

GLIBC **FILE** Structure

TABLE OF CONTENTS

01 Introduction
FILE Structure 是什麼？可以吃口

02 Background
一些 FILE structure 相關的知識

03 Arbitrary Read
任意讀

04 Arbitrary Write
任意寫

05 Hijack vtable
控 Execution Flow

06 MISC
也許是講古(0

INTRODUCTION

\$ Introduction

- 前兩堂提到了許多利用手法，然而實際在 **PWN** 的時候，很多時候都圍繞在這兩個問題
 - How to bypass **ASLR**?
 - How to control **execution flow**?
- 這堂我們會講一個常見、強大的利用手法
 - C standard library I/O mechanism
- 我們一樣是主要討論 **GLIBC 2.31** 上的實作
 - 許多 **C** 標準函式庫實作皆有相似的設計

\$ C standard library I/O

- 如果你寫過 **C**，相信你一定用過下面的 **function**
 - **scanf**
 - **printf**
 - **fread**
 - **fwrite**
 - **fopen**
 - **fclose**
 - ...
- **Q: GLIBC** 是如何去處理這些 **function** 底層的 **I/O** 的呢?

\$ High-level overview

User space

C standard library I/O
printf, scanf, fread, fwrite, fopen, fclose...

System I/O primitives (System Call)
read, write, open, close ...

Kernel space

System Call Handler

\$ High-level overview

User space

C standard library I/O
printf, scanf, fread, fwrite, fopen, fclose...

Buffer

System I/O primitives (System Call)
read, write, open, close ...

Kernel space

System Call Handler

\$ High-level overview

User space

C standard library I/O
printf, scanf, fread, fwrite, fopen, fclose...

Buffer

System I/O primitives (System Call)
read, write, open, close ...

Kernel space

System Call Handler

BACKGROUN

\$ FILE

- fopen

- __fopen_internal ([libio/iofopen.c](#))

```
FILE *  
__fopen_internal (const char *filename, const char *mode, int is32)  
{  
    struct locked_FILE  
    {  
        struct _IO_FILE_plus fp;  
        ...  
    } *new_f = (struct locked_FILE *) malloc (sizeof (struct locked_FILE));  
    ...  
}
```

\$ _IO_FILE_plus

- struct _IO_FILE_plus ([libio/libioP.h](#))

```
struct _IO_FILE_plus
{
    FILE file;
    const struct _IO_jump_t *vtable;
};
```

- libio/bits/types/FILE.h ([libio/bits/types/FILE.h](#))

```
typedef struct _IO_FILE FILE;
```

\$ _IO_FILE

- **struct _IO_FILE** ([libio/bits/types/struct_FILE.h](#))
 - flags
 - Read buffer
 - Write buffer
 - Buffer base
 - Buffer end
 - chain
 - fileno

```
struct _IO_FILE
{
    int _flags;      /* High-order word is _IO_MAGIC; rest is flags. */

    /* The following pointers correspond to the C++ streambuf protocol. */
    char *_IO_read_ptr; /* Current read pointer */
    char *_IO_read_end; /* End of get area. */
    char *_IO_read_base; /* Start of putback+get area. */
    char *_IO_write_base; /* Start of put area. */
    char *_IO_write_ptr; /* Current put pointer. */
    char *_IO_write_end; /* End of put area. */
    char *_IO_buf_base; /* Start of reserve area. */
    char *_IO_buf_end; /* End of reserve area. */

    ...
    struct _IO_FILE *_chain;

    int _fileno;

    ...
};
```

\$ _IO_FILE

- **_flags** ([libio/libio.h](#))
 - **_IO_UNBUFFERED**
 - **_IO_LINE_BUF**

```
#define _IO_MAGIC          0xFBAD0000 /* Magic number */
#define _IO_MAGIC_MASK    0xFFFF0000
#define _IO_USER_BUF      0x0001 /* Don't deallocate buffer on close. */
#define _IO_UNBUFFERED    0x0002
#define _IO_NO_READS      0x0004 /* Reading not allowed. */
#define _IO_NO_WRITES     0x0008 /* Writing not allowed. */
#define _IO_EOF_SEEN      0x0010
#define _IO_ERR_SEEN      0x0020
#define _IO_DELETE_DONT_CLOSE 0x0040 /* Don't call close(_fileno) on close. */
#define _IO_LINKED        0x0080 /* In the list of all open files. */
#define _IO_IN_BACKUP     0x0100
#define _IO_LINE_BUF      0x0200
#define _IO_TIED_PUT_GET  0x0400 /* Put and get pointer move in unison. */
#define _IO_CURRENTLY_PUTTING 0x0800
#define _IO_IS_APPENDING  0x1000
#define _IO_IS_FILEBUF     0x2000
                          /* 0x4000 No longer used, reserved for compat. */
#define _IO_USER_LOCK     0x8000
```

\$ _IO_FILE memory layout

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		

\$ _IO_FILE_plus (cont.)

- struct _IO_FILE_plus ([libio/libioP.h](#))

```
struct _IO_FILE_plus
{
    FILE file;
    const struct _IO_jump_t *vtable;
};
```

\$ _IO_jump_t

- struct _IO_jump_t ([libio/libioP.h](#))
 - vtable



```
#define JUMP_FIELD(TYPE, NAME) TYPE NAME

struct _IO_jump_t
{
    JUMP_FIELD(size_t, __dummy);
    JUMP_FIELD(size_t, __dummy2);
    JUMP_FIELD(_IO_finish_t, __finish);
    JUMP_FIELD(_IO_overflow_t, __overflow);
    JUMP_FIELD(_IO_underflow_t, __underflow);
    JUMP_FIELD(_IO_underflow_t, __uflow);
    JUMP_FIELD(_IO_pbackfail_t, __pbackfail);
    /* showmany */
    JUMP_FIELD(_IO_xsputn_t, __xsputn);
    JUMP_FIELD(_IO_xsgetn_t, __xsgetn);
    JUMP_FIELD(_IO_seekoff_t, __seekoff);
    JUMP_FIELD(_IO_seekpos_t, __seekpos);
    JUMP_FIELD(_IO_setbuf_t, __setbuf);
    JUMP_FIELD(_IO_sync_t, __sync);
    JUMP_FIELD(_IO_doallocate_t, __doallocate);
    JUMP_FIELD(_IO_read_t, __read);
    JUMP_FIELD(_IO_write_t, __write);
    JUMP_FIELD(_IO_seek_t, __seek);
    JUMP_FIELD(_IO_close_t, __close);
    JUMP_FIELD(_IO_stat_t, __stat);
    JUMP_FIELD(_IO_showmanyc_t, __showmanyc);
    JUMP_FIELD(_IO_imbue_t, __imbue);
};
```


\$ _IO_FILE_plus memory layout

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		vtable

\$ _IO_FILE_plus instance

- Standard Input / Output / Error

- libc data segment

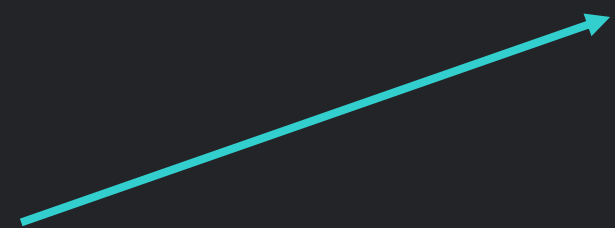
- _IO_2_1_stdin_

- _IO_2_1_stdout_

- _IO_2_1_stderr_

- stdin, stdout, stderr

- [libio/libio.h](#), [libio/stdio.c](#)



```
FILE *stdin = (FILE *) &_IO_2_1_stdin_;
```

```
FILE *stdout = (FILE *) &_IO_2_1_stdout_;
```

```
FILE *stderr = (FILE *) &_IO_2_1_stderr_;
```



