Linux Heap Introduction

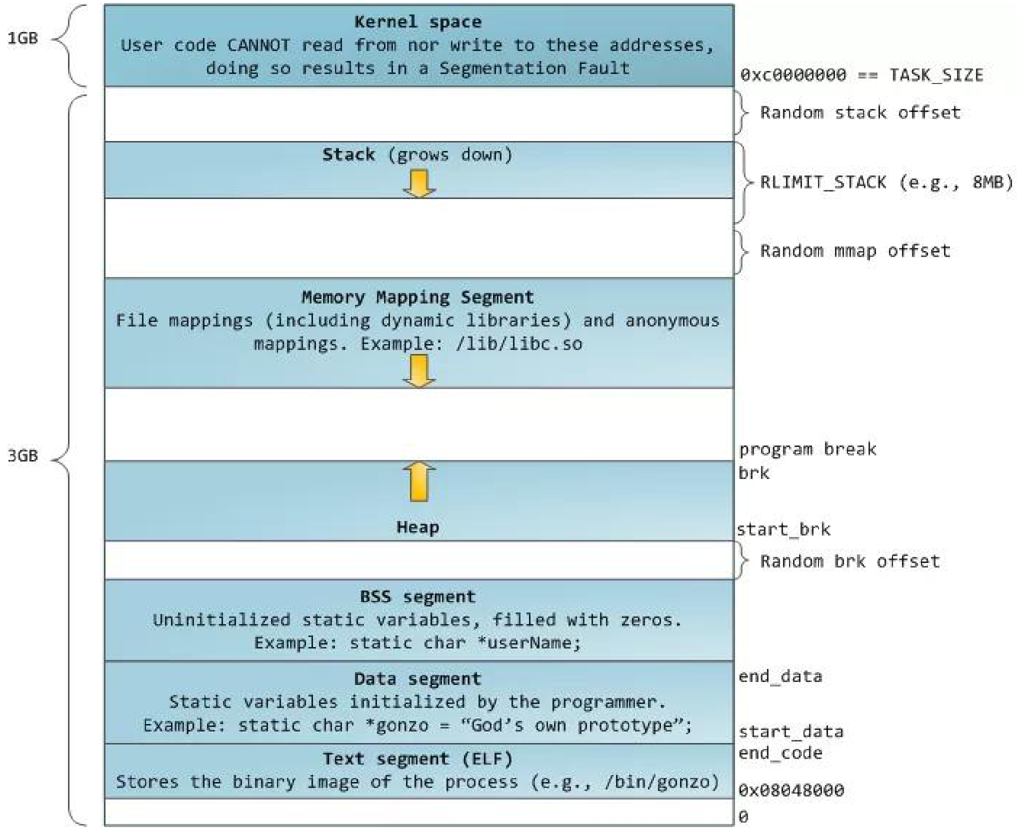
--w0lfzhang

blog:w0lfzhang.me

- Linux Heap Structure
- Some Techs in CTFs

Linux glibc: ptmalloc2

- Linux's *malloc: obtaining memory by invoking either brk or mmap syscall.
- mmap: when needing large memory.(> the value system setted)



Kernel space
User code CANNOT read from nor write to these addresses,
doing so results in a Segmentation Fault

Stack (grows down)

