WCTF 2019

世界黑客大师赛

WE CREATE THE FUTURE

LazyFragmentationHeap

Angelboy @ 217

Outline

- Description
- Vulnerability
- Exploit

Description

- Windows 10 - 64 bit PE
- Protection
  - DEP
  - CFG
  - ASLR
  - Child Process Policy

Description

- A magic file reader.
- You can read the magic.txt file in current directory.
- You must allocate a buffer at first.

```
******************************
    LazyFragmentationHeap
******************************
 1. Allocate buffer for File
 2. Edit File content
 3. Show content
 4. Clean content
 5. LazyFileHandler
 6. Exit
******************************
Your choice: 5
==============================
      Lazy File Handler
==============================
 1. OpenFile
 2. ReadFile
 3. Back
==============================
Your choice:
```

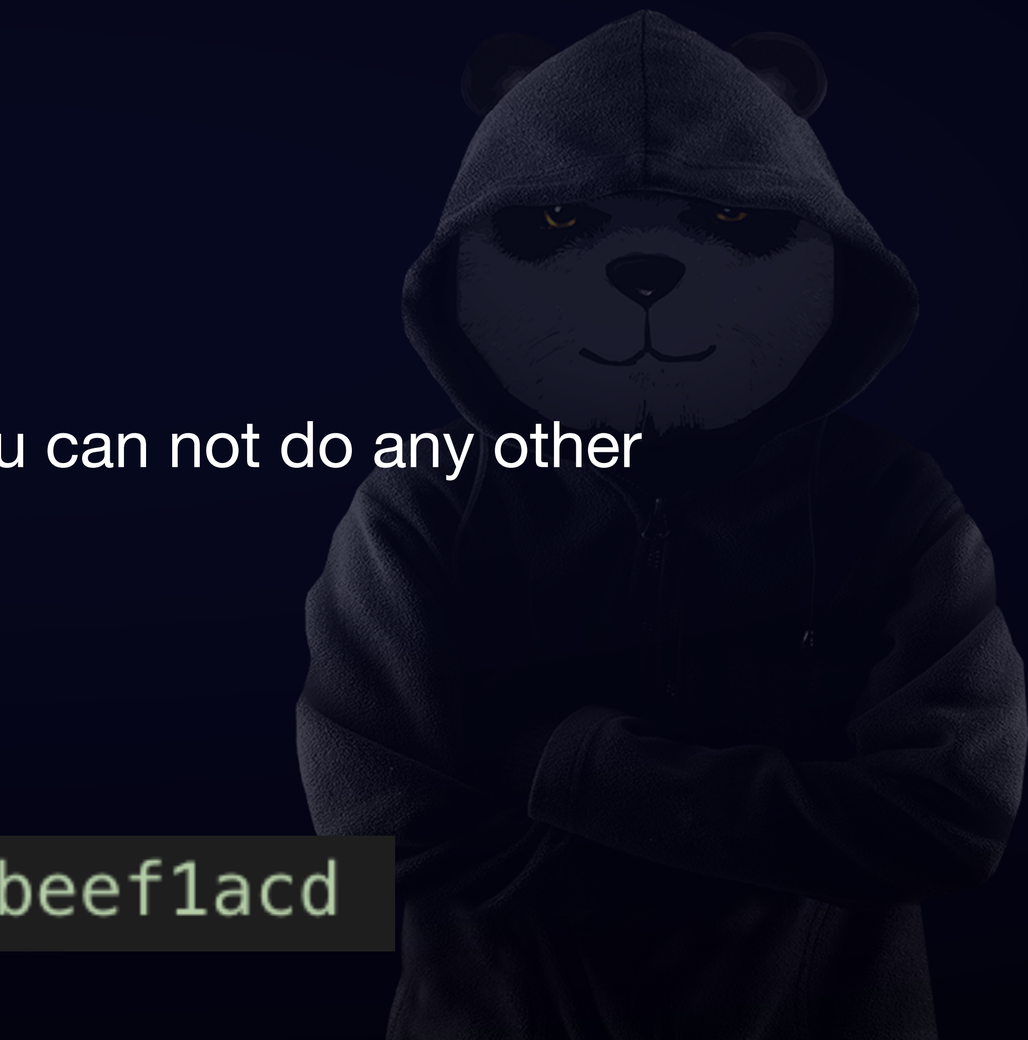
Description

- Structure
  - The data structure be used to store is fdata like the Figure.
  - It would record id,content and size of content.
  - It has extra member magic to protect data in the structure.

```
struct fdata {
    size_t magic2;
    size_t size;
    size_t id;
    size_t magic;
    char* content;
};
```

