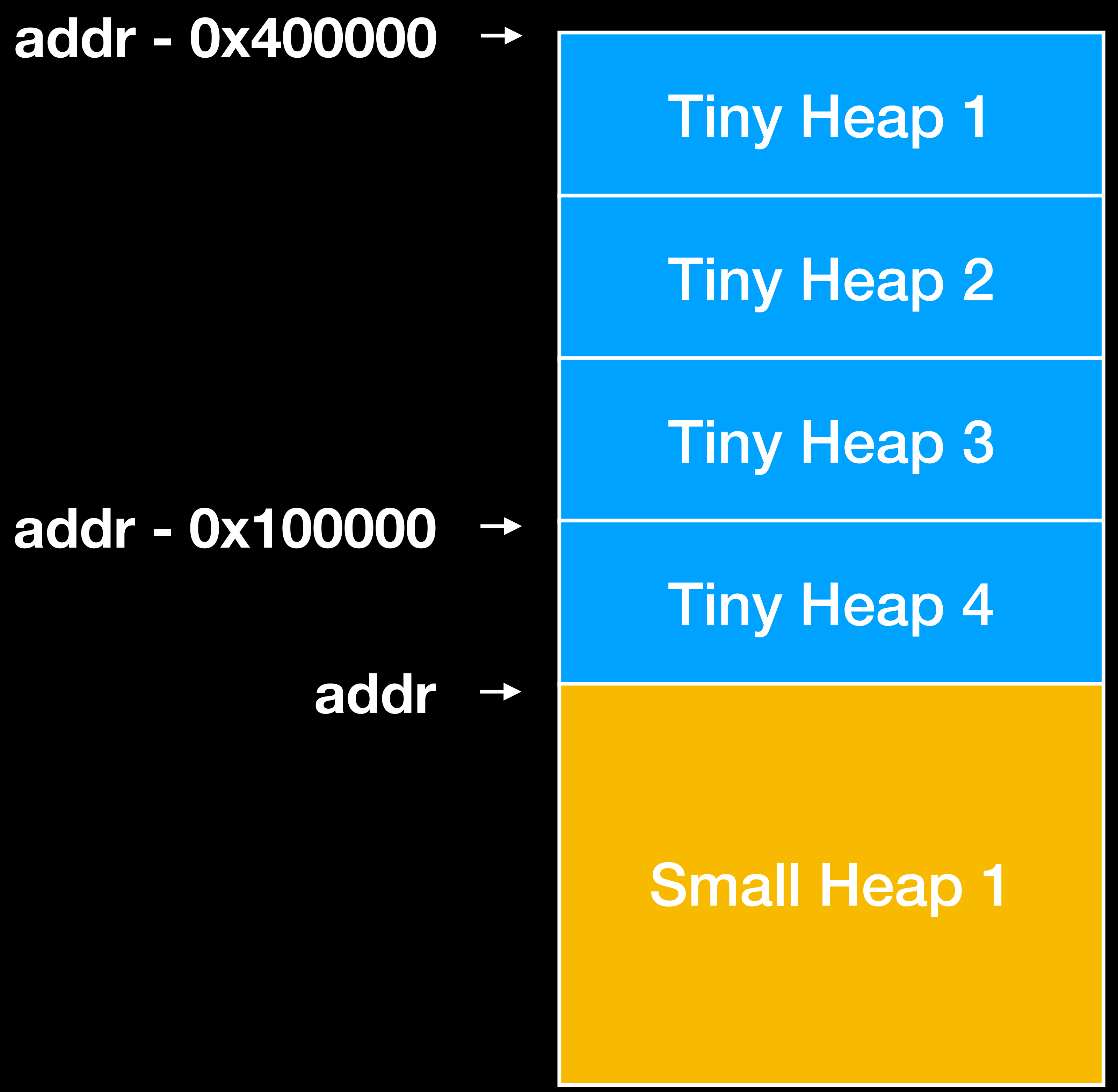
- Challenge info
- Vulnerability
- Exploit

Exploit Stage

- Information leak
- Corrupting Small Heap Region Metadata
- Overlapped chunk
- Shared cache
- Overwrite FILE pointer
- Arbitrary Write

Information leak

- Using bookname overflow vulnerability to leak Tiny Heap address
- Tiny Heap and Small Heap are adjacent to each other
- Small Heap is always 0x800000 aligned
- Given Tiny Heap address, we can calculate Small Heap address