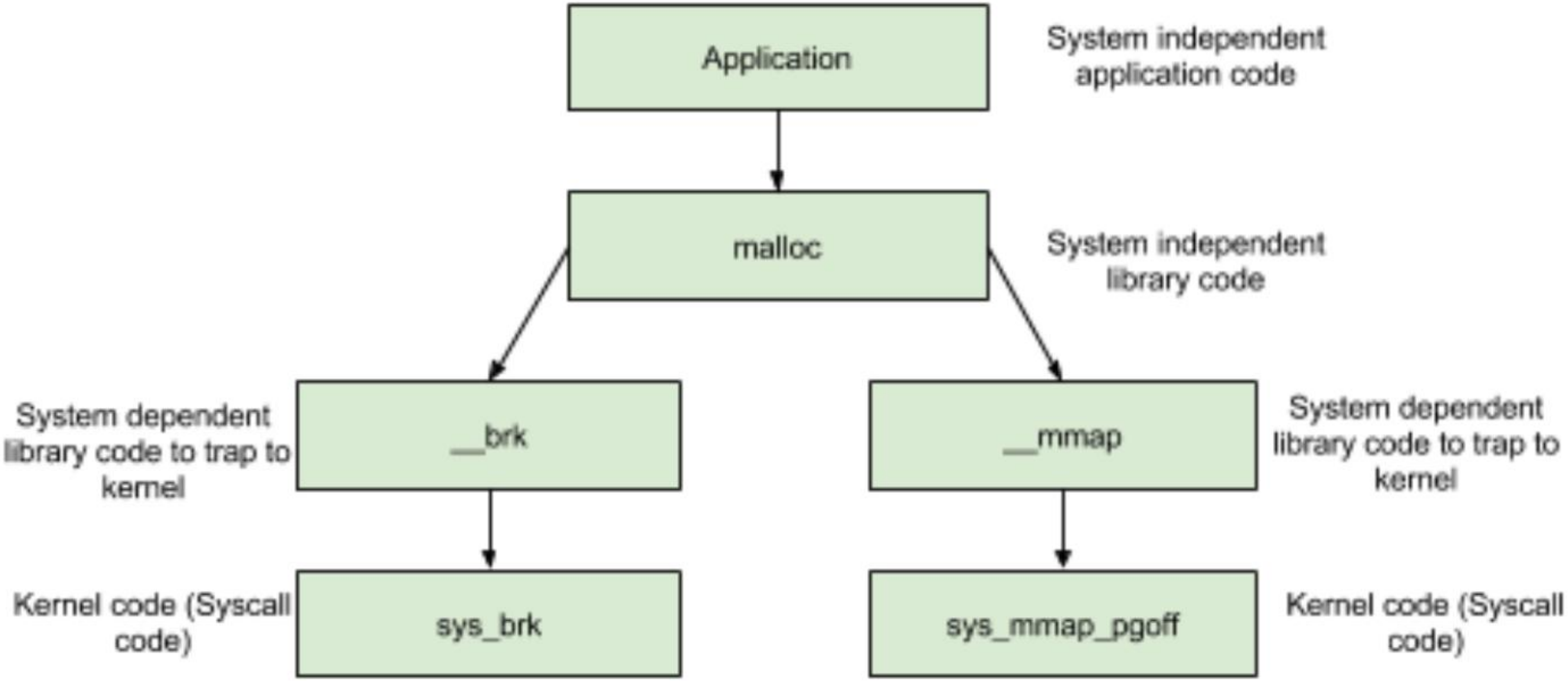
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous
mappings. Example: /lib/libc.so

Heap

BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;

Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = "God's own prototype";

Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)



Application

System independent application code

malloc

System independent library code

__brk

System dependent library code to trap to kernel

__mmap

System dependent library code to trap to kernel

sys_brk

Kernel code (Syscall code)

sys_mmap_pgoff

Kernel code (Syscall code)

Some important data structures

- heap_info: heap header. arena can have multiple heaps(not including main thread)
- malloc_state: arena header. The process have just one main_arena and non_main_arena(maybe > 1). The thread just have one arna.
- malloc_chunk: chunk header. Fastbin, smallbin ad largebin

- Main thread does't have the heap_info structrue
- main_arena is a global varible

heap_info

- typedef struct _heap_info
- {
- mstate ar_ptr; /* Arena for this heap. */
- struct _heap_info *prev; /* Previous heap. */
- size_t size;   /* Current size in bytes. */
- size_t mprotect_size; /* Size in bytes that has been mprotected
- PROT_READ|PROT_WRITE. */
- /* Make sure the following data is properly aligned, particularly
- that sizeof (heap_info) + 2 * SIZE_SZ is a multiple of
- MALLOC_ALIGNMENT. */
- char pad[-6 * SIZE_SZ & MALLOC_ALIGN_MASK];
- } heap_info;

malloc_state

- struct malloc_state
- {
- /* Serialize access. */
- mutex_t mutex;
- /* Flags (formerly in max_fast). */
- int flags;
- /* Fastbins */
- mfastbinptr fastbinsY[NFASTBINS];
- /* Base of the topmost chunk -- not otherwise kept in a bin */
- mchunkptr top;
- /* The remainder from the most recent split of a small request */
- mchunkptr last_remainder;

