Heap Exploitation

ddaa

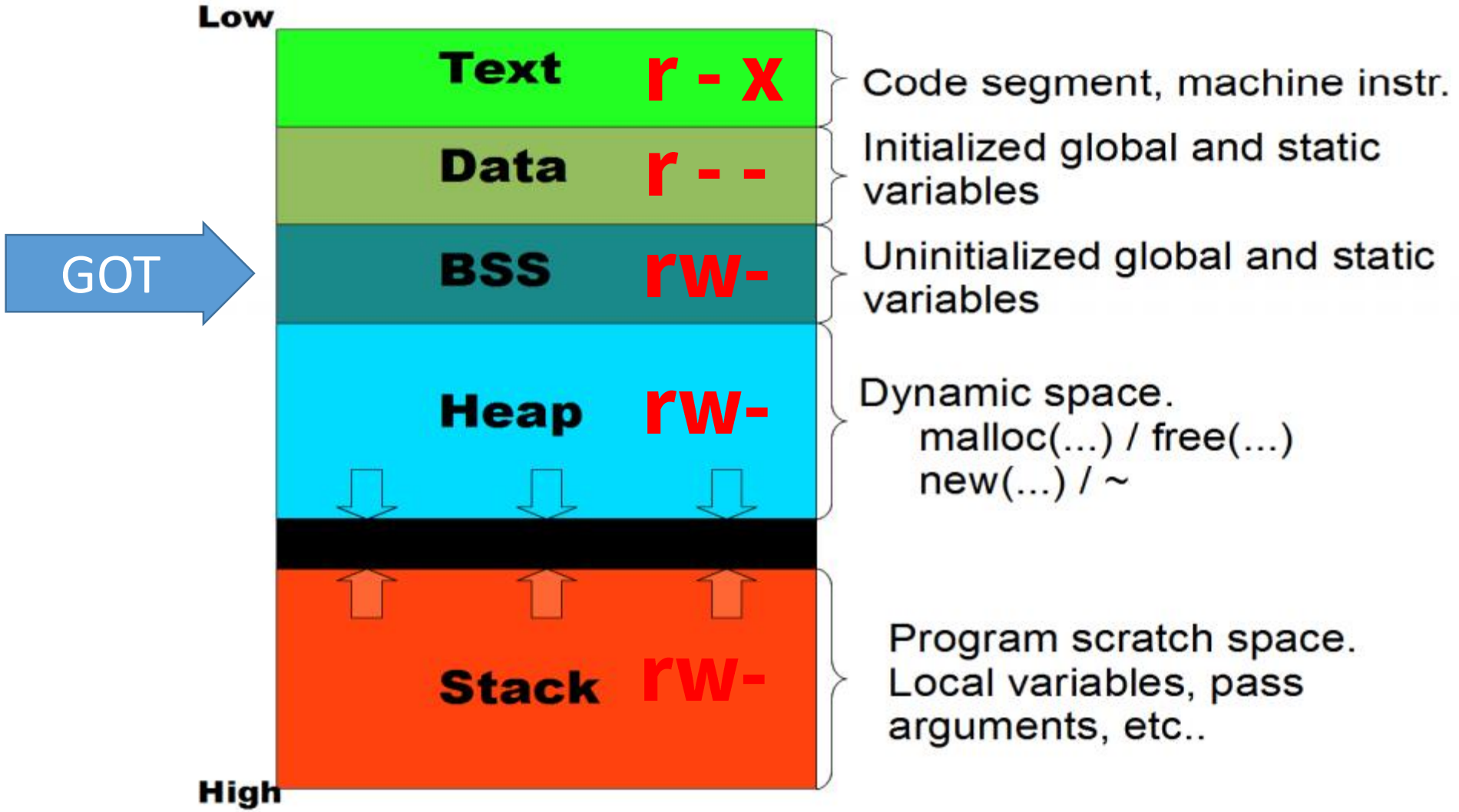
Review – buffer overflow

Buffer Overflow

- 在對變數做操作時,沒有檢查邊界就直接寫入
  - 後面的資料被覆蓋
  - 程式Crash

```
cychao@CatKali:~/ctf/nctu/slide$ ./foo aaaaaaaaaaaaaaaaaaaaaa
Segmentation fault
cychao@CatKali:~/ctf/nctu/slide$
```