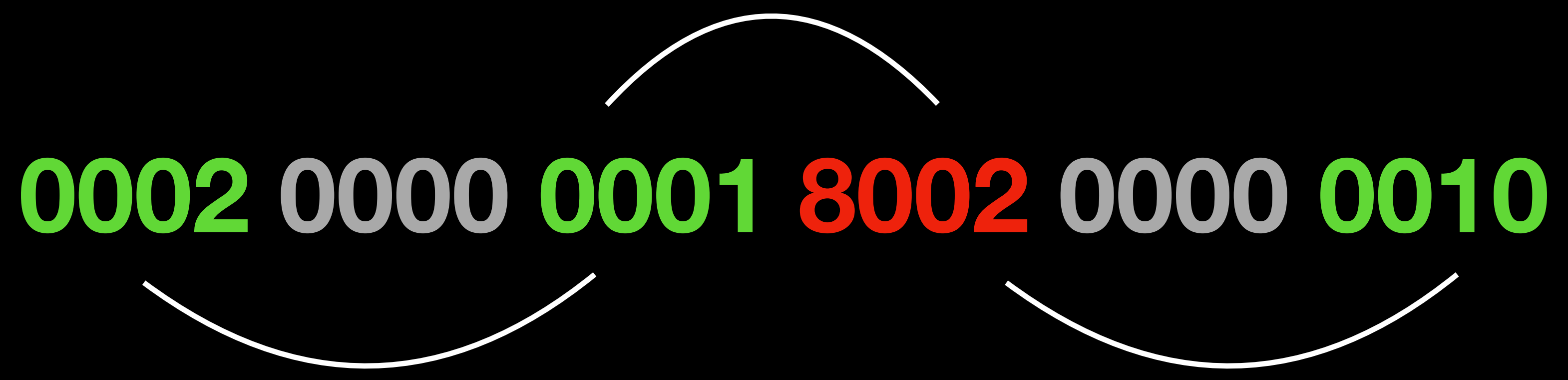


Exploit Stage

- Information leak
- Corrupting Small Heap Region Metadata
- Overlapped chunk
- Shared cache
- Overwrite FILE pointer
- Arbitrary Write

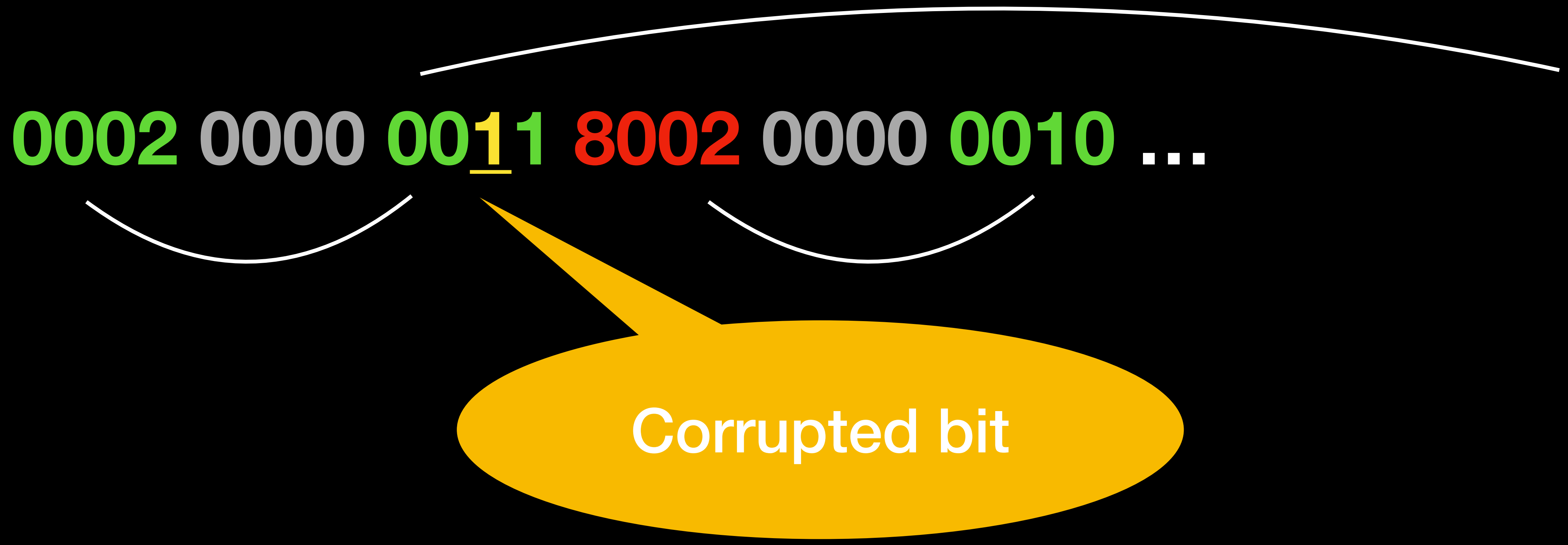
Small Heap Metadata

- Small Heap metadata describe allocated chunk size and is free status
  - uint16_t
  - Highest bit indicates is free status
  - Other bits indicate how many blocks (1 block = 0x200)
- In macOS Catalina 10.15.7, Heap metadata is located at the beginning of each magazine



Small Heap Metadata Corruption

- If we have arbitrary write, we can corrupt Small Heap metadata to larger size
  - Free-ing the corrupted metadata chunk will lead to free-ing a chunk size larger than expected
- In this challenge, we can use bookname overflow vulnerability to overwrite 1 bit shown as below