Description

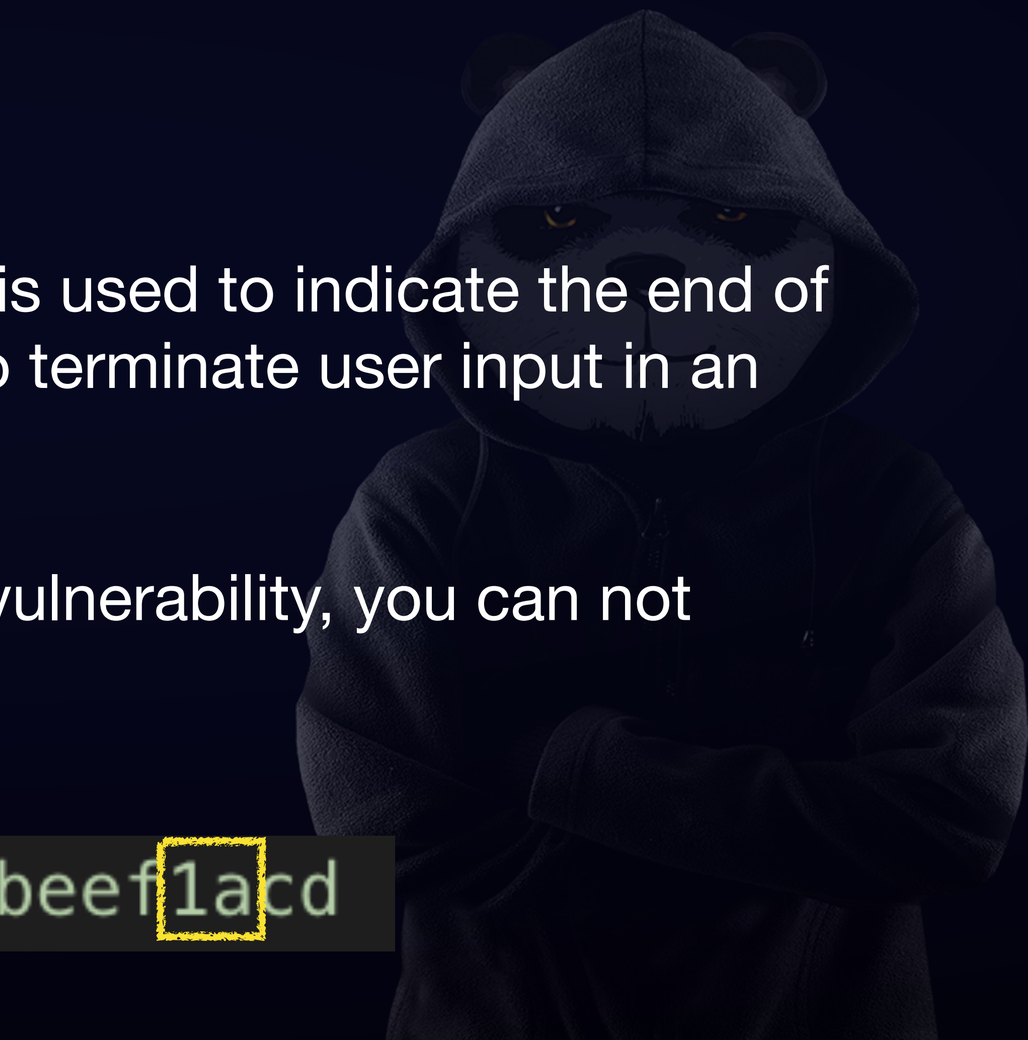
- Structure
  - About magic
    - The value of magic is 0xddaabeef1acd
    - If the magic is not equal to the value, you can not do any other operation.



```
#define MAGIC 0xddaabeef1acd
```

Description

- Structure
  - About magic
    - In windows, the SUB(0x1a) character is used to indicate the end of character stream, and thereby used to terminate user input in an interactive command line.
    - Therefore if you have buffer overflow vulnerability, you can not overwrite it from user input directly.



```
#define MAGIC 0xddaabeef1acd
```

Description

- Allocate
  - Just allocate a buffer for file reader
  - You can specify size & id for the buffer
- Edit File content
  - You can modify the content of file buffer
  - You only can do it once for a file buffer.

Description

- Show content
  - Write out the content of file buffer
- Clean content
  - Release the file buffer

Description

- LazyFileHandler
  - OpenFile
    - Open the magic.txt
  - ReadFile
    - Read from the content of the opened file to specified file buffer
    - You only can do it twice.

Outline

- Description
- Vulnerability
- Exploit

Vulnerability

- Information leak
  - It uses *fread()* and *read_input()* to read the content of file to file buffer and it not terminated with NULL byte.
  - So if it has some sensitive value follow by the buffer, you can leak it.

```
fread_s(filebuffer[i].content, size, 1, size, fp);
puts("Done !");
return;
```

Vulnerability

- Heap overflow
  - When you modify the content of file buffer, it would use *strlen()* to calculate the size if the size is large than the size stored in fdata. If the content does not have a NULL byte truncation, it will lead to heap overflow.
  - Actually, you can only overwrite the header of next chunk.

```
if (filebuffer[i].content && (filebuffer[i].magic == MAGIC) && filebuffer[i].magic2 == MAGIC) {
    printf("Content:");
    if ((strlen(filebuffer[i].content) > filebuffer[i].size) && filebuffer[i].magic == MAGIC) {
        size = strlen(filebuffer[i].content);
    }
    else {
        size = filebuffer[i].size;
    }
    read_input(filebuffer[i].content, size);
```

Outline

- Description
- Vulnerability
- Exploit

Exploit

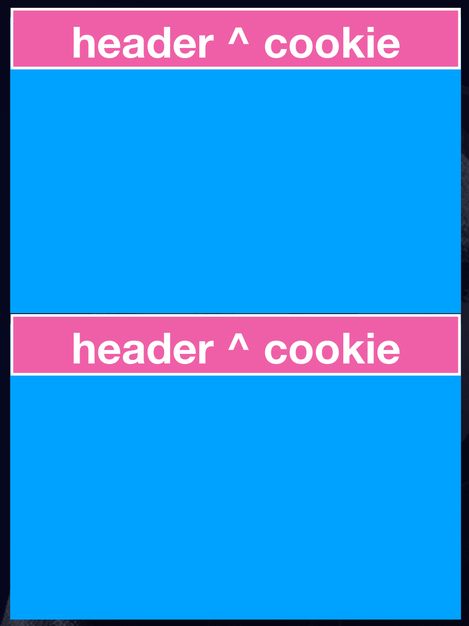
- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