- fopen()

- heap

- [libio/iofopen.c](#)

\$ How buffered I/O works

_IO_2_1_stdout_	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
_chain	
	_fileno
...	
vtable	

```
printf("Hello World");  
puts("");
```

* 因為這邊只是示意用，接下來的過程會簡化

\$ How buffered I/O works

_IO_2_1_stdout_	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
_chain	
	_fileno
...	
vtable	

```
printf("Hello World");  
      ↓  
__vfprintf_internal(stdout, "Hello World", arg, 0);
```

\$ How buffered I/O works

<u>_IO_2_1_stdout_</u>	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
	_chain
	_fileno
...	
	vtable

```
printf("Hello World");
```

```
__vfprintf_internal (stdout, "Hello World", arg, 0);
```

__dummy	__finish
...	
__xspn	__xsgetn
...	

\$ How buffered I/O works

<u>_IO_2_1_stdout_</u>	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
	_chain
	_fileno
...	
	vtable

```
printf("Hello World");  
↓  
__vfprintf_internal (stdout, "Hello World", arg, 0);  
↓
```

__dummy	__finish
...	
__xspn	__xsgetn
...	

\$ How buffered I/O works

_IO_2_1_stdout_	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
	_chain
	_fileno
...	
	vtable

```
printf("Hello World");
```

```
__vfprintf_internal (stdout, "Hello World", arg, 0);
```

__dummy	__finish
...	
_IO_new_file_xspn	__xsgetn
...	

\$ How buffered I/O works

<u>_IO_2_1_stdout_</u>	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
_chain	
	_fileno
...	
vtable	

```
printf("Hello World");  
    ↓  
__vfprintf_internal (stdout, "Hello World", arg, 0);  
    ↓  
_IO_new_file_xsputn (stdout, "Hello World", 11)
```


\$ How buffered I/O works

<code>_IO_2_1_stdout_</code>	
<code>_flags</code>	<code>_IO_read_ptr</code>
<code>_IO_read_end</code>	<code>_IO_read_base</code>
<code>_IO_write_base</code>	<code>_IO_write_ptr</code>
<code>_IO_write_end</code>	<code>_IO_buf_base</code>
<code>_IO_buf_end</code>	
...	
<code>_chain</code>	
	<code>_fileno</code>
...	
	<code>vtable</code>

```
_IO_new_file_xsputn (stdout, "Hello World", 11)
```



\$ How buffered I/O works

<code>_IO_2_1_stdout_</code>	
<code>_flags</code>	<code>_IO_read_ptr</code>
<code>_IO_read_end</code>	<code>_IO_read_base</code>
<code>_IO_write_base</code>	<code>_IO_write_ptr</code>
<code>_IO_write_end</code>	<code>_IO_buf_base</code>
<code>_IO_buf_end</code>	
...	
<code>_chain</code>	
	<code>_fileno</code>
...	
	<code>vtable</code>

```
_IO_new_file_xsputn (stdout, "Hello World", 11)
```

H e l l o W o r l d

\$ How buffered I/O works

<u>_IO_2_1_stdout_</u>	
<u>_flags</u>	<u>_IO_read_ptr</u>
<u>_IO_read_end</u>	<u>_IO_read_base</u>
<u>_IO_write_base</u>	<u>_IO_write_ptr</u>
<u>_IO_write_end</u>	<u>_IO_buf_base</u>
<u>_IO_buf_end</u>	
...	
<u>_chain</u>	
	<u>_fileno</u>
...	
	vtable

```
_IO_new_file_xsputn (stdout, "Hello World", 11)
```

return

H e l l o W o r l d

\$ How buffered I/O works

_IO_2_1_stdout_	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
_chain	
	_fileno
...	
vtable	

```
puts("");
```



\$ How buffered I/O works

_IO_2_1_stdout_	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
_chain	
	_fileno
...	
vtable	

```
puts("");
```



```
_IO_new_file_xsputn (stdout, "", 0)
```



```
return
```

H	e	l	l	o		W	o	r	l	d	
---	---	---	---	---	--	---	---	---	---	---	--

\$ How buffered I/O works

<code>_IO_2_1_stdout_</code>	
<code>_flags</code>	<code>_IO_read_ptr</code>
<code>_IO_read_end</code>	<code>_IO_read_base</code>
<code>_IO_write_base</code>	<code>_IO_write_ptr</code>
<code>_IO_write_end</code>	<code>_IO_buf_base</code>
<code>_IO_buf_end</code>	
...	
<code>_chain</code>	
	<code>_fileno</code>
...	
	<code>vtable</code>

```
puts("");
```



```
_IO_new_file_overflow (stdout, '\n');
```

H	e	l	l	o		W	o	r	l	d	
---	---	---	---	---	--	---	---	---	---	---	--

\$ How buffered I/O works

_IO_2_1_stdout_	
_flags	_IO_read_ptr
_IO_read_end	_IO_read_base
_IO_write_base	_IO_write_ptr
_IO_write_end	_IO_buf_base
_IO_buf_end	
...	
_chain	
	_fileno
...	
vtable	

```
puts("");
```



```
_IO_new_file_overflow (stdout, '\n');
```



```
_IO_do_write (stdout, "Hello World\n", 12);
```

H	e	l	l	o		W	o	r	l	d	
---	---	---	---	---	--	---	---	---	---	---	--

\$ How buffered I/O works

<code>_IO_2_1_stdout_</code>	
<code>_flags</code>	<code>_IO_read_ptr</code>
<code>_IO_read_end</code>	<code>_IO_read_base</code>
<code>_IO_write_base</code>	<code>_IO_write_ptr</code>
<code>_IO_write_end</code>	<code>_IO_buf_base</code>
<code>_IO_buf_end</code>	
...	
	<code>_chain</code>
	<code>_fileno</code>
...	
	<code>vtable</code>

```
puts("");
```



```
_IO_new_file_overflow (stdout, '\n');
```



```
_IO_do_write (stdout, "Hello World\n", 12);
```



```
sys_write (stdout._fileno, "Hello World\n", 12);
```

H	e	l	l	o		W	o	r	l	d	
---	---	---	---	---	--	---	---	---	---	---	--

\$ How buffered I/O works

<code>_IO_2_1_stdout_</code>	
<code>_flags</code>	<code>_IO_read_ptr</code>
<code>_IO_read_end</code>	<code>_IO_read_base</code>
<code>_IO_write_base</code>	<code>_IO_write_ptr</code>
<code>_IO_write_end</code>	<code>_IO_buf_base</code>
<code>_IO_buf_end</code>	
...	
	<code>_chain</code>
	<code>_fileno</code>
...	
	<code>vtable</code>

```
puts("");
```



```
_IO_new_file_overflow (stdout, '\n');
```



```
_IO_do_write (stdout, "Hello World\n", 12);
```