how2free 🤔

- No free() is called in this challenge

how2free 🤔

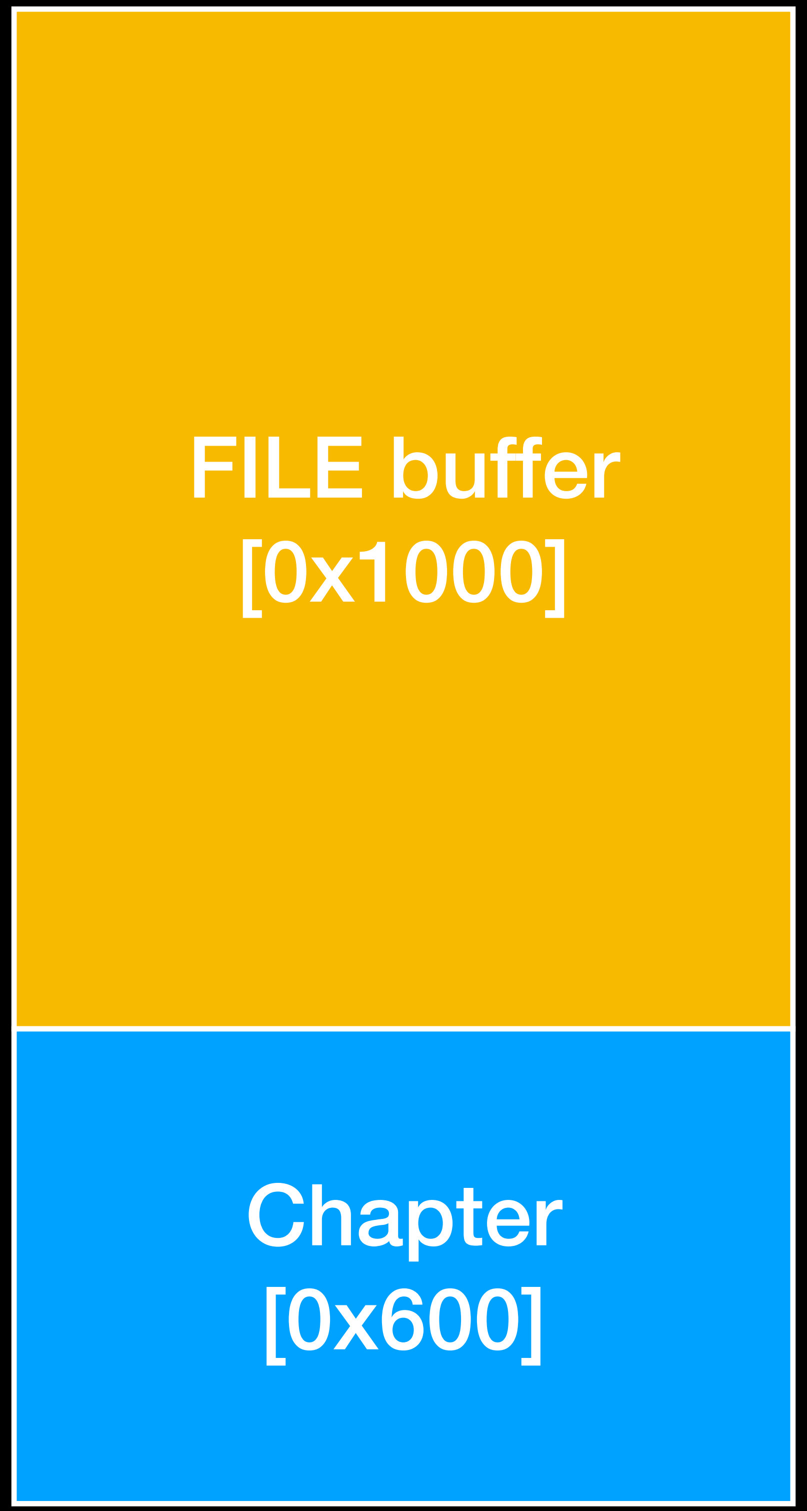
- No free() is called in this challenge
- fclose() will call free()
  - fopen() a file will allocate a 0x1000 buffer (in Small Heap)
  - fclose() to free that buffer
- Note: You may need to guess which Small Heap the buffer is allocated (probability = 1/4)

Exploit Stage

- Information leak
- Corrupting Small Heap Region Metadata
- Overlapped chunk
- Shared cache
- Overwrite FILE pointer
- Arbitrary Write