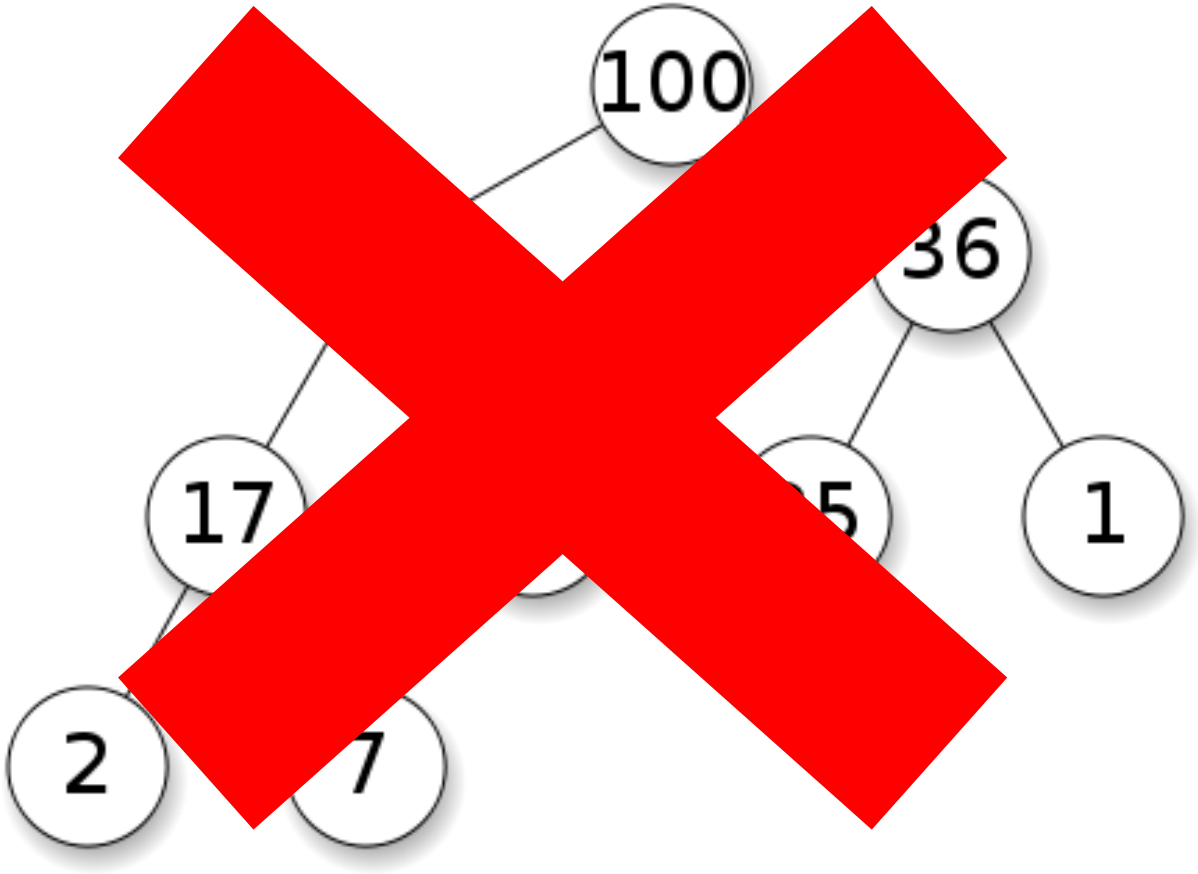
Virtual memory



名詞解釋

- Buffer Overflow
  - 在對變數做操作時，沒檢查邊界就直接寫入，導致後面的資料被覆蓋
  - (投影片後面稱 overflow 或 bof)
- Stack Overflow
  - 在 stack 段發生 overflow，導致 stack 中的變數資料被覆蓋
  - return address 被覆蓋 => eip 可控
- Heap Overflow
  - 在 heap 段發生 overflow，導致 heap 中的變數資料被覆蓋
  - 早期的 heap overflow 是指 doug lea malloc 的機制引起的問題
    - http://en.wikipedia.org/wiki/Heap_overflow
  - 不能直接控制 eip，是因為 bof 後 glibc 進行記憶體配置發生問題才可控 eip

What is heap?



What is heap?

- 有時資料需要在 runtime 依據實際情況動態增減，因此不適合在 stack 儲存
- 此時系統就會將資料儲存至另一塊空間，稱之為 heap
- 在 linux 的環境中，由 glibc 內部實做 memory allocator，programmer 使用 malloc(), free() 去動態配置和釋放記憶體
- libc 裡是 ptmalloc，是基於 dlmalloc 實作的
- Reference
  - http://g.oswego.edu/dl/html/malloc.html

How it work?