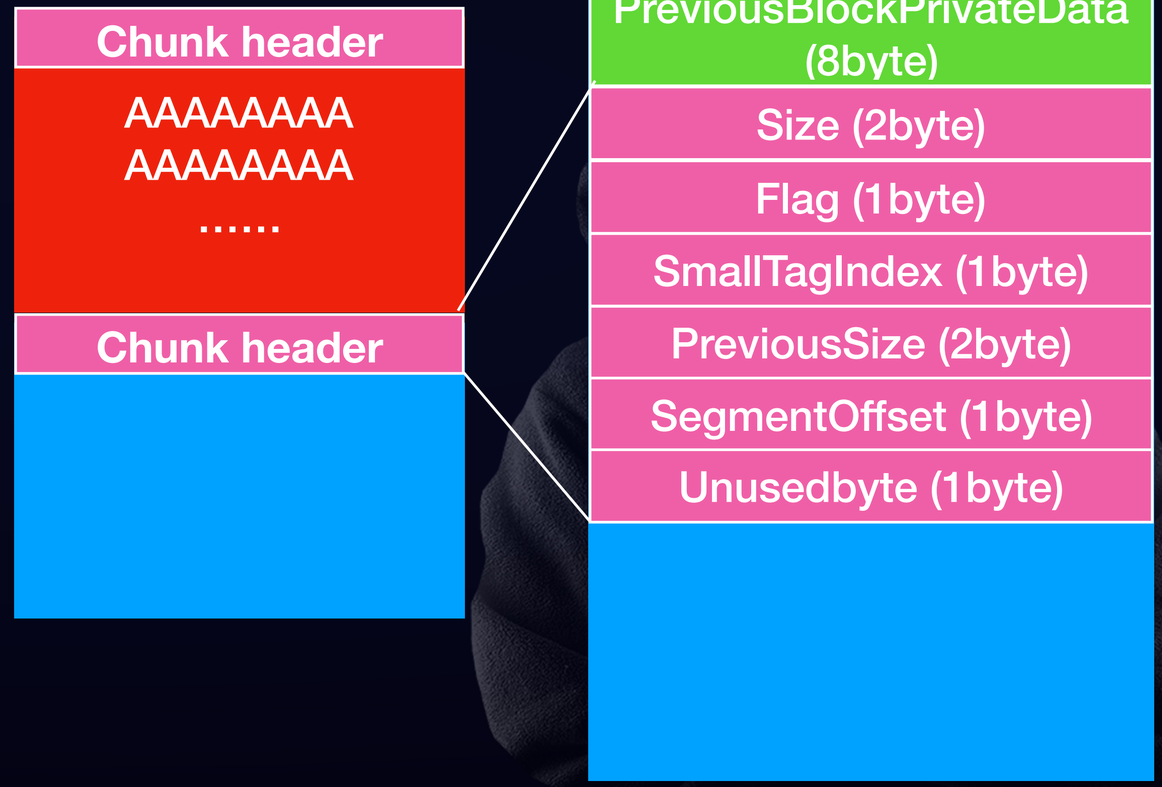
Exploit

- About windows heap
  - Windows 10 is using Nt heap for memory allocator by default
  - Every header of chunk in Nt heap is xor with _HEAP->Encoding
    - If you want to modify the chunk header correctly, you need to leak it.



Exploit

- Leak _HEAP->Encoding
  - Use heap Feng Shui to arrange the heap layout and use the vulnerability to leak the header of next chunk.
  - You can use it to calculate the Encoding cookie