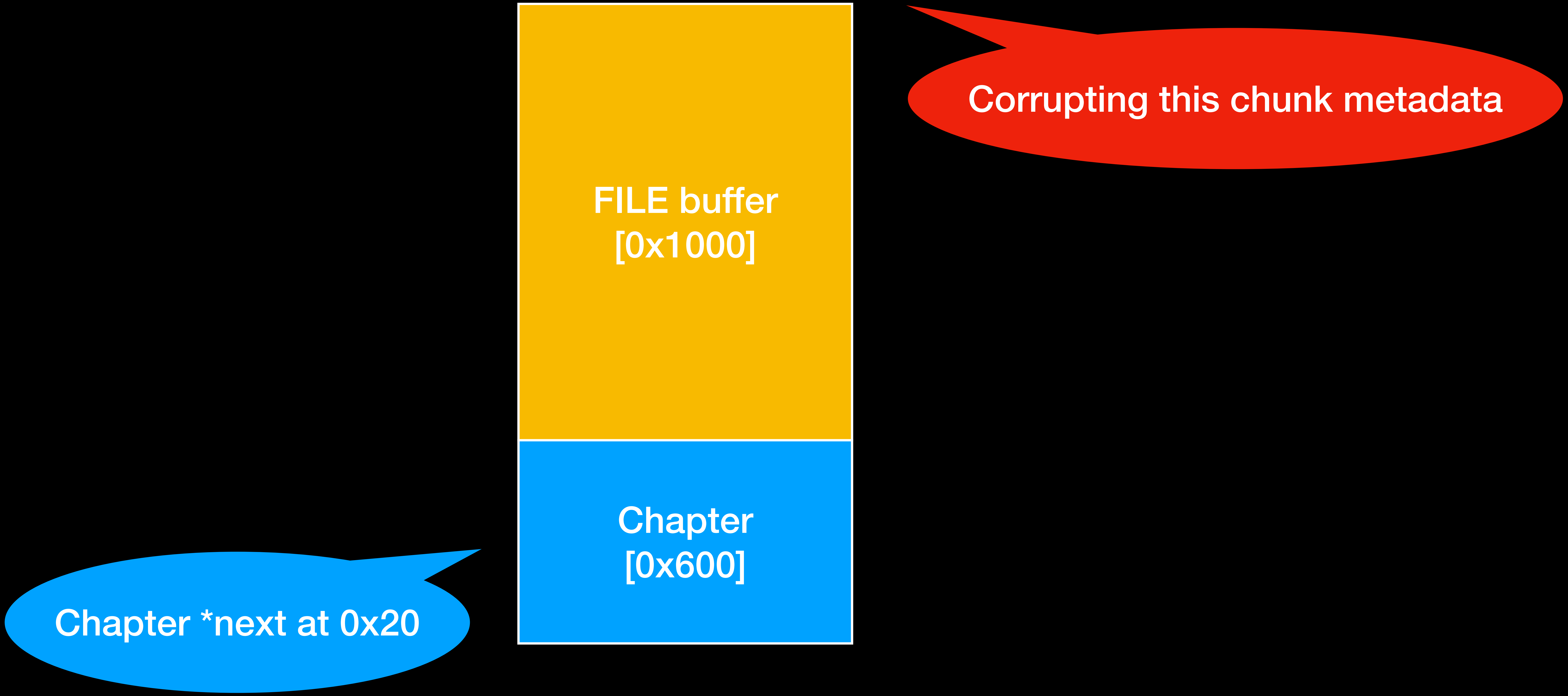
Overlapped Chunk

- We need a heap layout shown as following:



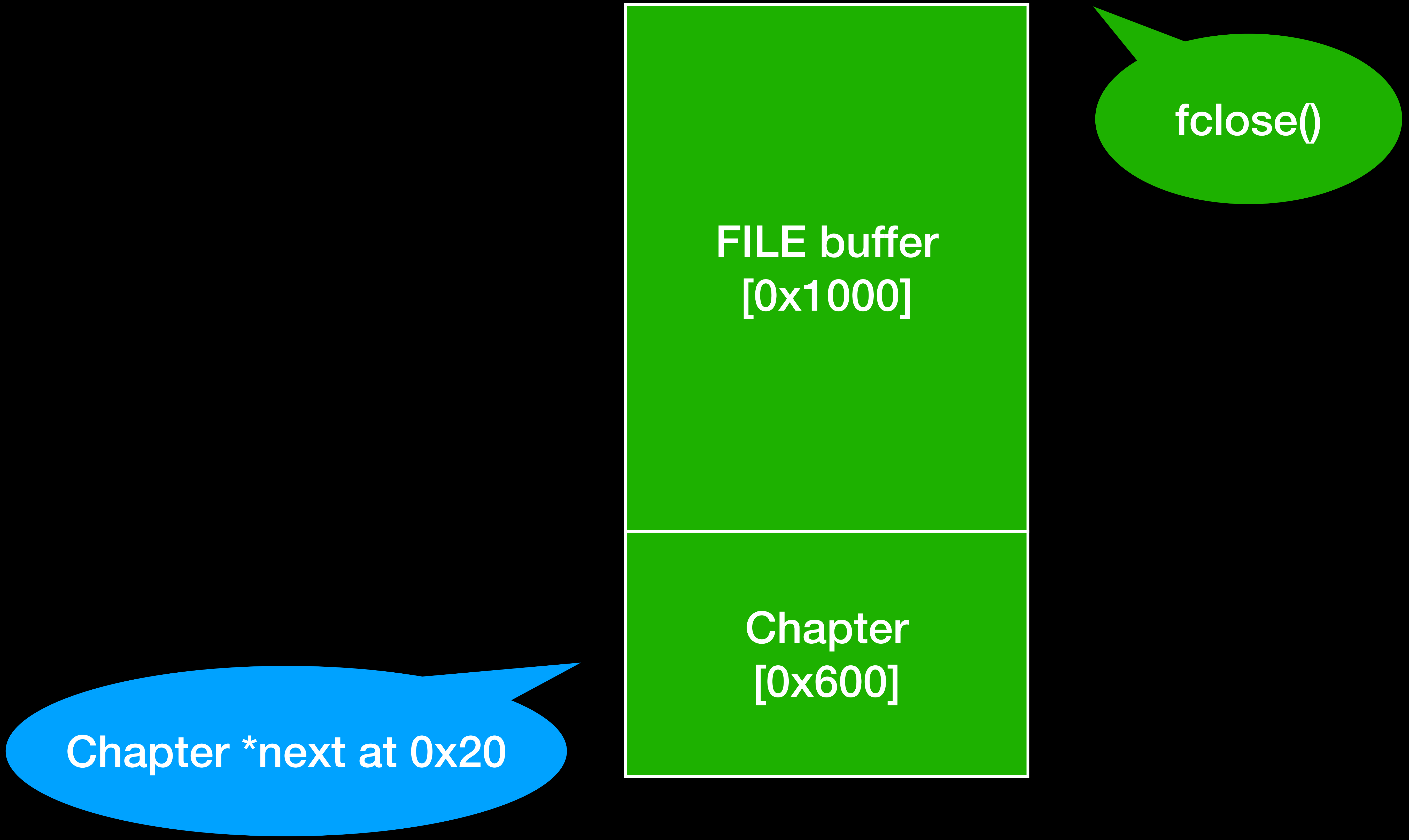
Overlapped Chunk

- We need a heap layout shown as following:



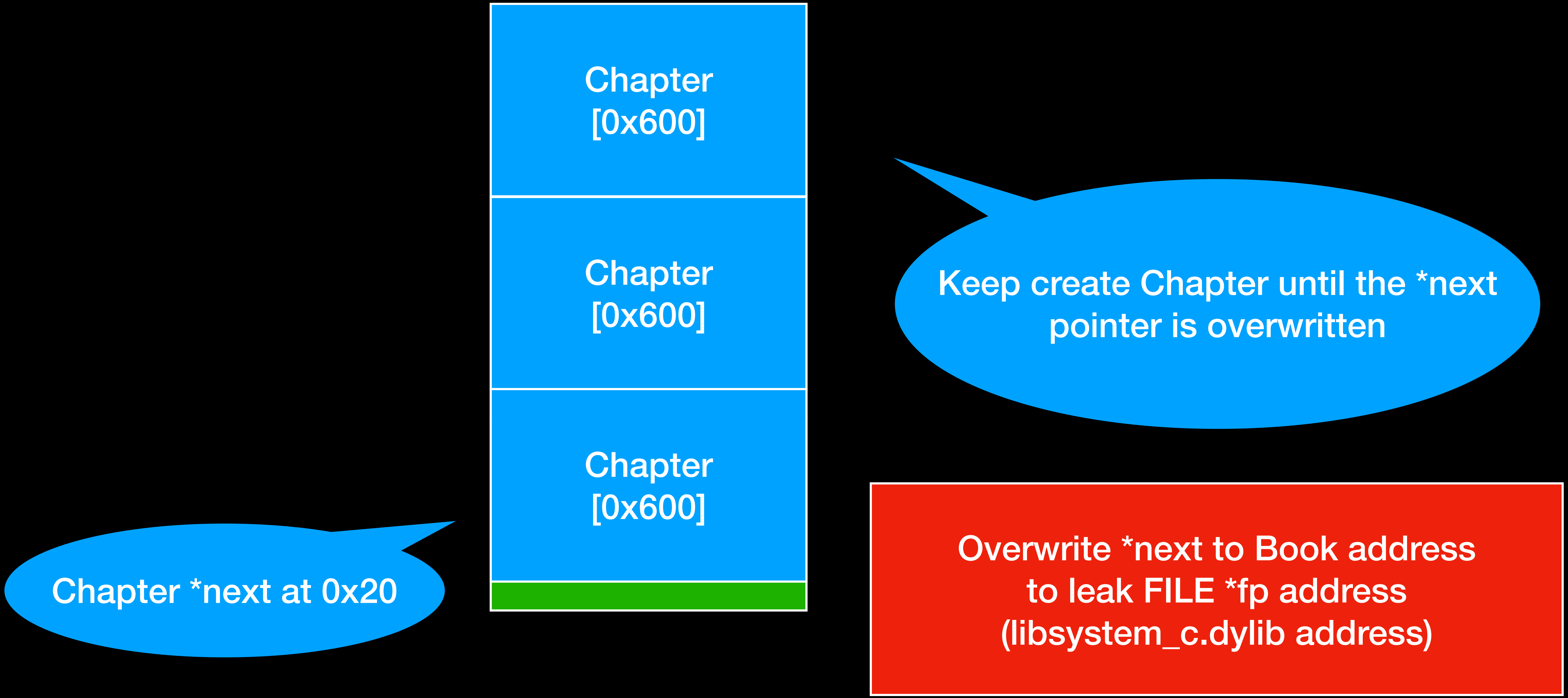
Overlapped Chunk

- We need a heap layout shown as following:



Overlapped Chunk

- We need a heap layout shown as following:


Chapter [0x600]
Chapter [0x600]
Chapter [0x600]
Keep create Chapter until the *next pointer is overwritten
Chapter *next at 0x20
Overwrite *next to Book address to leak FILE *fp address (libsystem_c.dylib address)

Exploit Stage

- Information leak
- Corrupting Small Heap Region Metadata
- Overlapped chunk
- Shared cache
- Overwrite FILE pointer
- Arbitrary Write