-

- /* Normal bins packed as described above */
- mchunkptr bins[NBINS * 2 - 2];

- /* Bitmap of bins */
- unsigned int binmap[BINMAPSIZE];

- /* Linked list */
- struct malloc_state *next;

- /* Linked list for free arenas. */
- struct malloc_state *next_free;

- /* Memory allocated from the system in this arena. */
- INTERNAL_SIZE_T system_mem;
- INTERNAL_SIZE_T max_system_mem;
- };

malloc_chunk

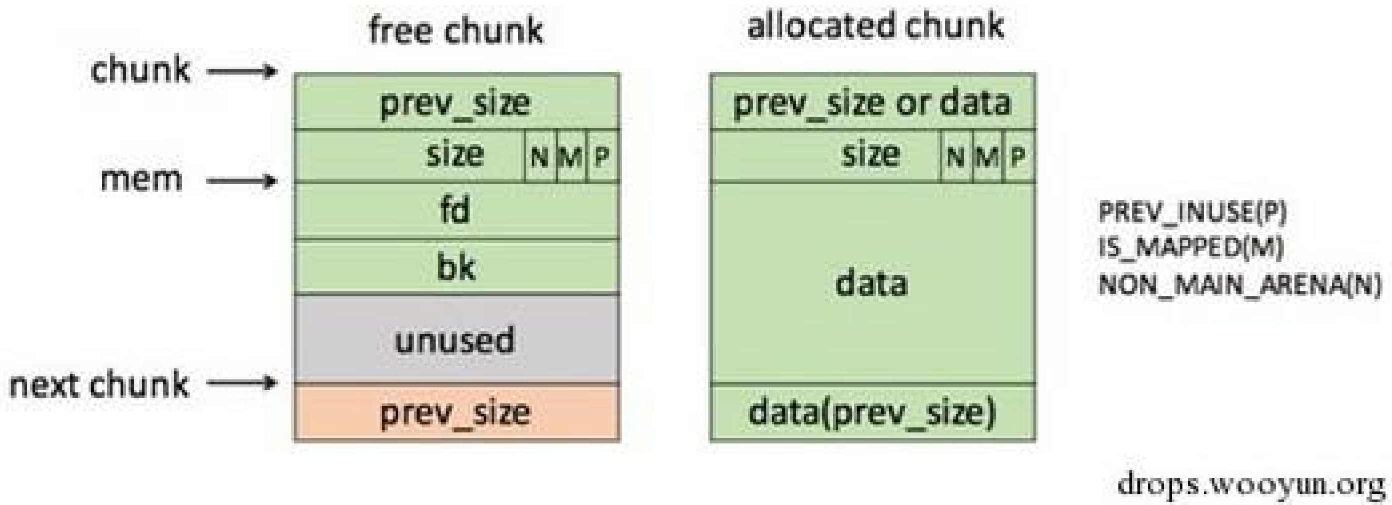
- struct malloc_chunk {

- INTERNAL_SIZE_T prev_size; /* Size of previous chunk (if free). */
- INTERNAL_SIZE_T size; /* Size in bytes, including overhead. */

- struct malloc_chunk* fd; /* double links -- used only if free. */
- struct malloc_chunk* bk;

- /* Only used for large blocks: pointer to next larger size. */
- struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
- struct malloc_chunk* bk_nextsize;
- };