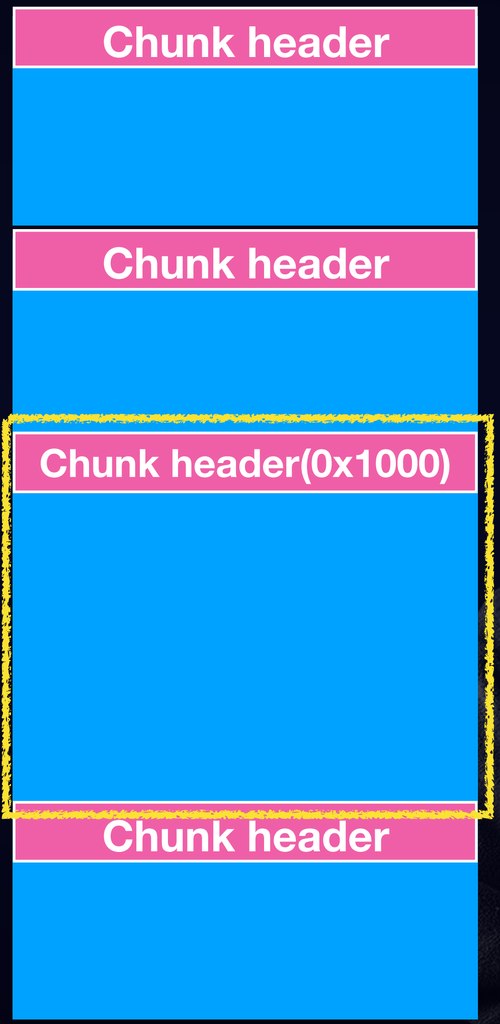


Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

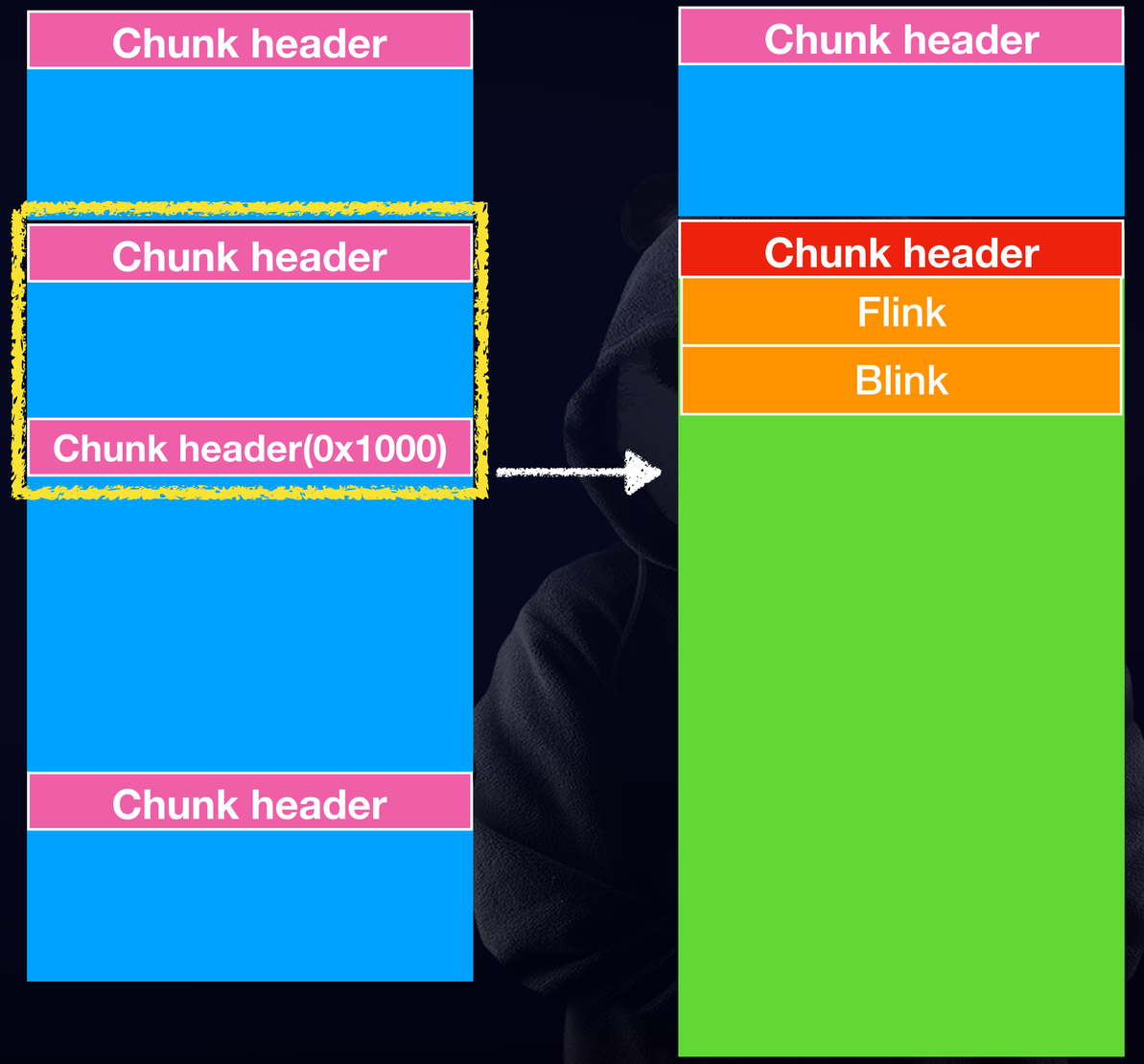
Exploit

- Create overlap chunk
  - After we leak the cookies, we rearrange the heap like the diagram.
  - The third chunk is a large chunk.
    - Size about 0x1000



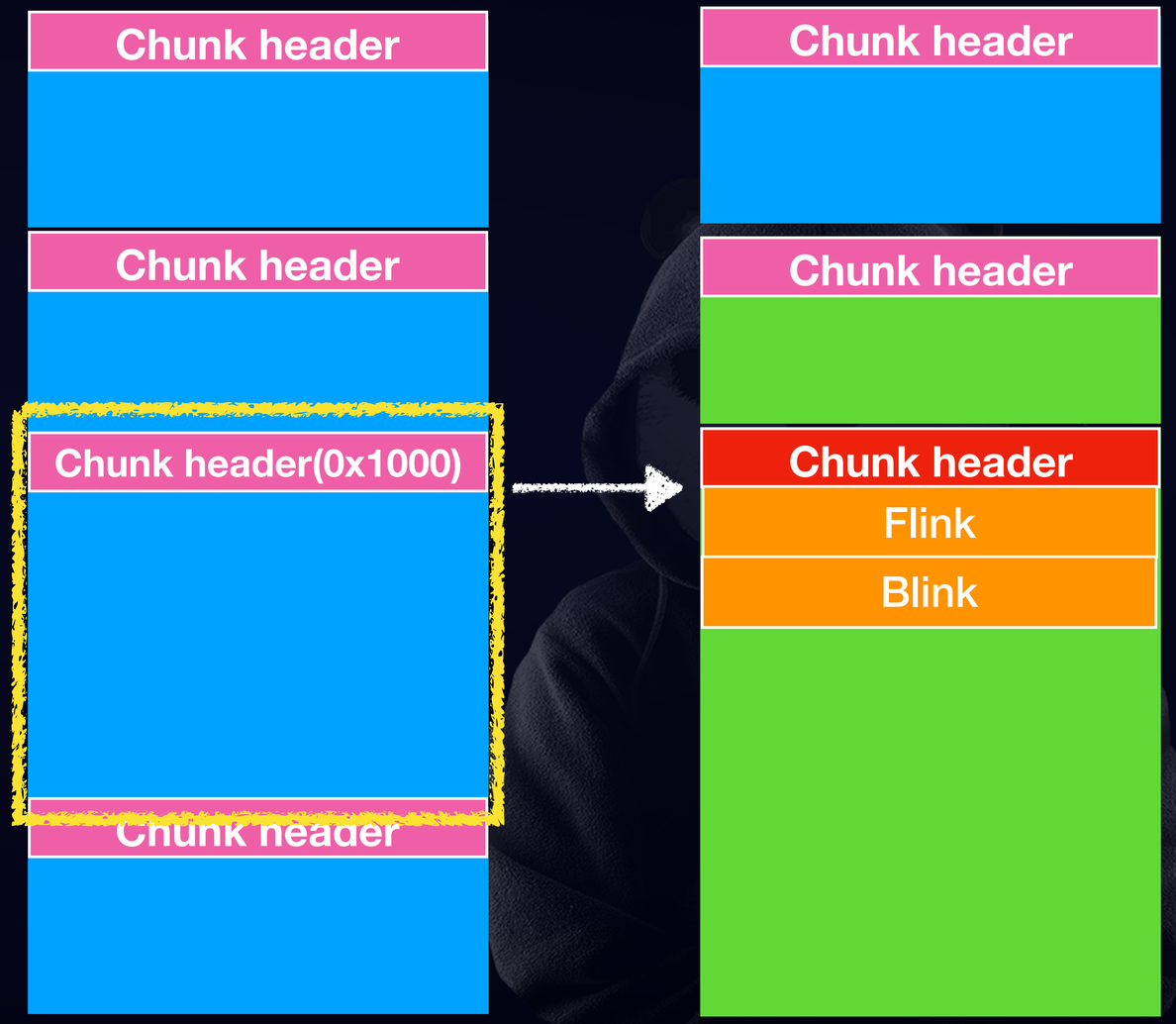
Exploit

- Create overlap chunk
  - Use the vulnerability to overwrite the second chunk and free the second chunk
  - Because of the heap coalesce mechanism, we will get a large overlap chunk



Exploit

- Create overlap chunk
  - Allocate a chunk of the same size
  - We can use third chunk to leak heap address