Shared Library Cache

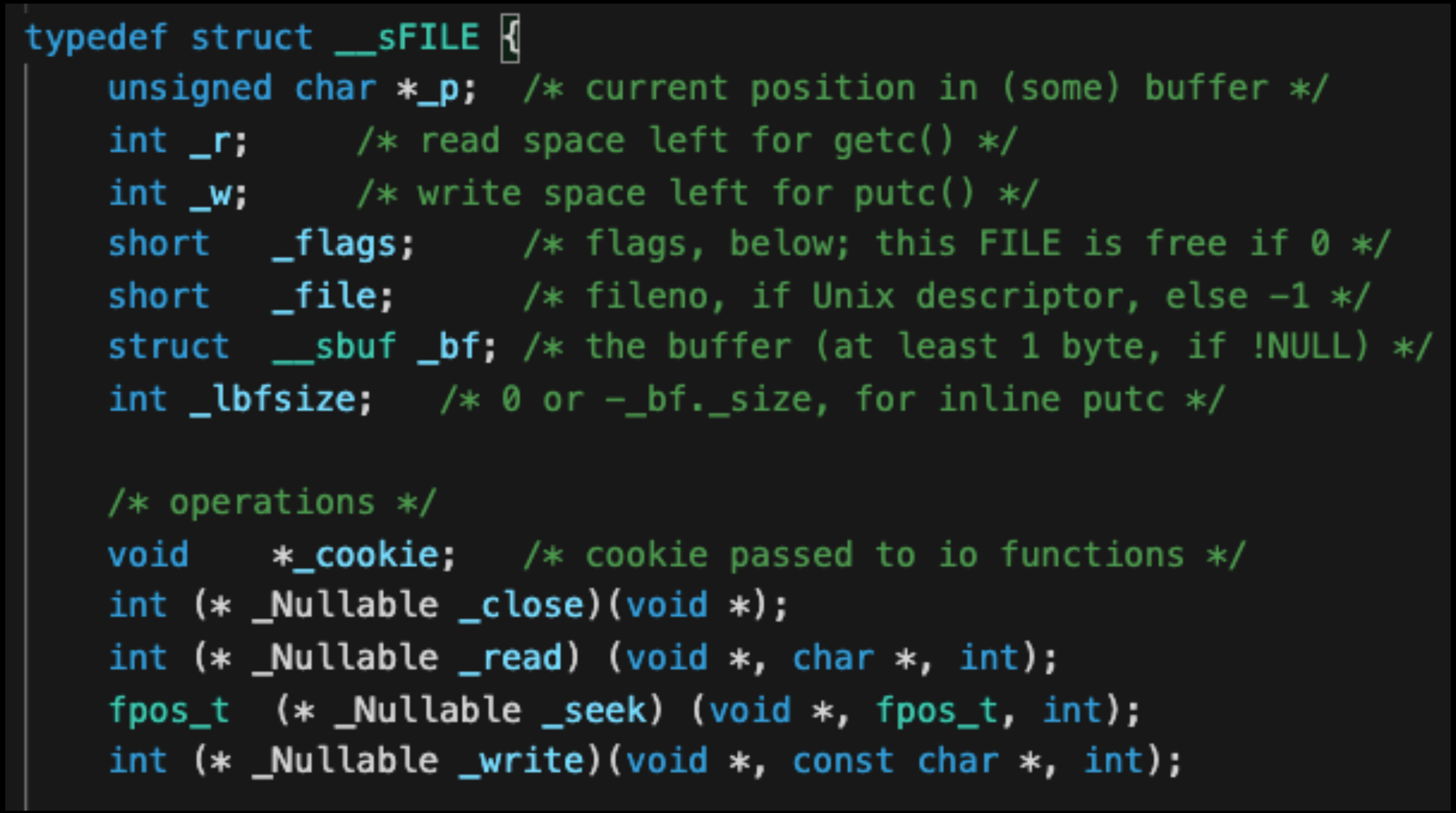
- All the heap allocation layout might not be stable after overlapped chunk
- Luckily due to macOS Shared Library Cache, we can leak library address in different connections

Exploit Stage

- Information leak
- Corrupting Small Heap Region Metadata
- Overlapped chunk
- Shared cache
- Overwrite FILE pointer
- Arbitrary Write

FILE in libsystem_c.dylib

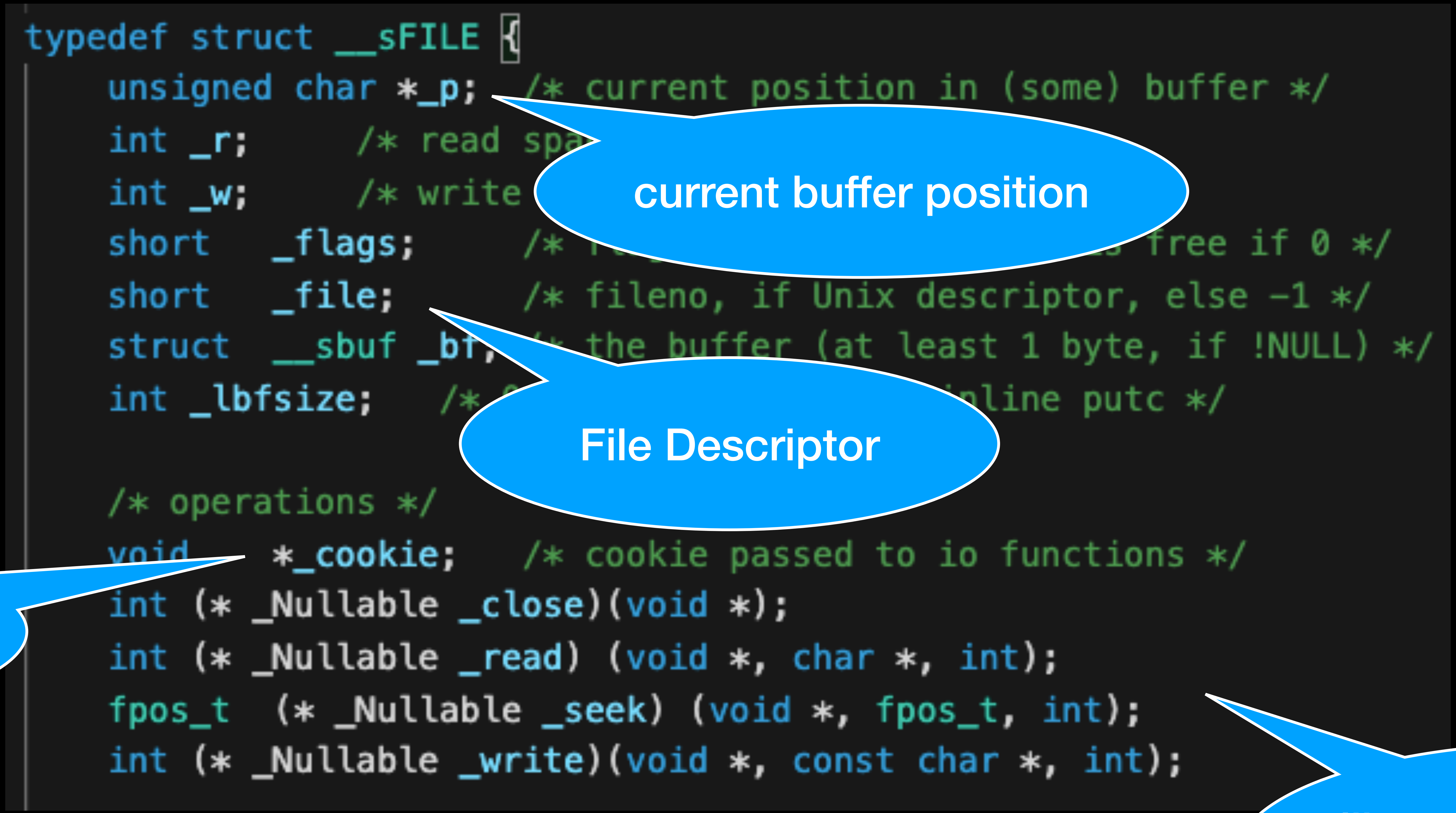
- FILE structure exploitation is also working on macOS :D



```
typedef struct __sFILE {
    unsigned char *_p;  /* current position in (some) buffer */
    int _r;     /* read space left for getc() */
    int _w;     /* write space left for putc() */
    short   _flags;     /* flags, below; this FILE is free if 0 */
    short   _file;      /* fileno, if Unix descriptor, else -1 */
    struct  __sbuf _bf; /* the buffer (at least 1 byte, if !NULL) */
    int _lbfsize;   /* 0 or -_bf._size, for inline putc */

    /* operations */
    void    *_cookie;   /* cookie passed to io functions */
    int (* _Nullable _close)(void *);
    int (* _Nullable _read) (void *, char *, int);
    fpos_t  (* _Nullable _seek) (void *, fpos_t, int);
    int (* _Nullable _write)(void *, const char *, int);
```