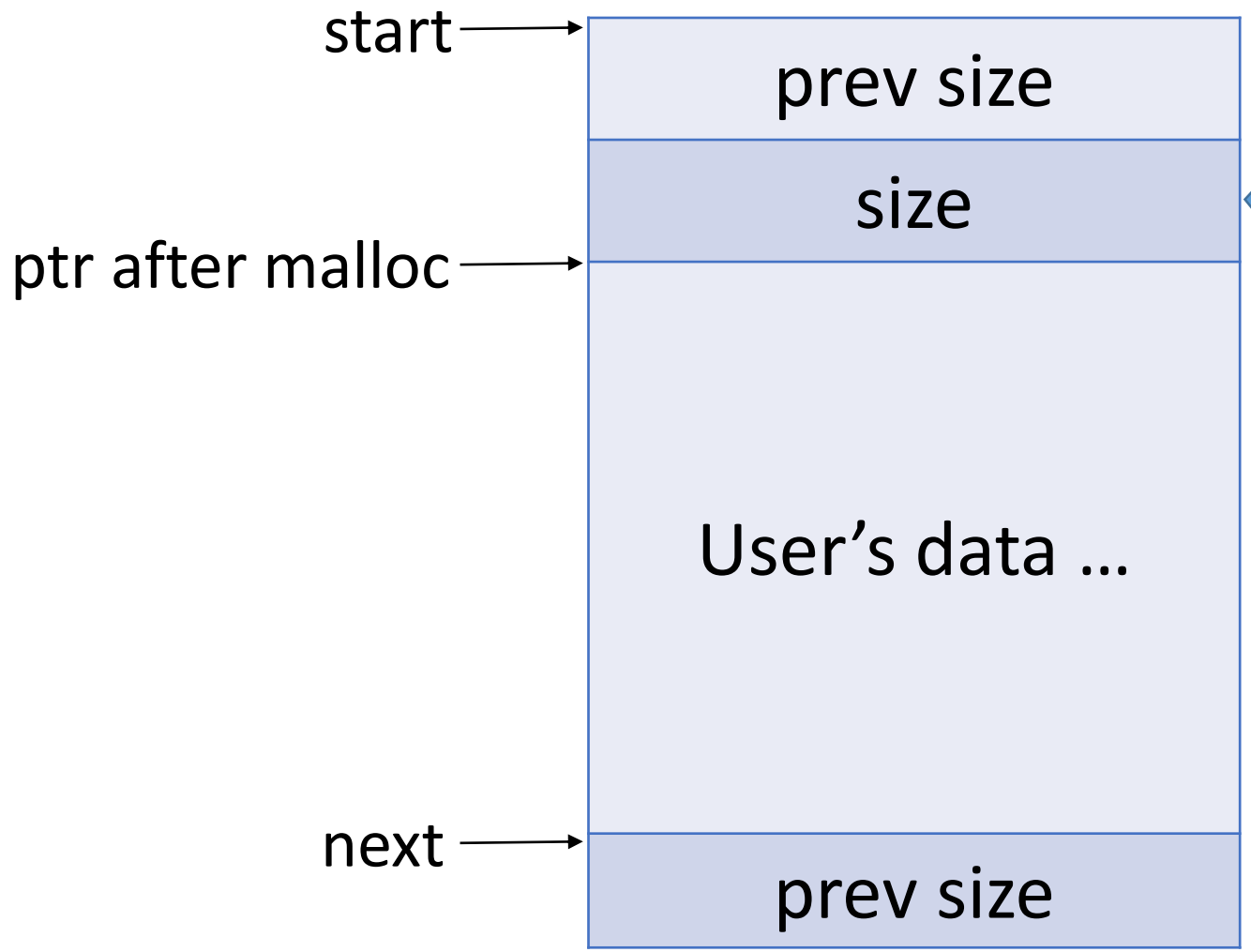
- chunk = prev_size + size +配置的記憶體
- 用多個 linked list管理可用的 chunk
  - linked list 的 head 稱為 bin
- chunk 會根據 chunk size 分類，以不同的 bin 去紀錄可用的 chunk
- malloc 時 bin 裡不一定有可以用的 chunk
  - No freed chunk
  - No suitable chunk
- malloc 會使用 brk() 增加 data segment 的大小，並配置新的 chunk
  - 原則上，chunk 在記憶體中是連續的

Merge freed chunk

- 為了避免太多破碎的 chunk，如果 freed chunk 在記憶體中是連續的，則合併起來
- 合併後呼叫 unlink 移除 bin 中重複的 chunk
- heap overflow能造成記憶體任意寫入的主因