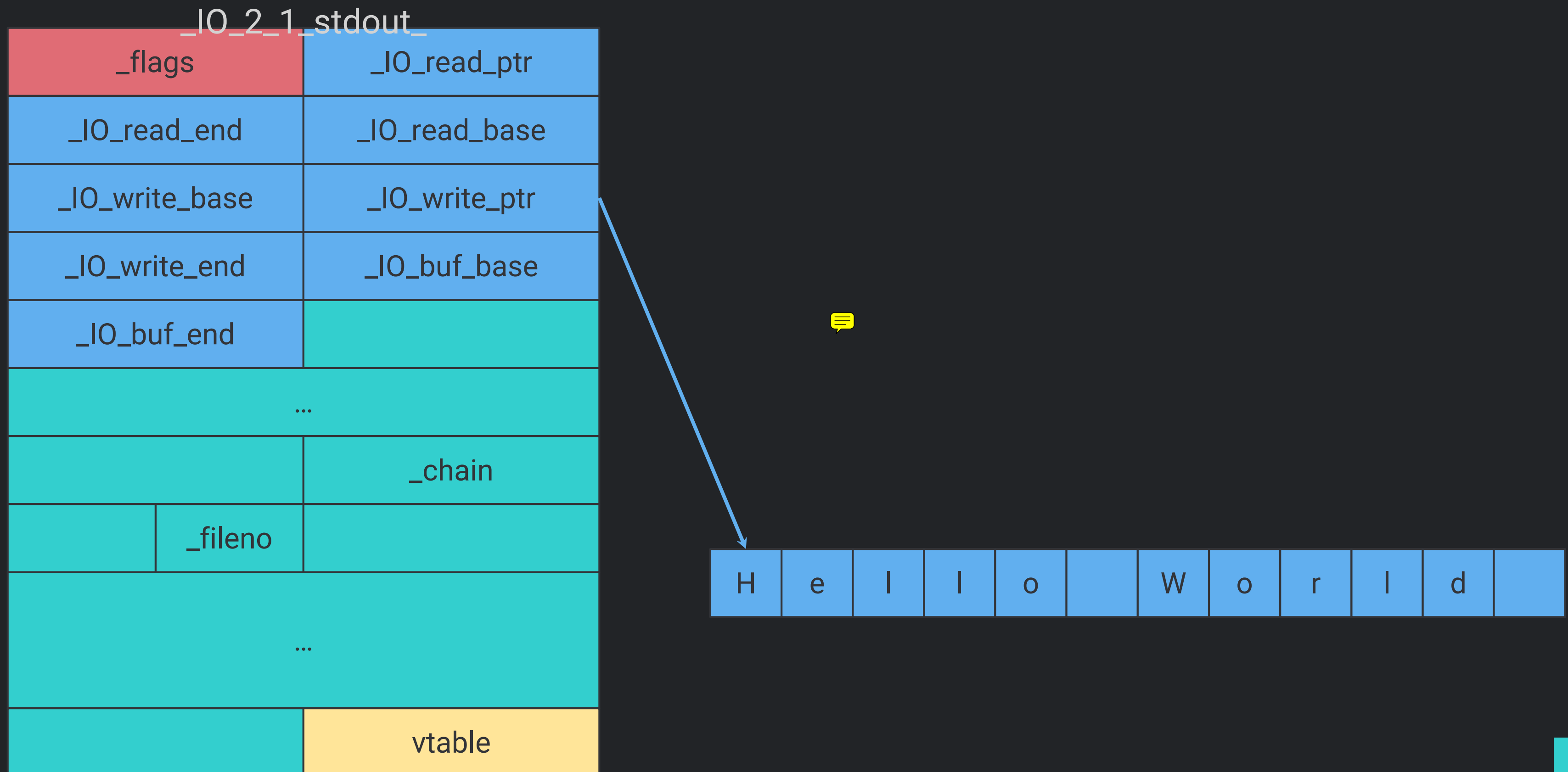
```
sys_write (stdout._fileno, "Hello World\n", 12);
```



Hello World

H	e	l	l	o		W	o	r	l	d	
---	---	---	---	---	--	---	---	---	---	---	--

\$ How buffered I/O works



ARBITRARY READ

\$ Arbitrary Read

- 想法
 - 將 **buffer** 的 **base** 設成 開始的位置
 - 將 **buffer** 的 **ptr** 設成 結束的位置
 - 在 **output** 的時候，將 **buffer** 內容 **output**
- Q:
 - **buffer base** 是什麼?
 - **buffer ptr** 是什麼?
 - 如何將 **buffer** 內容 **output**?

\$ _IO_file_xputn

- [libio/fileops.c](#)
- line buffered
 - 當輸出字串小於 **buffer** 剩餘空間
 - 找 **output** 的字串有沒有包含 **\n**

```
size_t
_IO_new_file_xputn (FILE *f, const void *data, size_t n)
{
    const char *s = (const char *) data;
    size_t to_do = n;
    int must_flush = 0;
    size_t count = 0;

    if (n <= 0)
        return 0;

    /* First figure out how much space is available in the buffer. */
    if ((f->_flags & _IO_LINE_BUF) && (f->_flags & _IO_CURRENTLY_PUTTING))
    {
        count = f->_IO_buf_end - f->_IO_write_ptr;
        if (count >= n) {
            const char *p;
            for (p = s + n; p > s; ) {
                if (*--p == '\n') {
                    count = p - s + 1;
                    must_flush = 1;
                    break;
                }
            }
        }
    }

    else if (f->_IO_write_end > f->_IO_write_ptr)
        count = f->_IO_write_end - f->_IO_write_ptr; /* Space available. */
```

\$ _IO_file_xsputn

- fully buffered / unbuffered
 - 計算 buffer 剩餘的空間

```
size_t
_IO_new_file_xsputn (FILE *f, const void *data, size_t n)
{
    const char *s = (const char *) data;
    size_t to_do = n;
    int must_flush = 0;
    size_t count = 0;

    if (n <= 0)
        return 0;

    /* First figure out how much space is available in the buffer. */
    if ((f->_flags & _IO_LINE_BUF) && (f->_flags & _IO_CURRENTLY_PUTTING))
    {
        count = f->_IO_buf_end - f->_IO_write_ptr;
        if (count >= n) {
            const char *p;
            for (p = s + n; p > s; ) {
                if (*--p == '\\n') {
                    count = p - s + 1;
                    must_flush = 1;
                    break;
                }
            }
        }
    }

    else if (f->_IO_write_end > f->_IO_write_ptr)
        count = f->_IO_write_end - f->_IO_write_ptr; /* Space available. */
}
```

\$ _IO_file_xsputn

- 把 buffer 填滿

```
size_t
_IO_new_file_xsputn (FILE *f, const void *data, size_t n)
{
    /* Then fill the buffer. */
    if (count > 0)
    {
        if (count > to_do)
            count = to_do;
        f->_IO_write_ptr = __mempcpy (f->_IO_write_ptr, s, count);
        s += count;
        to_do -= count;
    }

    if (to_do + must_flush > 0)
    {
        size_t block_size, do_write;
        /* Next flush the (full) buffer. */
        if (_IO_OVERFLOW (f, EOF) == EOF)
            /* If nothing else has to be written we must not signal the
               caller that everything has been written. */
            return to_do == 0 ? EOF : n - to_do;

        ...
    }
    return n - to_do;
}
```