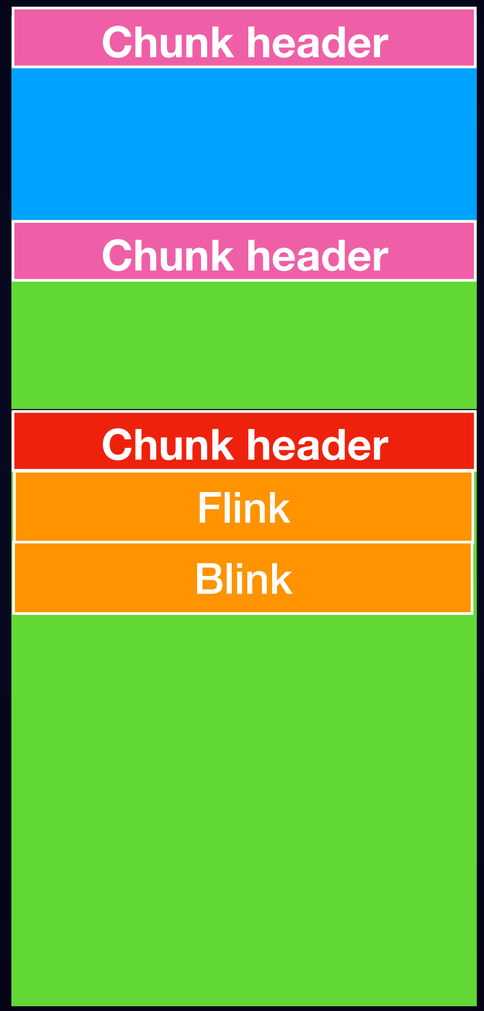


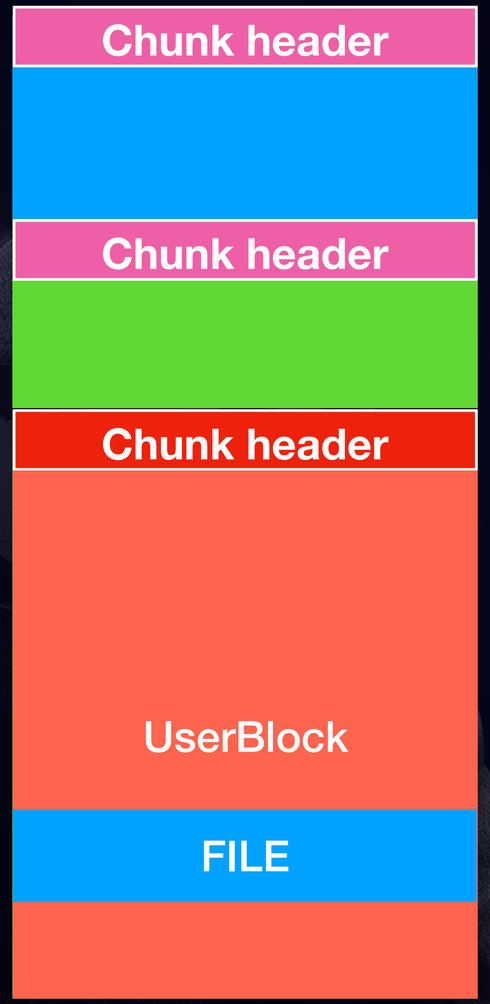
Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

Exploit

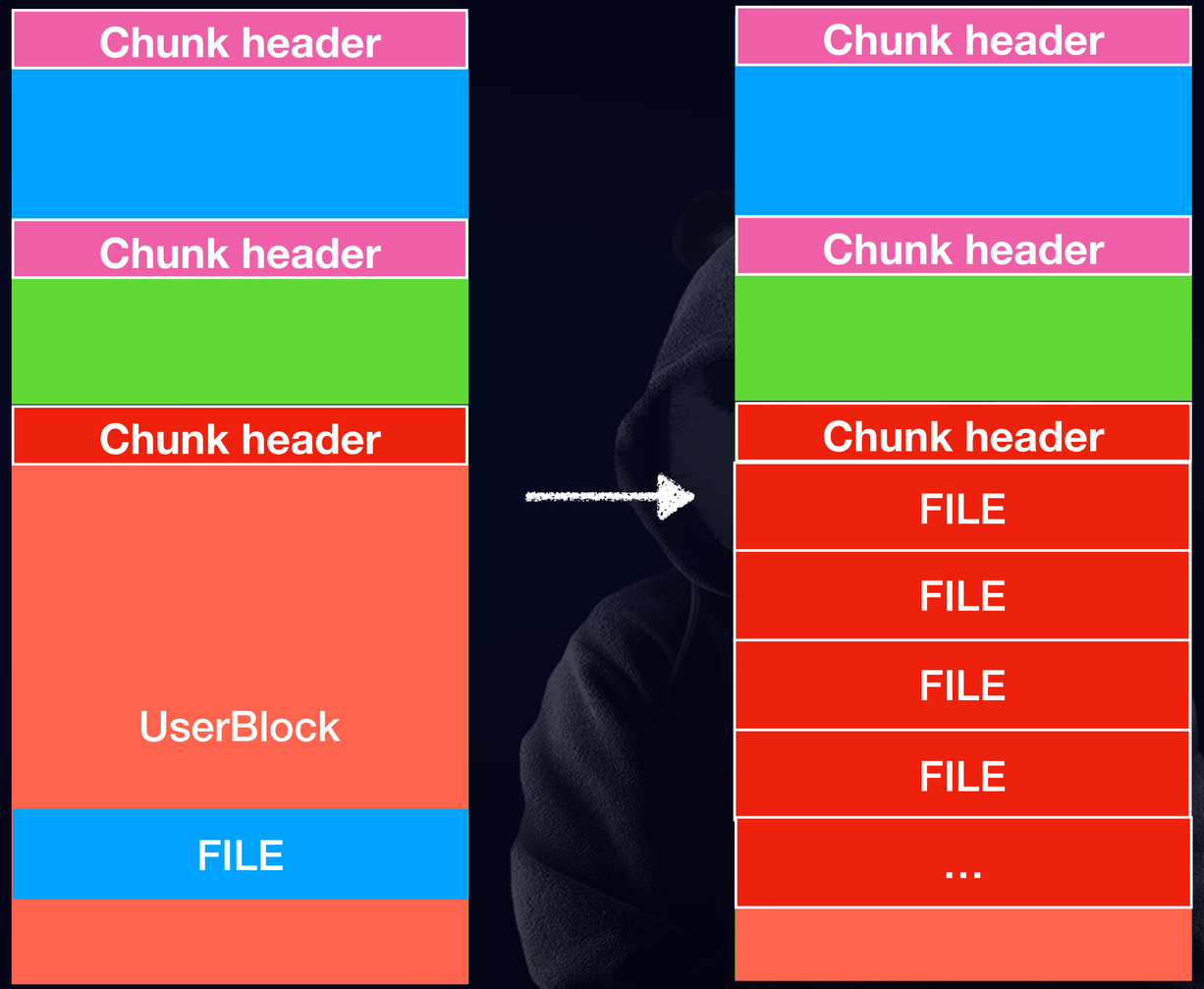
- Overwrite FILE structure
  - Use open file to create FILE structure and then trigger LowFragmentationHeap.
    - It would allocate a large chunk about 0x1000 byte for userblock, that is, it will fill third chunk.
  - The chunk will be UserBlock for LFH and FILE will allocate in it.
    - But because of randomness of LFH, we can not know address of FILE