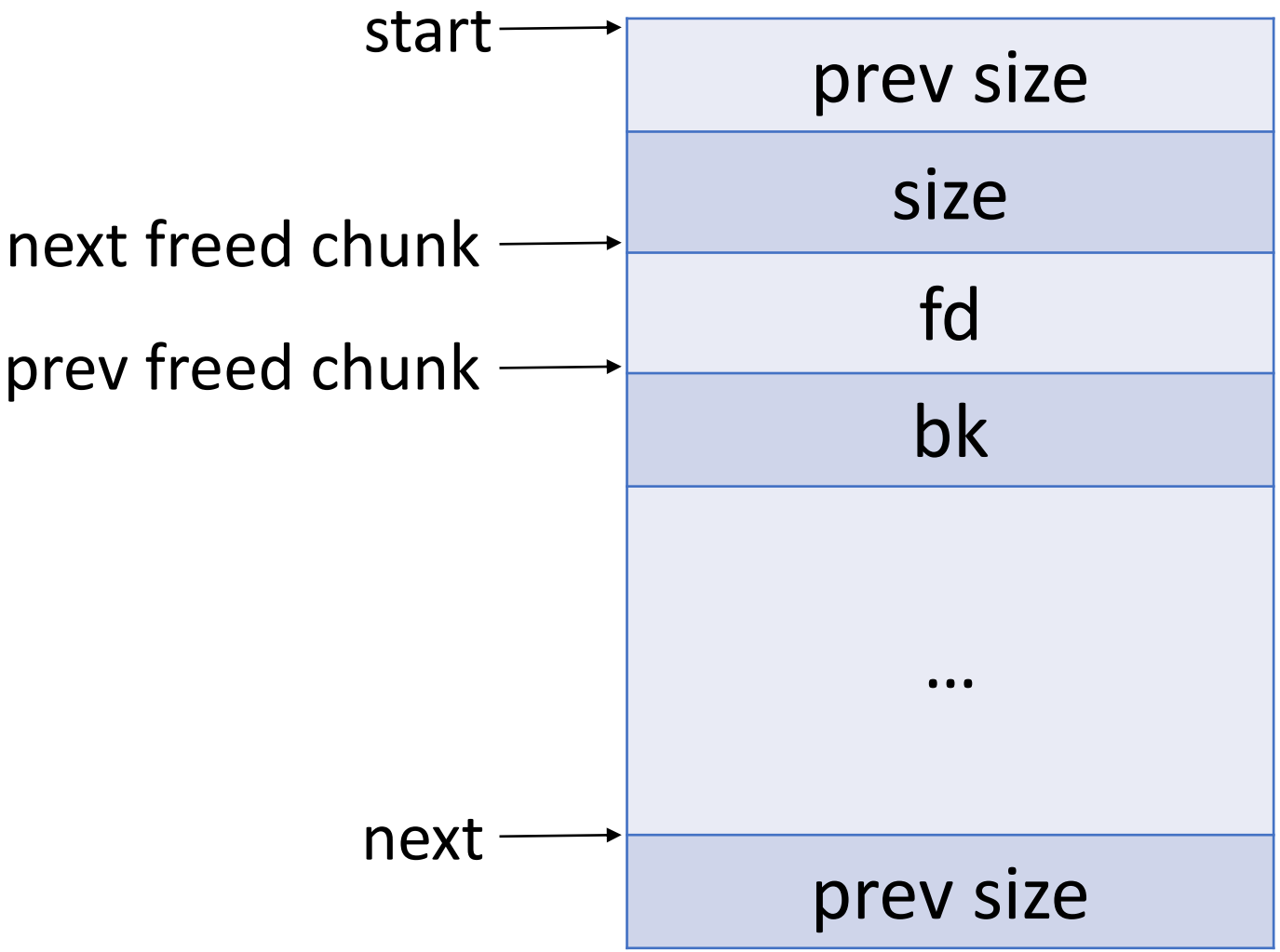
```
#define unlink(P, BK, FD) {
    FD = P->fd;
    BK = P->bk;
    FD->bk = BK;
    BK->fd = FD;
}
```

Chunk (inuse)



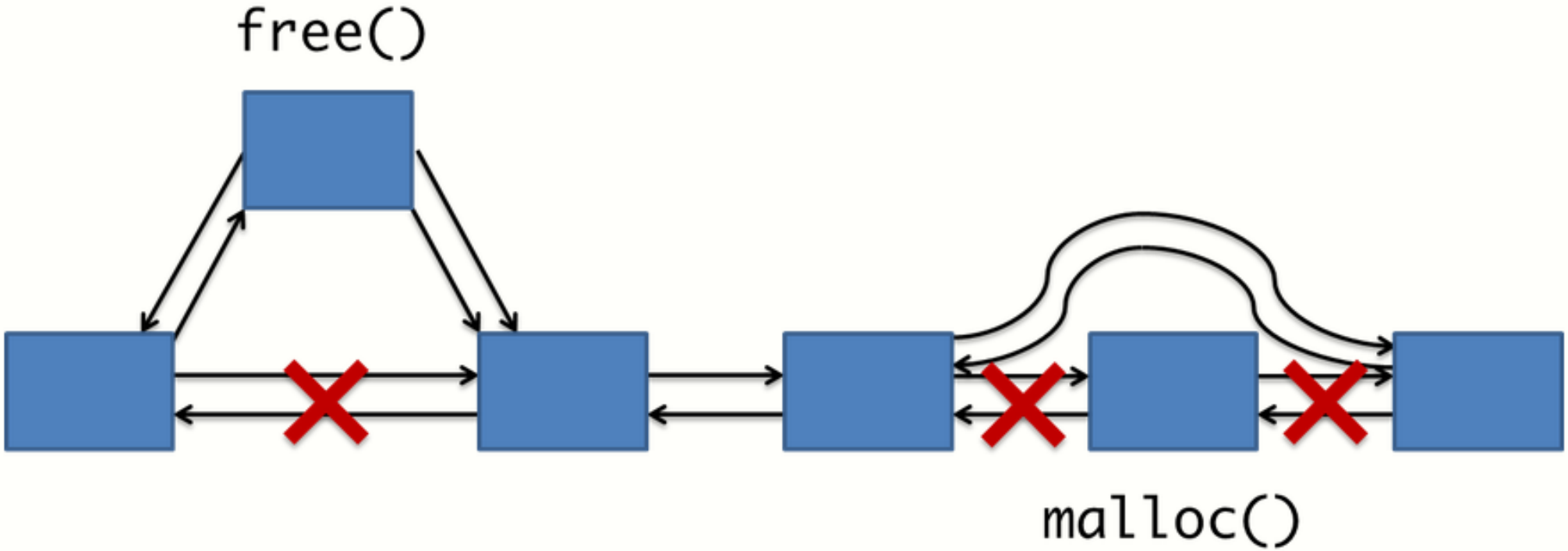
- Size 最低的 3 bits 有特殊用途
- PREV_INUSE 0x1 前一個 chunk 是使用中 (非 freed)
- IS_MMAPPED 0x2 這個 chunk 是用 mmap() 建立的
- NON_MAIN_ARENA 0x4 這個 chunk 所在的 heap 是不是在 main_arena 內