\$ _IO_file_xputn

- 如果 **buffer** 空間不夠 / 遇到換行 (line buffered)

- **flush buffer**

- **_IO_OVERFLOW**

- **_IO_file_overflow**

```
size_t
_IO_new_file_xputn (FILE *f, const void *data, size_t n)
{
    ...
    /* Then fill the buffer. */
    if (count > 0)
    {
        if (count > to_do)
            count = to_do;
        f->_IO_write_ptr = __mempcpy (f->_IO_write_ptr, s, count);
        s += count;
        to_do -= count;
    }
    if (to_do + must_flush > 0)
    {
        size_t block_size, do_write;
        /* Next flush the (full) buffer. */
        if (_IO_OVERFLOW (f, EOF) == EOF)
            /* If nothing else has to be written we must not signal the
               caller that everything has been written. */
            return to_do == 0 ? EOF : n - to_do;
        ...
    }
    return n - to_do;
}
```


\$ _IO_file_overflow

- [libio/fileops.c](#)
- **_IO_NO_WRITES**
- **_flags &= ~_IO_NO_WRITES**

```
#define _IO_NO_WRITES          0x0008 /* Writing not allowed. */

int
_IO_new_file_overflow (FILE *f, int ch)
{
    if (f->_flags & _IO_NO_WRITES) /* SET ERROR */
    {
        ...
    }

    /* If currently reading or no buffer allocated. */
    if ((f->_flags & _IO_CURRENTLY_PUTTING) == 0 || f->_IO_write_base == NULL)
    {
        ...
    }

    if (ch == EOF)
        return _IO_do_write (f, f->_IO_write_base,
                             f->_IO_write_ptr - f->_IO_write_base);

    ...
}
```

\$ _IO_file_overflow

- [libio/fileops.c](#)
- 調整 buffer
- **_flags &= ~_IO_NO_WRITES**
- **_flags |= _IO_CURRENTLY_PUTTING**
- **_IO_write_base != NULL**

```
int
_IO_new_file_overflow (FILE *f, int ch)
{
    if (f->_flags & _IO_NO_WRITES) /* SET ERROR */
    {
        ...
    }

    /* If currently reading or no buffer allocated. */
    if ((f->_flags & _IO_CURRENTLY_PUTTING) == 0 || f->_IO_write_base == NULL)
    {
        ...
    }

    if (ch == EOF)
        return _IO_do_write (f, f->_IO_write_base,
                             f->_IO_write_ptr - f->_IO_write_base);

    ...
}
```

\$ _IO_file_overflow

- [libio/fileops.c](#)
- `_IO_do_write`
- `_flags &= ~_IO_NO_WRITES`
- `_flags |= _IO_CURRENTLY_PUTTING`
- `_IO_write_base != NULL`

```
int
_IO_new_file_overflow (FILE *f, int ch)
{
    if (f->_flags & _IO_NO_WRITES) /* SET ERROR */
    {
        ...
    }
    /* If currently reading or no buffer allocated. */
    if ((f->_flags & _IO_CURRENTLY_PUTTING) == 0 || f->_IO_write_base == NULL)
    {
        ...
    }
    if (ch == EOF)
        return _IO_do_write (f, f->_IO_write_base,
                             f->_IO_write_ptr - f->_IO_write_base);
    ...
}
```

\$ _IO_do_write

- [libio/fileops.c](#)


- **_flags &= ~_IO_NO_WRITES**
- **_flags |= _IO_CURRENTLY_PUTTING**
- **_IO_write_base != NULL**
- **_IO_read_end = _IO_write_base**

```
int
_IO_new_do_write (FILE *fp, const char *data, size_t to_do)
{
    return (to_do == 0
        || (size_t) new_do_write (fp, data, to_do) == to_do) ? 0 : EOF;
}

libc_hidden_ver (_IO_new_do_write, _IO_do_write)

static size_t
new_do_write (FILE *fp, const char *data, size_t to_do)
{
    size_t count;
    if (fp->_flags & _IO_IS_APPENDING)
        fp->_offset = _IO_pos_BAD;
    else if (fp->_IO_read_end != fp->_IO_write_base)
    {
        off64_t new_pos
        = _IO_SYSSEEK (fp, fp->_IO_write_base - fp->_IO_read_end, 1);
        if (new_pos == _IO_pos_BAD)
            return 0;
        fp->_offset = new_pos;
    }
    count = _IO_SYSWRITE (fp, data, to_do);
    ...
}
```

\$ Arbitrary Read - Summary

- Q:
 - buffer base 是什麼?
 - `_IO_write_base`
 - buffer ptr 是什麼?
 - `_IO_write_ptr`
 - 如何將 buffer 內容 output?
 - `_flags &= ~_IO_NO_WRITES`
 - `_flags |= _IO_CURRENTLY_PUTTING`
 - `_IO_write_base != NULL`
 - `_IO_read_end = _IO_write_base`
 - `_IO_write_end < _IO_write_ptr` 


\$ LAB - FILE Note (R)

ARBITRARY WRITE

\$ Arbitrary Write

- 想法
 - 在 **input** 的時候，**input** 內容會先寫到 **buffer**
 - 將 **buffer** 的 **base** 設成 開始的位置
 - 將 **buffer** 的 **end** 設成 結束的位置
- Q:
 - **buffer base** 是什麼?
 - **buffer end** 是什麼?
 - 如何將 **input** 寫到 **buffer**?

\$ input functions

- scanf
 - _IO_file_underflow ([libio/fileops.c](#))
- fread 
 - _IO_file_xsgetn ([libio/fileops.c](#))
 - _IO_file_underflow
- fgets
 - _IO_getline ([libio/iogetline.c](#))
 - _IO_getline_info ([libio/iogetline.c](#))
 - _IO_file_underflow

\$ _IO_file_underflow

- [libio/fileops.c](#)
- 填滿 buffer

\$ _IO_file_underflow

- Check flags
- `_flags &= ~_IO_EOF_SEEN`
- `_flags &= ~_IO_NO_READS`

```
int
_IO_new_file_underflow (FILE *fp)
{
    ssize_t count;

    /* C99 requires EOF to be "sticky" */
    if (fp->_flags & _IO_EOF_SEEN)
        return EOF;

    if (fp->_flags & _IO_NO_READS)
    {
        fp->_flags |= _IO_ERR_SEEN;
        __set_errno (EBADF);
        return EOF;
    }

    if (fp->_IO_read_ptr < fp->_IO_read_end)
        return *(unsigned char *) fp->_IO_read_ptr;

    if (fp->_IO_buf_base == NULL)
    {
        /* Maybe we already have a push back pointer. */
        if (fp->_IO_save_base != NULL)
        {
            free (fp->_IO_save_base);
            fp->_flags &= ~_IO_IN_BACKUP;
        }

        _IO_doallocbuf (fp);
    }
}
```

\$ _IO_file_underflow

- Buffer 內還有 data
- `_flags &= ~_IO_EOF_SEEN`
- `_flags &= ~_IO_NO_READS`
- `_IO_read_ptr >= _IO_read_end`

```
int
_IO_new_file_underflow (FILE *fp)
{
    ssize_t count;

    /* C99 requires EOF to be "sticky". */
    if (fp->_flags & _IO_EOF_SEEN)
        return EOF;

    if (fp->_flags & _IO_NO_READS)
    {
        fp->_flags |= _IO_ERR_SEEN;
        __set_errno (EBADF);
        return EOF;
    }

    if (fp->_IO_read_ptr < fp->_IO_read_end)
        return *(unsigned char *) fp->_IO_read_ptr;

    if (fp->_IO_buf_base == NULL)
    {
        /* Maybe we already have a push back pointer. */
        if (fp->_IO_save_base != NULL)
        {
            free (fp->_IO_save_base);
            fp->_flags &= ~_IO_IN_BACKUP;
        }

        _IO_doallocbuf (fp);
    }
}
```