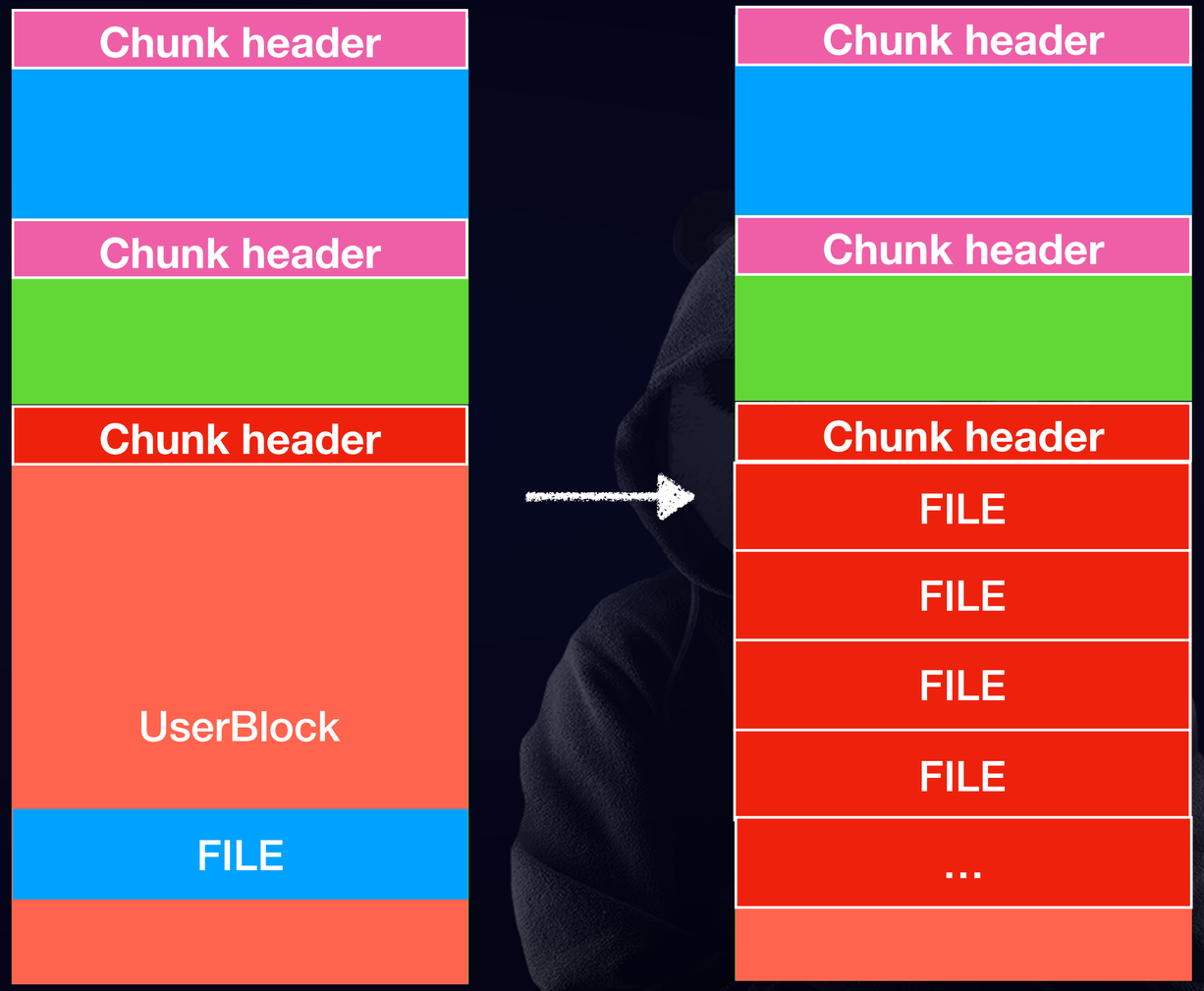
Exploit

- Overwrite FILE structure
  - We can use edit to modify the content of buffer.
    - Use it to fill up fake FILE structure in the buffer



Exploit

- Overwrite FILE structure
  - Now, we can use the FILE exploitation technique to do arbitrary memory writing once.



Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

Exploit

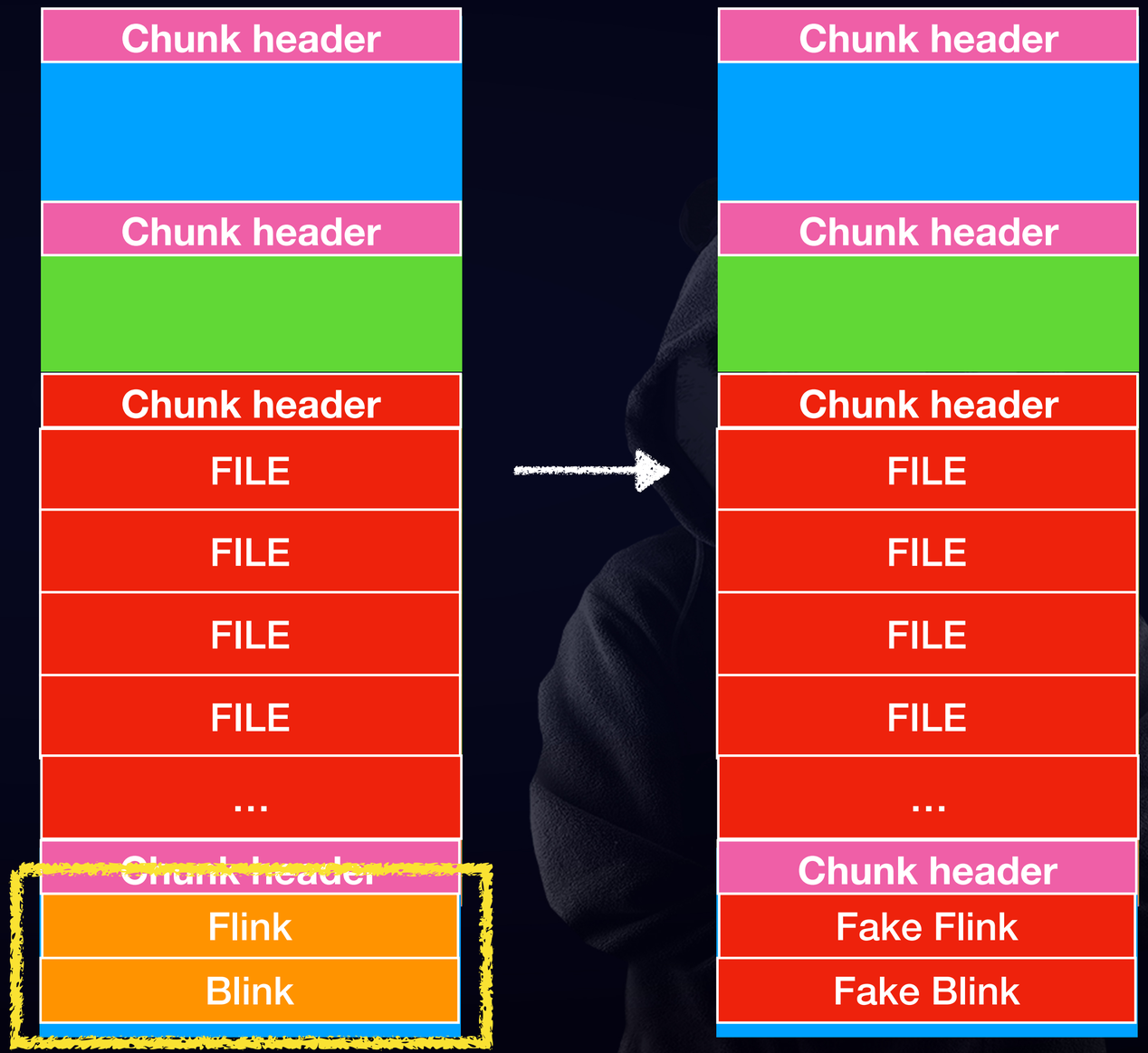
- Overwrite FILE structure
  - Because of magic in the fdata, we can not overwrite fdata directly
  - We need to overwrite ucrtbase!_pioinfo[0].flag to change stream to binary mode. (You need overwrite buffer pointer to leak ucrtbase first)
    - We use the technique to bypass the SUB character problem.
    - After do that it can read anything from our input

Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

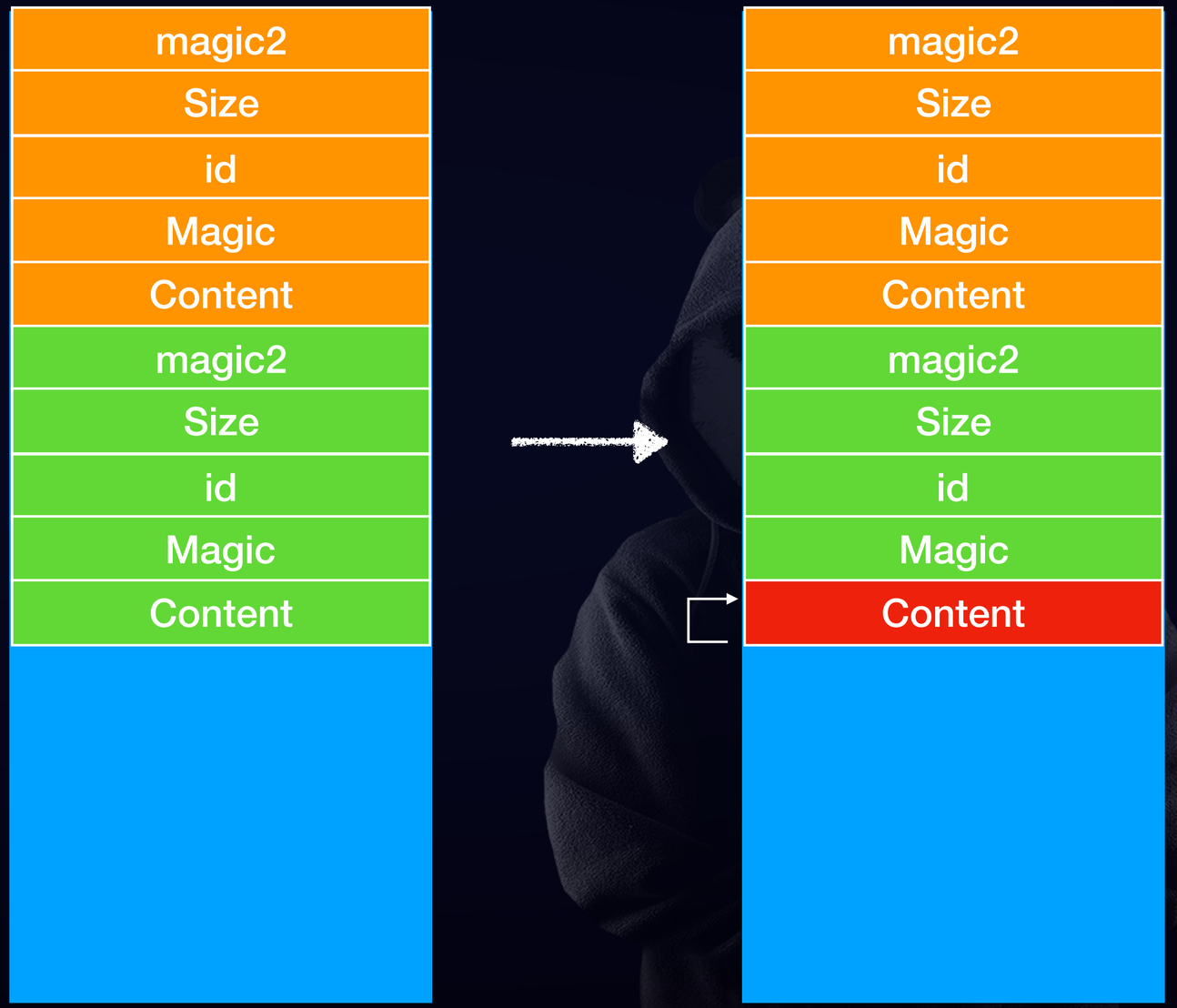
Exploit

- Unlink attack
  - We can use overlap chunk in last step, and then overwrite Flink and Blink