Chunk (freed)


start
prev size
size
next freed chunk
fd
prev freed chunk
bk
...
next
prev size

malloc & free

- malloc 從 list 裡拿出可用的 chunk；free 時放回 list



Heap Overflow (Doug Lea's malloc)

- 紀錄 heap 的各種 structures、meta-data 和 buffer 是利用相同的 memory area
- 如果發生 heap segmentation 發生 overflow，在 malloc 和 free 等操作可能會發生任意讀寫 address 的問題
- 如果前一個 malloc 出的 chunk 發生 bof，覆蓋後一個 chunk 的structure，在 free 這兩段 chunk 時就會發生 heap overflow 的問題，造成可以寫入任意 address

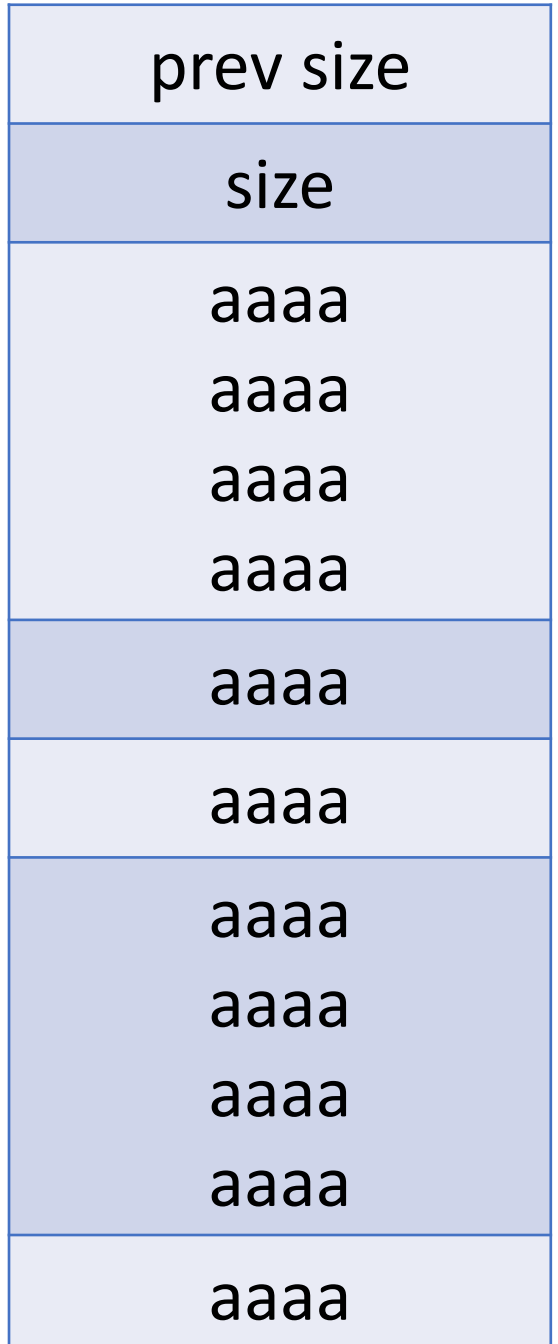
※目前 glibc 版本已經修正此問題

Example

```
int main() {
        char *p, *q;
        p = malloc(256);
        q = malloc(256);
        gets(p);
        free(q);
        free(p);
        return 0;
}
```

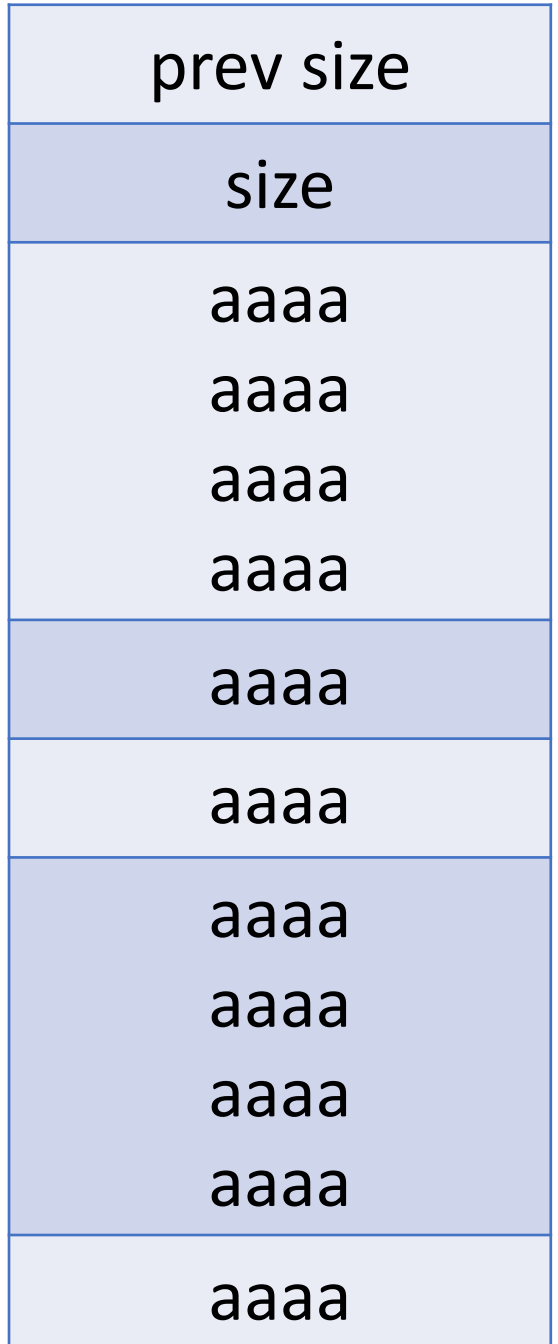
Overflow

```
int main() {
        char *p, *q;
        p = malloc(256);
        q = malloc(256);
        gets(p);
        free(q);
        free(p);
        return 0;
}
```