\$ _IO_file_underflow

- 沒有 buffer
- _flags &= ~_IO_EOF_SEEN
- _flags &= ~_IO_NO_READS
- _IO_read_ptr >= _IO_read_end
- _IO_buf_base != NULL

```
int
_IO_new_file_underflow (FILE *fp)
{
    ssize_t count;

    /* C99 requires EOF to be "sticky". */
    if (fp->_flags & _IO_EOF_SEEN)
        return EOF;

    if (fp->_flags & _IO_NO_READS)
    {
        fp->_flags |= _IO_ERR_SEEN;
        __set_errno (EBADF);
        return EOF;
    }

    if (fp->_IO_read_ptr < fp->_IO_read_end)
        return *(unsigned char *) fp->_IO_read_ptr;
```

```
if (fp->_IO_buf_base == NULL)
{
    /* Maybe we already have a push back pointer. */
    if (fp->_IO_save_base != NULL)
    {
        free (fp->_IO_save_base);
        fp->_flags &= ~_IO_IN_BACKUP;
    }

    _IO_doallocbuf (fp);
}
```

\$ _IO_file_underflow

- flush stdout
- _flags &= ~_IO_EOF_SEEN
- _flags &= ~_IO_NO_READS
- _IO_read_ptr >= _IO_read_end
- _IO_buf_base != NULL

```
int
_IO_new_file_underflow (FILE *fp)
{
```

```
    if (fp->_flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
    {
        _IO_acquire_lock (stdout);

        if ((stdout->_flags & (_IO_LINKED | _IO_NO_WRITES | _IO_LINE_BUF))
            == (_IO_LINKED | _IO_LINE_BUF))
            _IO_OVERFLOW (stdout, EOF);

        _IO_release_lock (stdout);
    }
```

```
    _IO_switch_to_get_mode (fp);
```

```
    fp->_IO_read_base = fp->_IO_read_ptr = fp->_IO_buf_base;
    fp->_IO_read_end = fp->_IO_buf_base;
    fp->_IO_write_base = fp->_IO_write_ptr = fp->_IO_write_end
        = fp->_IO_buf_base;
```

```
    count = _IO_SYSREAD (fp, fp->_IO_buf_base,
                        fp->_IO_buf_end - fp->_IO_buf_base);
```

```
    if (count <= 0)
    {
```

```
        if (count == 0)
            fp->_flags |= _IO_EOF_SEEN;
```

\$ _IO_file_underflow

- 從 `_IO_buf_base` 讀到 `_IO_buf_end`
- `_flags &= ~_IO_EOF_SEEN`
- `_flags &= ~_IO_NO_READS`
- `_IO_read_ptr >= _IO_read_end`
- `_IO_buf_base != NULL`

```
int
_IO_new_file_underflow (FILE *fp)
{
    ...
    if (fp->_flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
    {
        _IO_acquire_lock (stdout);

        if ((stdout->_flags & (_IO_LINKED | _IO_NO_WRITES | _IO_LINE_BUF))
            == (_IO_LINKED | _IO_LINE_BUF))
            _IO_OVERFLOW (stdout, EOF);

        _IO_release_lock (stdout);
    }

    _IO_switch_to_get_mode (fp);

    fp->_IO_read_base = fp->_IO_read_ptr = fp->_IO_buf_base;
    fp->_IO_read_end = fp->_IO_buf_base;
    fp->_IO_write_base = fp->_IO_write_ptr = fp->_IO_write_end
        = fp->_IO_buf_base;

    count = _IO_SYSREAD (fp, fp->_IO_buf_base,
                        fp->_IO_buf_end - fp->_IO_buf_base);

    ...
}

libc_hidden_ver (_IO_new_file_underflow, _IO_file_underflow)
```

\$ Arbitrary Write - Summary

- Q:
 - **buffer base** 是什麼?
 - `_IO_buf_base`
 - **buffer end** 是什麼?
 - `_IO_buf_end`
 - 如何將 **input** 寫到 **buffer**?
 - `_flags &= ~_IO_EOF_SEEN`
 - `_flags &= ~_IO_NO_READS`
 - `_IO_read_ptr >= _IO_read_end`
 - `_IO_buf_base != NULL`

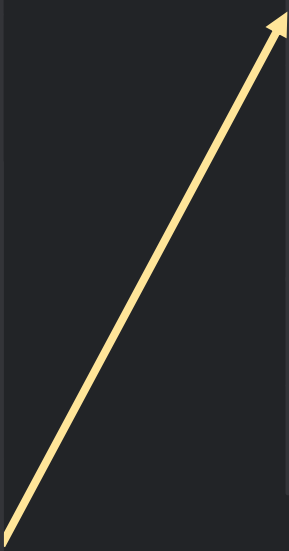
\$ LAB - FILE Note (W)

HIJACK VTABLE

\$ Hijack vtable

- 直接改 **vtable pointer**?

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		vtable



__dummy	__finish
__overflow	__underflow
__uflow	__pbackfail
__xspn	__xsgetn
...	

\$ Hijack vtable

- 直接改 **vtable pointer?**

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		fake vtable

one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
...	

__dummy	__finish
__overflow	__underflow
__uflow	__pbackfail
__xspn	__xsgetn
...	

\$ Hijack vtable

- 直接改 **vtable pointer?**
 - **GLIBC 2.24** 以前可以
- **GLIBC 2.24** 以後
 - **vtable verification**

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		fake vtable

one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
...	

__dummy	__finish
__overflow	__underflow
__uflow	__pbackfail
__xspn	__xsgetn
...	

\$ vtable verification

- 我們是如何 call 到這些 vtable 中的 function pointer 的呢？

```
int
_IO_puts (const char *str)
{
    int result = EOF;
    size_t len = strlen (str);
    _IO_acquire_lock (stdout);

    if ((_IO_vtable_offset (stdout) != 0
        || _IO_fwide (stdout, -1) == -1)
        && !_IO_sputn (stdout, str, len) == len
        && !_IO_putc_unlocked ('\n', stdout) != EOF)
        result = MIN (INT_MAX, len + 1);

    _IO_release_lock (stdout);
    return result;
}
weak_alias (_IO_puts, puts)
```

\$ vtable verification

- 我們是如何 call 到這些 vtable 中的 function pointer 的呢？

```
#define _IO_sputn(__fp, __s, __n) _IO_XSPUTN (__fp, __s, __n)
```

```
#define _IO_XSPUTN(FP, DATA, N) JUMP2 (__xspn, FP, DATA, N)
```

```
#define JUMP2(FUNC, THIS, X1, X2) (_IO_JUMPS_FUNC(THIS)->FUNC) (THIS, X1, X2)
```

```
#define _IO_JUMPS_FUNC(THIS) (IO_validate_vtable (_IO_JUMPS_FILE_plus (THIS)))
```