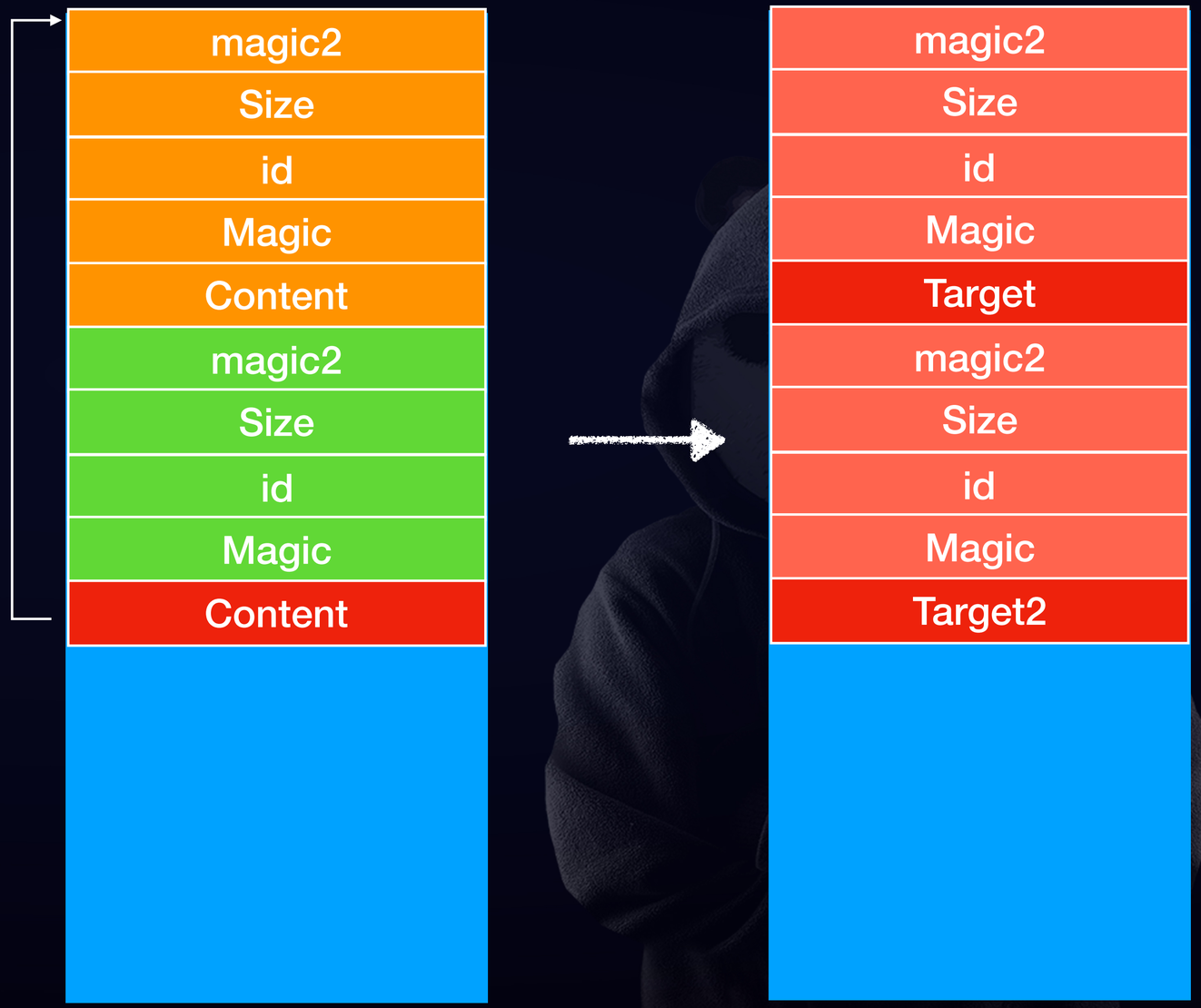
Exploit

- Unlink attack
  - We can create overlap chunk again, and then overwrite Flink and Blink
  - Then using unlink attack to overwrite filebuffer pointer with filebuffer



Exploit

- Unlink attack
  - Now, we can use modify filebuffer to overwrite anything in fdata structure.
  - That is, we can do arbitrary memory reading and writing.

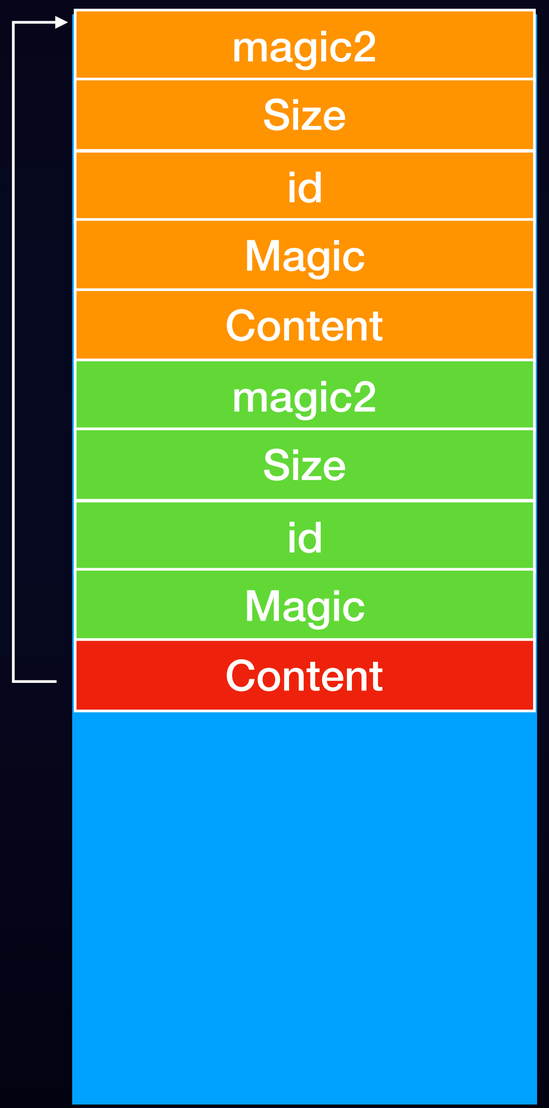


Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

Exploit

- Arbitrary memory reading and writing
  - Leak
    - _HEAP->lock (ntdll.dll)
    - ntdll!PebLdr
    - Binary
    - Kernel32.dll
    - Peb/teb
    - Stack



Exploit

- Leak _HEAP->Encoding
- Create overlap chunk
- Overwrite FILE structure
- Overwrite ucrtbase!_pioinfo[0]
- Unlink attack
- Arbitrary memory reading and writing
- ROP to read flag

Exploit

- ROP to read flag
  - Create new heap
  - Modify process heap
    - Prevent heap corruption problem
  - VirtualAlloc
  - Run shellcode to read the flag

Thank you for listening

angelboy@chroot.org

@scwuaptx