Free first chunk

```
int main() {
        char *p, *q;
        p = malloc(256);
        q = malloc(256);
        gets(p);
        free(q);
        free(p);
        return 0;
}
```

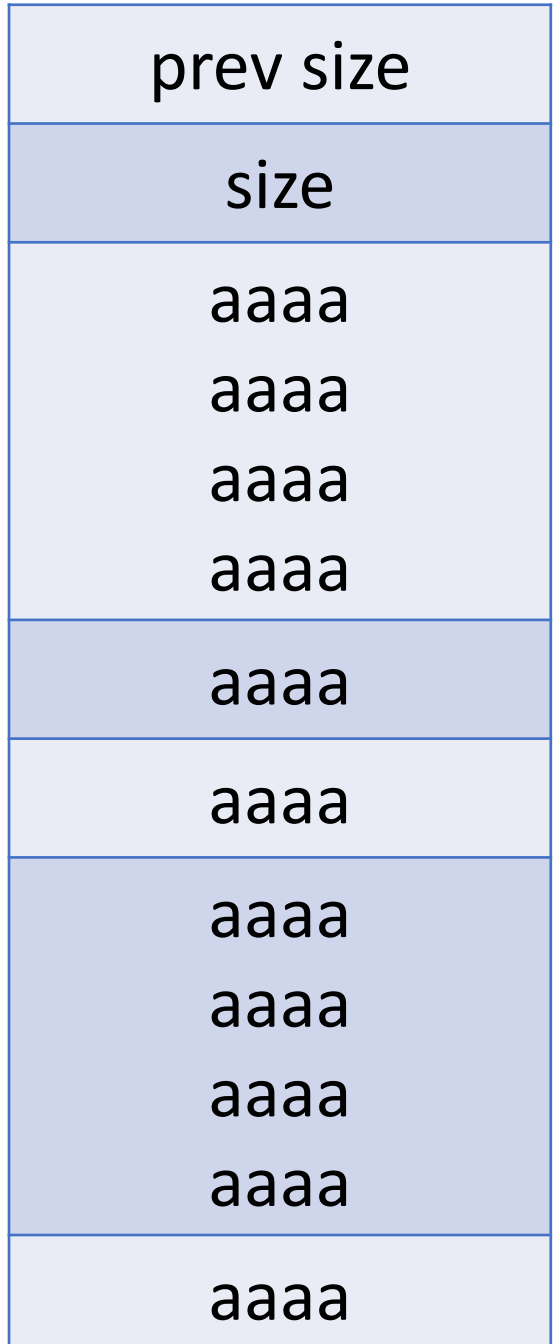


Free second chunk

```
int main() {
        char *p, *q;
        p = malloc(256);
        q = malloc(256);
        gets(p);
        free(q);
        free(p);
        return 0;
}
```

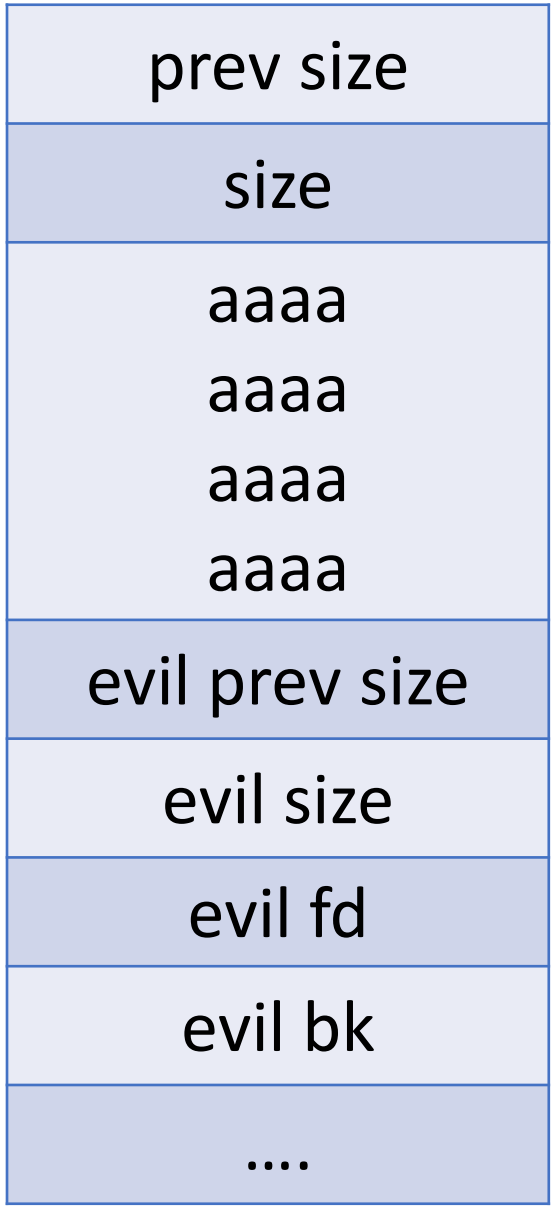
Segmentation fault!