\$ vtable verification

- 檢查 vtable 是否在 GLIBC 預設的 vtable 範圍中

```
/* Perform vtable pointer validation.  If validation fails, terminate  
the process.  */  
static inline const struct _IO_jump_t *  
IO_validate_vtable (const struct _IO_jump_t *vtable)  
{  
    /* Fast path: The vtable pointer is within the __libc_IO_vtables  
section.  */  
    uintptr_t section_length = __stop__libc_IO_vtables - __start__libc_IO_vtables;  
    uintptr_t ptr = (uintptr_t) vtable;  
    uintptr_t offset = ptr - (uintptr_t) __start__libc_IO_vtables;  
    if (__glibc_unlikely (offset >= section_length))  
        /* The vtable pointer is not in the expected section.  Use the  
slow path, which will terminate the process if necessary.  */  
        _IO_vtable_check ();  
    return vtable;  
}
```


\$ vtable verification

- Bypass?
 - 不太實際

```
void attribute_hidden
_IO_vtable_check (void)
{
    void (*flag) (void) = atomic_load_relaxed (&IO_accept_foreign_vtables);
    PTR_DEMANGLE (flag);
    if (flag == &_amp;_IO_vtable_check)
        return;

    {
        Dl_info di;
        struct link_map *l;
        if (!rtld_active ())
            || (_dl_addr (_IO_vtable_check, &di, &l, NULL) != 0
                && l->l_ns != LM_ID_BASE))
            return;
    }

    __libc_fatal ("Fatal error: glibc detected an invalid stdio handle\n");
}
```

\$ Hijack vtable

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		fake vtable

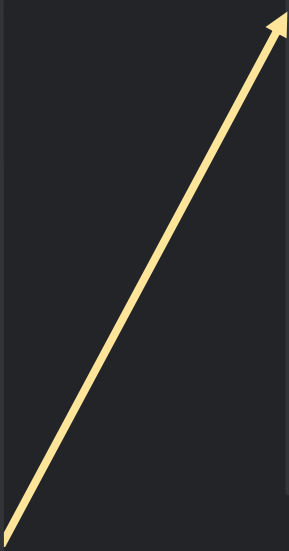
one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
...	

__dummy	__finish
__overflow	__underflow
__uflow	__pbackfail
__xspn	__xsgetn
...	

\$ Hijack vtable

- 改 **vtable** 中的 **pointer**?

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		vtable



__dummy	__finish
__overflow	__underflow
__uflow	__pbackfail
__xspn	__xsgetn
...	

\$ Hijack vtable

- 改 **vtable** 中的 **pointer**?
 - 可以
 - **GLIBC 2.29** 以後

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		vtable

one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
...	

\$ LAB - FILE Note (X)

MISC

\$ FSOP

- File-Stream Oriented Programming
- GLIBC version < 2.24

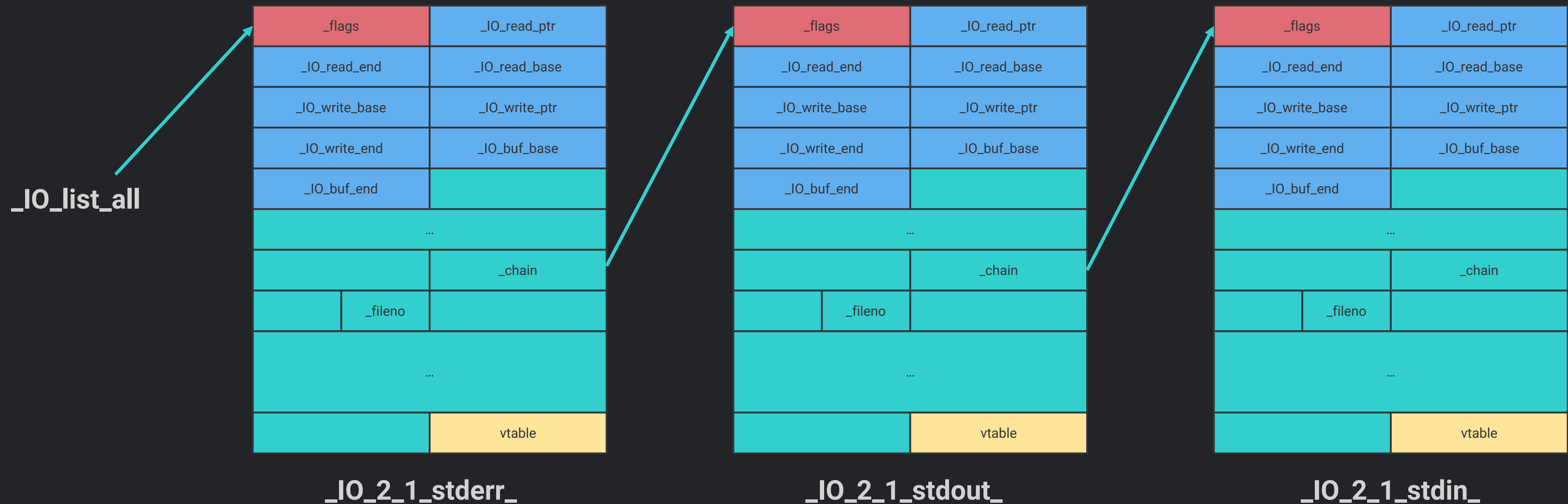
\$ FSOP

- 還記得前面的 **_chain** 嗎?

_flags		_IO_read_ptr
_IO_read_end		_IO_read_base
_IO_write_base		_IO_write_ptr
_IO_write_end		_IO_buf_base
_IO_buf_end		
...		
		_chain
	_fileno	
...		
		vtable

\$ FSOP

- 所有的 **FILE** 結構會透過 **_chain** 串成一個 **list**



\$ FSOP

- 在程式結束時
 - 會 call `_IO_flush_all_lockp` 這個 function
 - flush `_IO_list_all` 這個 list
- `_IO_flush_all_lockp` ([libio/genops.c](#))

```
int
_IO_flush_all_lockp (int do_lock)
{
    int result = 0;
    FILE *fp;
    ...
    for (fp = (FILE *) _IO_list_all; fp != NULL; fp = fp->_chain)
    {
        run_fp = fp;
        ...

        if (((fp->_mode <= 0 && fp->_IO_write_ptr > fp-
>_IO_write_base)
            || (_IO_vtable_offset (fp) == 0
                && fp->_mode > 0 && (fp->_wide_data->_IO_write_ptr
                    > fp->_wide_data->_IO_write_base))
            )
            && _IO_OVERFLOW (fp, EOF) == EOF)
        result = EOF;

        ...
    }
    ...
}
```