FILE in libsystem_c.dylib

- FILE structure exploitation is also working on macOS :D



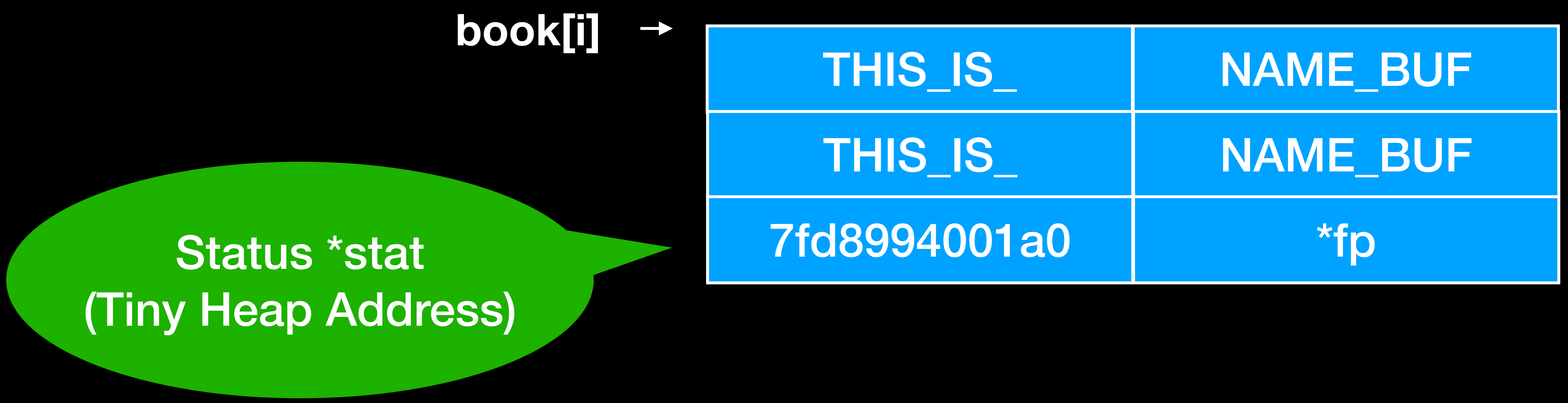
```
typedef struct __sFILE {
    unsigned char *_p;  /* current position in (some) buffer */
    int _r;      /* read spa
    int _w;      /* write
    short    _flags;      /*                         free if 0 */
    short    _file;       /* fileno, if Unix descriptor, else -1 */
    struct   __sbuf _bf;  /* the buffer (at least 1 byte, if !NULL) */
    int _lbfsize;    /*                    inline putc */

    /* operations */
    void     *_cookie;    /* cookie passed to io functions */
    int (* _Nullable _close)(void *);
    int (* _Nullable _read) (void *, char *, int);
    fpos_t  (* _Nullable _seek) (void *, fpos_t, int);
    int (* _Nullable _write)(void *, const char *, int);
```

FILE in libsystem_c.dylib

- FILE structure exploitation is also working on macOS :D
- If we can forge a FILE structure, we can control
  - Buffer address (where to read or write)
  - File descriptor (0: stdin, 1: stdout, …)