mov eax, DWORD PTR [eax+0x4]



Example

```
int main() {
        char *p, *q;
        p = malloc(256);
        q = malloc(256);
        gets(p);
        free(q);
        free(p);
        return 0;
}
```



Demo – Quals Baby's First heap

- http://ddaa.logdown.com/posts/200443-defcon-22-quals-babys-first-heap

Use after free

- free(p)，被 free 掉的 chunk 會被放進對應的 bin
- q = malloc(N)，malloc 如果 N 和 p chunk 的 size 大小相同，得到的 addr 會是上次 free 的 chunk
- 結果將造成 p == q，此時如果 p 繼續被使用，會與 q 相互影響，產生不可預期的錯誤
- 能造成的結果取決於 use 的方法，有可能造成任意讀取、任意寫入等問題

bins

- glibc 實作裡，bins 分為三種
- fastbin
  - size ≤ max_fast (default: 64B)
- smallbin
  - Size ≤ 512B
- largebin
  - size ≤ 128KB
- malloc 時可以快速找到大小適合的 chunks

fastbin

- free chunk 時，如果 chunk size <= 72，會被放進 fastbin
  - 64 + prev_size + size = 72
- 不取消 inuse bit，即不參與 freed chunk 的合併
- Singly linked list
- malloc 時拿 bin 裡的第一個 chunk
  - Last in, First out

smallbin

- size <= 512 時，使用 smallbin
- 為 normal bins 中前 64 (512/8) 個
- 每種 size 有對應的 bin，每個 bin 中存的 chunk 大小都相同
- malloc 時先找對應的 bin 裡有沒有可用的 chunk

largebin

- size > 512 時，會被放進 largebin
- 大約是指數遞增的 bin range
- 因為 bin 裡的 chunk 大小不一，用 sorted list 存，chunk 由小到大排列
- fd_nextsize, bk_nextsize 指向下一個大小不同的 chunk，用來加快 search

Unsorted bin

- free chunk 後，chunk 並不會立刻放進正確的 bin
  - fastbin 除外
- 如果 malloc 找不到恰好符合的 chunk ，此時會處理 unsorted bin，如果有大小剛好的就拿來使用
- 在沒有夠大的 free chunk 可以用時，會使用 top chunk
  - 記憶體中位置最大的 chunk
  - 可以任意調整大小
  - Top chunk 縮小時，多出的部份會成為新的 top chunk

Control eip

- ret
  - ret2text
  - ret2libc
- call eax
  - function pointer
  - hijacking GOT
- jmp [offset]