\$ FSOP

- 在程式結束時
 - 會 call `_IO_flush_all_lockp` 這個 function
 - flush `_IO_list_all` 這個 list
- `_IO_flush_all_lockp` ([libio/genops.c](#))

```
int
_IO_flush_all_lockp (int do_lock)
{
    int result = 0;
    FILE *fp;

    ...
    for (fp = (FILE *) _IO_list_all; fp != NULL; fp = fp->_chain)
    {
        run_fp = fp;
        ...

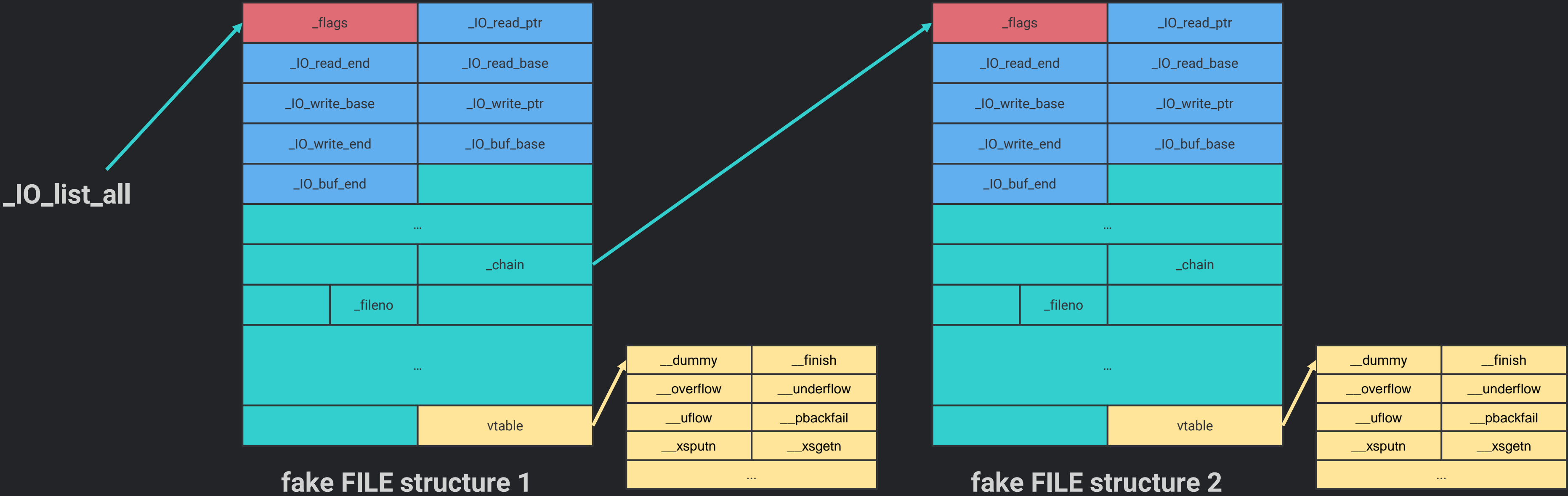
        if (((fp->_mode <= 0 && fp->_IO_write_ptr > fp->_IO_write_base)
            || (_IO_vtable_offset (fp) == 0
                && fp->_mode > 0 && (fp->_wide_data->_IO_write_ptr
                    > fp->_wide_data->_IO_write_base)))

            && _IO_OVERFLOW (fp, EOF) != EOF)
        result = EOF;

        ...
    }
    ...
}
```