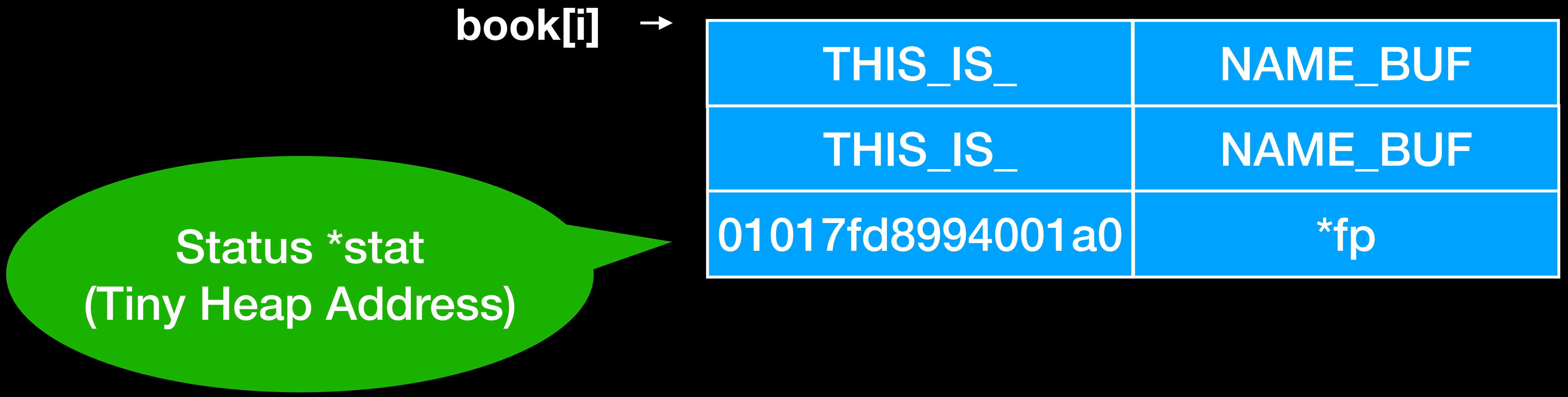
Overwrite FILE pointer

- Using bookname overflow vulnerability to overflow *fp



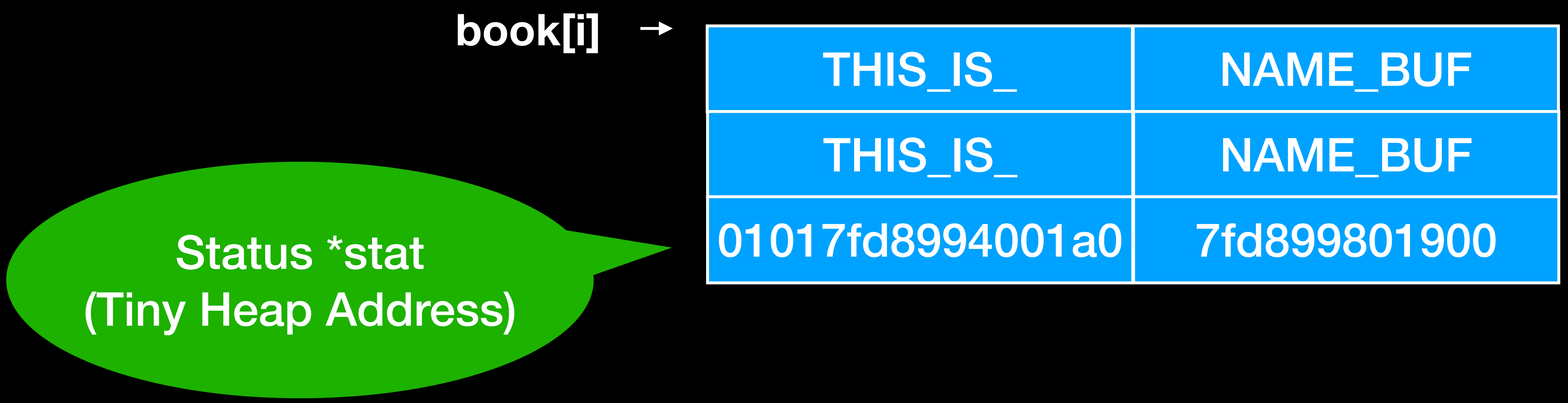
Overwrite FILE pointer

- Using bookname overflow vulnerability to overflow *fp
    - Selective 1 bit overwrite at (book[i] + 0x26)
    - Selective 1 bit overwrite at (book[i] + 0x27)



Overwrite FILE pointer

- Using bookname overflow vulnerability to overflow *fp
  - Selective 1 bit overwrite at (book[i] + 0x26)
  - Selective 1 bit overwrite at (book[i] + 0x27)
- **Edit Bookname** to overwrite *fp to a forged FILE structure on heap



Exploit Stage

- Information leak
- Corrupting Small Heap Region Metadata
- Overlapped chunk
- Shared cache
- Overwrite FILE pointer
- Arbitrary Write

Arbitrary Write

- Forge a FILE structure with
  - buffer = stack address
  - fd = 0 (stdin)
- **Reload Book** to trigger arbitrary write
- Overwrite return address to ROP

Summary

- Information leak: Leak Tiny & Small Heap
- Corrupting Small Heap Region Metadata: Create overlapped chunk
- Overlapped chunk: Arbitrary read
- Shared cache: For clean heap layout
- Overwrite FILE pointer: Arbitrary write
- Arbitrary Write: Hijack Control Flow

Exploit

Thanks for listening!

Balsn

@_how2hack_