Global Offset Tables

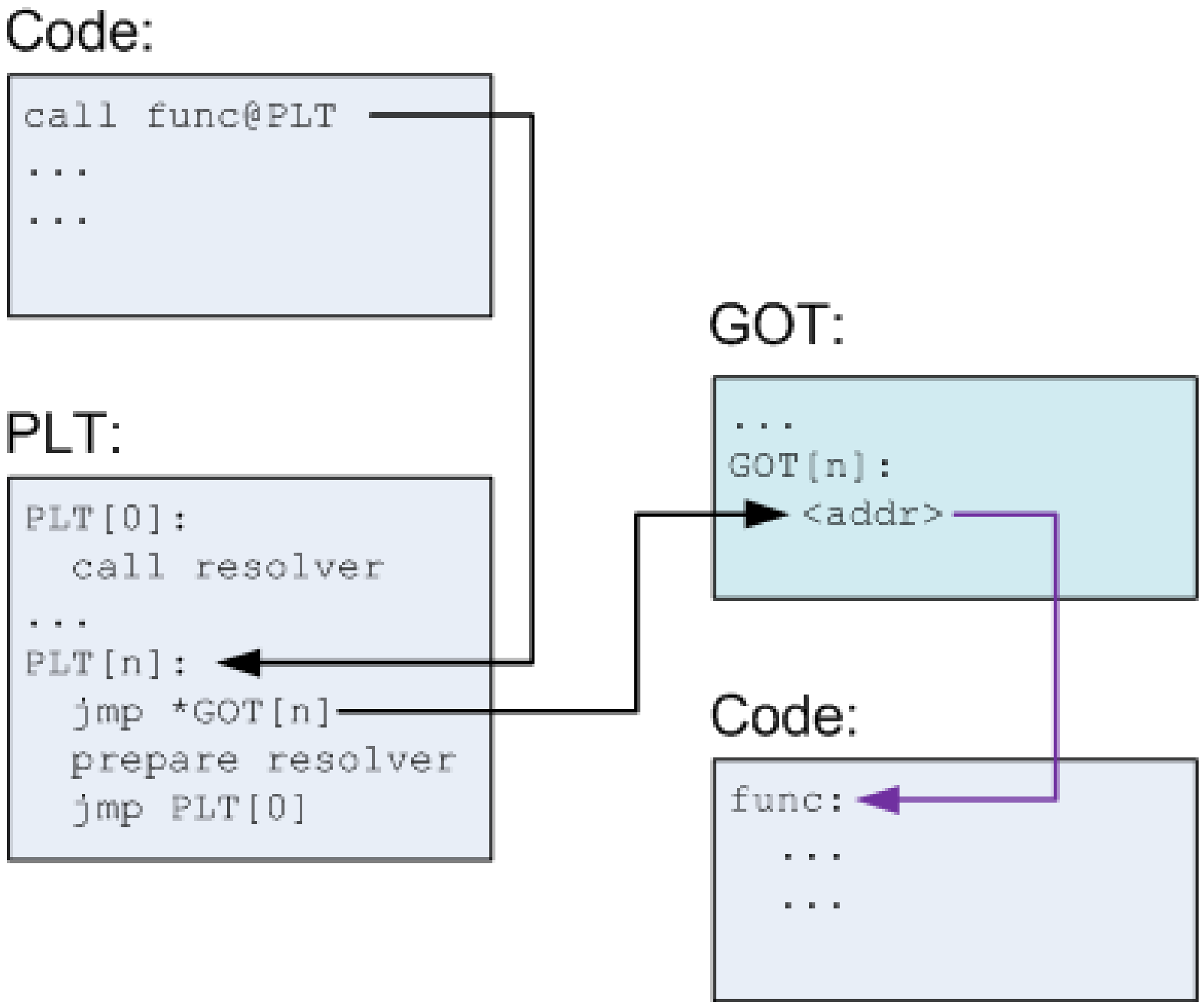
- 大部分的程序都會用到 glibc，如果一個 process 就使用一份 libc，很浪費記憶體空間
- 透過 GOT & PLT table 去動態 link 真正的 libc 真正的位置

```
Disassembly of section .got.plt:

0804a000 <.got.plt>:
 804a000:       14 9f                   adc    $0x9f,%al
 804a002:       04 08                   add    $0x8,%al
        ...

Disassembly of section .plt:

080483f0 <mprotect@plt-0x10>:
 80483f0:       ff 35 04 a0 04 08       pushl  0x804a004
 80483f6:       ff 25 08 a0 04 08       jmp    *0x804a008
 80483fc:       00 00                   add    %al,(%eax)
        ...
```


Code:
call func@PLT
...
...
PLT:
PLT[0]:
call resolver
...
PLT[n]:
jmp *GOT[n]
prepare resolver
jmp PLT[0]
GOT:
...
GOT[n]:
<addr>
Code:
func:
...
...

hijacking GOT

- bss segmentation is writable

```
00400000-0040b000 r-xp 00000000 fc:00 3407885  /bin/cat
0060a000-0060b000 r--p 0000a000 fc:00 3407885  /bin/cat
0060b000-0060c000 rw-p 0000b000 fc:00 3407885  /bin/cat
025c5000-025e6000 rw-p 00000000 00:00 0        [heap]
```

- If we can write memory arbitrarily, we can hijack GOT and control eip.
- Example: If exit() will be called at last, we can overwrite the GOT of exit() to our shellcode address.
- Then the program will execute our shellcode when calling exit().

Practice – use after free

- nc secprog.cs.nctu.edu.tw 10109

The Malloc Maleficarum

- Malloc Exploitation Techniques
  - The House of Prime
  - The House of Mind
  - The House of Force
  - The House of Lore
  - The House of Spirit
- Reference
  - http://packetstormsecurity.com/files/view/40638/MallocMaleficarum.txt

The House of Spirit

- fastbin 是用 linked list 紀錄可用的 chunk
- glibc 不會檢查 chunk 所在的 segment….
- 在可控的 address p 構造假的 chunk，free(p) 會將 p 放入 fastbin
- 再次 malloc 即可取得 p 的 pointer

Recent CTF about heap exploitation

- Doug Lea Heap overflow
  - Defcon 22 Quals Baby's First heap
  - http://ddaa.logdown.com/posts/200443-defcon-22-quals-babys-first-heap
- Use After Free
  - Defcon 22 Quals Shitsco
  - https://blog.skullsecurity.org/2014/defcon-quals-writeup-for-shitsco-use-after-free-vuln
- Heap Exploitaion
  - Hack.lu CTF 2014 OREO
  - https://github.com/ctfs/write-ups/blob/master/hack-lu-ctf-2014/oreo/exploit-by-cutz.pl