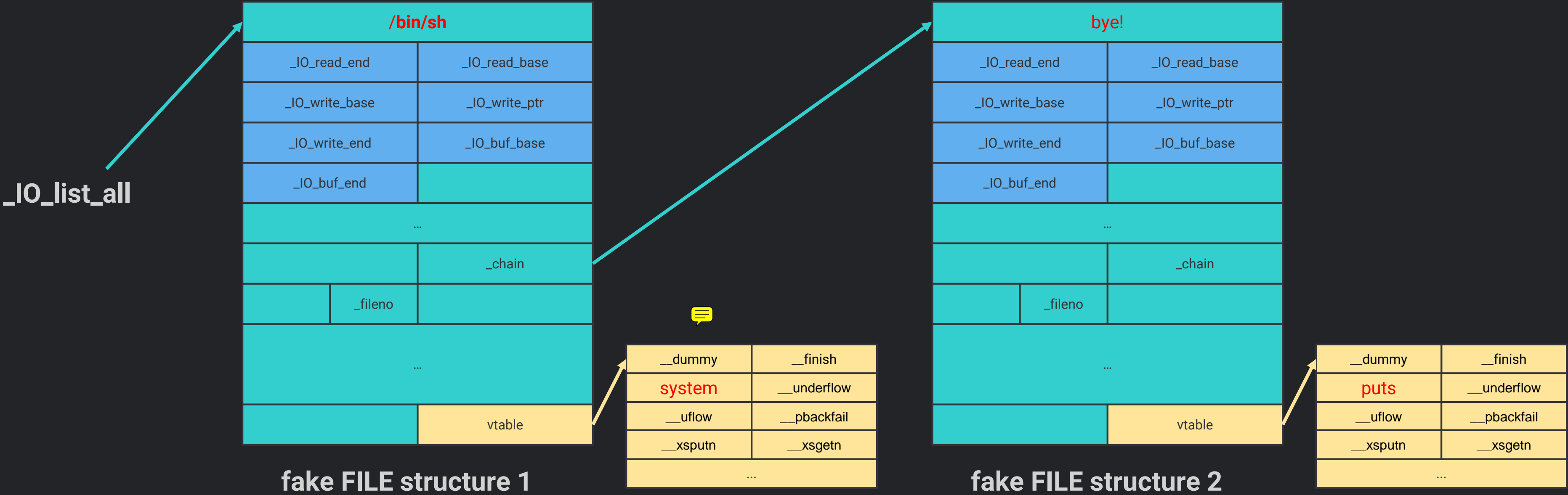
\$ FSOP

- 將 `_IO_list_all` 改成偽造的結構



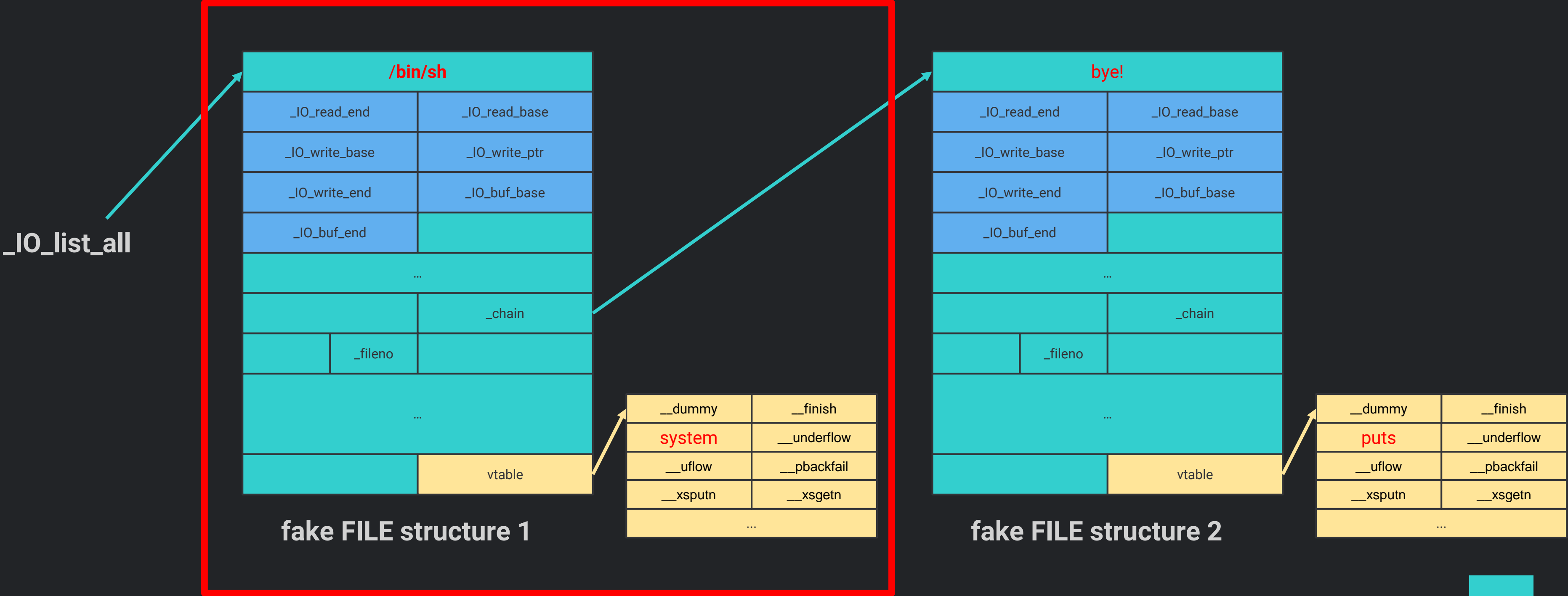
\$ FSOP

- 將 `_IO_list_all` 改成偽造的結構



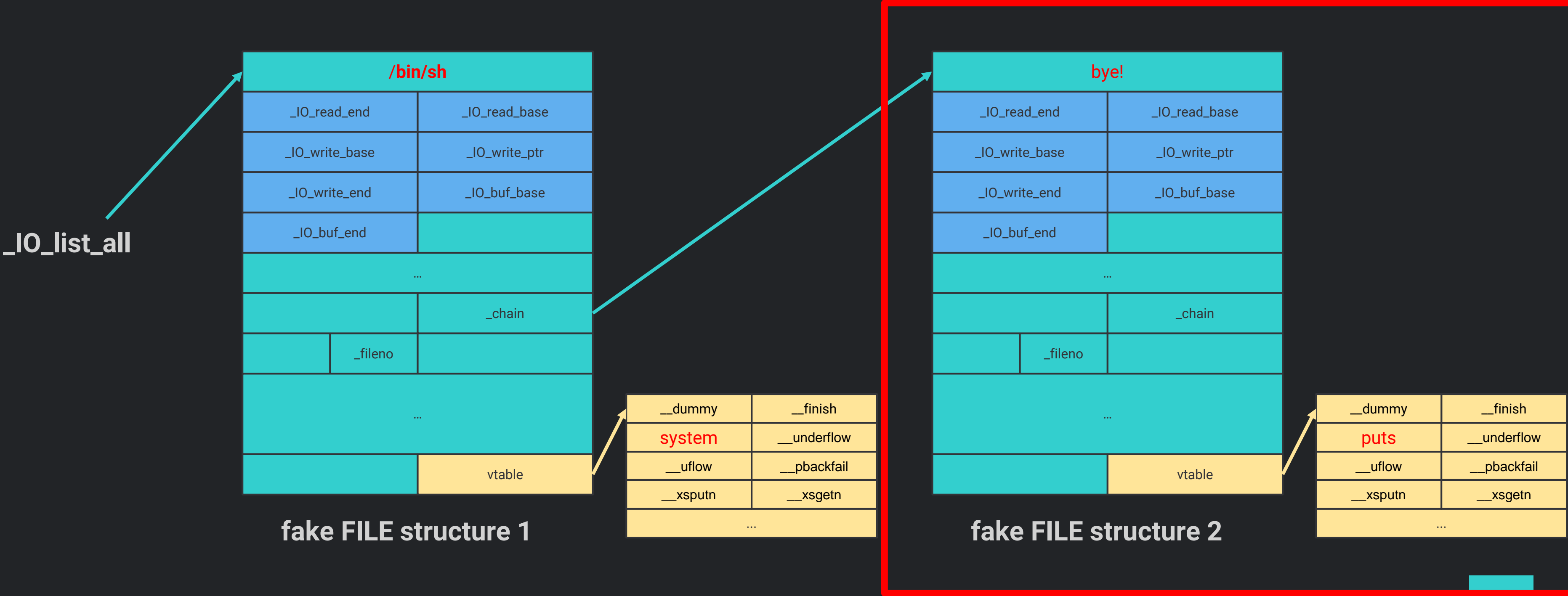
\$ FSOP

- 將 `_IO_list_all` 改成偽造的結構



\$ FSOP

- 將 `_IO_list_all` 改成偽造的結構

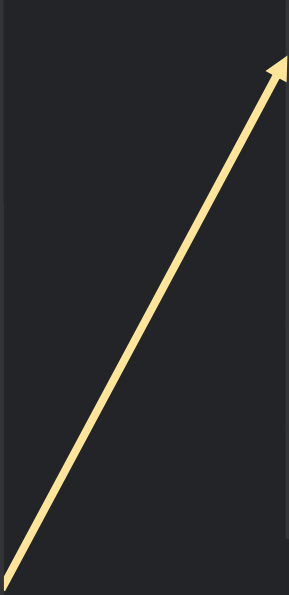


\$ _IO_str_jumps vtable

- 不能自己偽造 **vtable**
- 改不到 **vtable entry**

- 利用既有的 **vtable**
 - **_IO_str_jump**

_flags		_IO_read_ptr	
_IO_read_end		_IO_read_base	
_IO_write_base		_IO_write_ptr	
_IO_write_end		_IO_buf_base	
_IO_buf_end			
...			
		_chain	
	_fileno		
...			
		vtable	



one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
one gadget	one gadget
...	

\$ _IO_str_jumps vtable

- GLIBC < 2.28
- `_IO_strfile_` 這個結構所用的 **vtable**
- 結構裡面有 **function pointer**

```
typedef struct _IO_strfile_  
{  
    struct _IO_streambuf _sbf;  
    struct _IO_str_fields _s;  
} _IO_strfile;
```

```
struct _IO_str_fields  
{  
    _IO_alloc_type _allocate_buffer;  
    _IO_free_type _free_buffer;  
};
```

```
typedef void *(*_IO_alloc_type) (_IO_size_t);  
typedef void (*_IO_free_type) (void*);
```

\$ _IO_str_jumps vtable

- GLIBC < 2.28
- _IO_strfile_ 這個結構所用的 vtable
- vtable 中的 function 會去用到結構中的 function pointer

```
void
_IO_wstr_finish (_IO_FILE *fp, int dummy)
{
    if (fp->wide_data->_IO_buf_base && !(fp->flags2 & _IO_FLAGS2_USER_WBUF))
        (((_IO_strfile *) fp)->_s._free_buffer) (fp->_wide_data->_IO_buf_base);
    fp->_wide_data->_IO_buf_base = NULL;

    _IO_wdefault_finish (fp, 0);
}
```

\$ _IO_str_jumps vtable

- GLIBC < 2.28
- Exploit
 - 把 **vtable** 改到 **_IO_str_jumps**
 - 偽造結構中的 **function pointer**

```
void
_IO_wstr_finish (_IO_FILE *fp, int dummy)
{
    if (fp->wide_data->_IO_buf_base && !(fp->flags2 & _IO_FLAGS2_USER_WBUF))
        (((_IO_strfile *) fp)->_s._free_buffer) (fp->_wide_data->_IO_buf_base);
    fp->_wide_data->_IO_buf_base = NULL;

    _IO_wdefault_finish (fp, 0);
}
```

\$ HW - FILE Note

**THANK YOU FOR
LISTENING!**

ANY QUESTIONS?