chunks



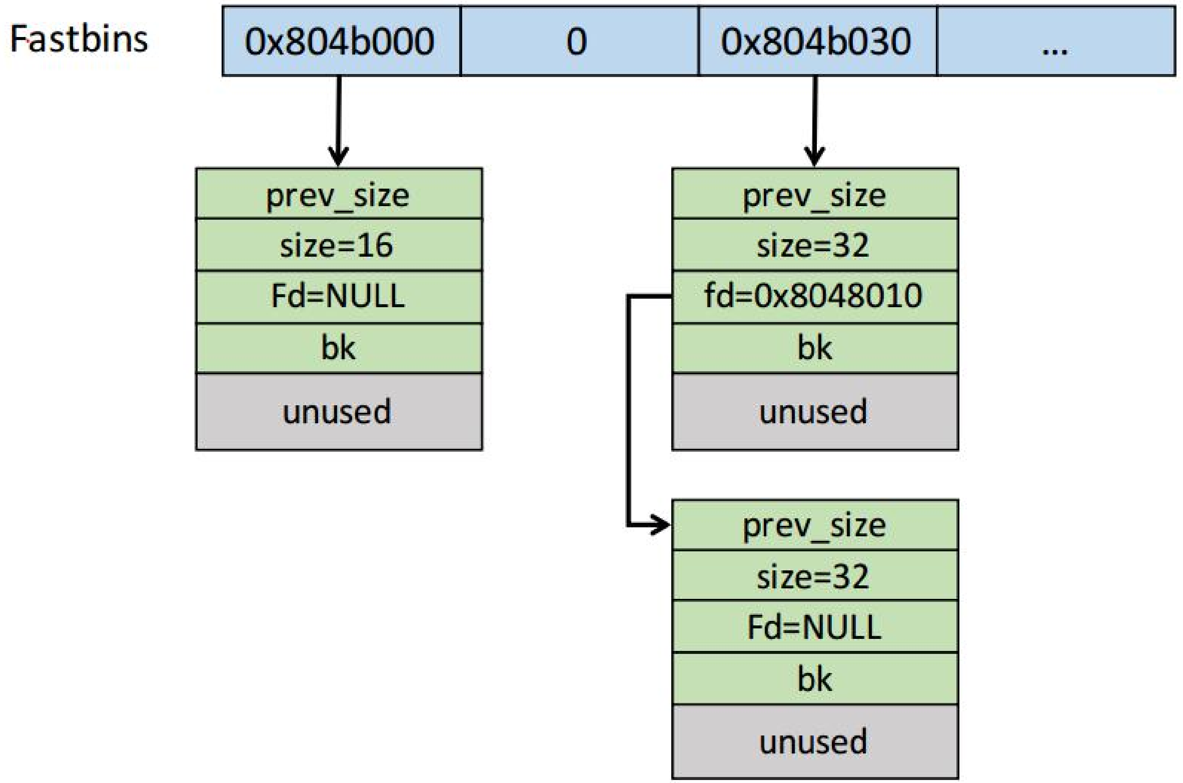
drops.wooyun.org

Bins

- fastbinY[]: just including fast bin
- bins[]: including unsorted bin, small bin and large bin
- bin 1: unsorted bin
- bin 2-bin 63: small bin
- bin 64-bin 126: large bin

Fast bin

- 16~64 bytes for x86_32
- 32~128 bytes for x86_64
- LIFO: Single Linked List
- No coalescing



Unsorted bin

- when the chunk was freed, the chunk was added in unsorted bin(not including fast bins)
- when mallocing a chunk, if the chunk's size(in unsorted bin) > requested chunk's size, malloc from the chunk(by cutting it) and the rest becomes the last reminder chunk
- if the chunk's size < requested chunk's size, malloc from top chunk and the chunk is linked into the corresponding bin

Small bin

- 16, 24, 32, …, 508 bytes for x86_32
- FIFO: Circular double linked list
- Coalescing when next to each other
- After free(), check if the next chunk is freed; if so, coalesce them and add the new chunk into unsorted bin

Large bin

- >=512 bytes(x86_32)
- the chunk's size can be different in the bin

- malloc: finding the proper chunk
- if =, mallocing directly
- if >, cutting it and malloc the requsted size chunk and the rest is added into unsorted bin

Top chunk & Last Reminder chunk

- Top chunk: the top chunk of an arena
- Used to service user request when there is NO free blocks
- Extended using sbrk (main arena) or mmap (thread arena) syscall

- Last Reminder chunk: remainder from the most recent split of a small request

Some Useful Skills in CTFs

- Malloc Maleficarum:
- house of force
- house of spirit
- house of lore
- house of mind
- house of prime

- blog: gbmaster.wordpress.com

unlink

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

- The check:
- if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
-       malloc_printerr (check_action, "corrupted double-linked list", P);

- Solution:
- Find a pointer ptr pointing to p(*ptr == p)
- p->fd = ptr - 0x18(64bit)
- p->bk = ptr - 0x10

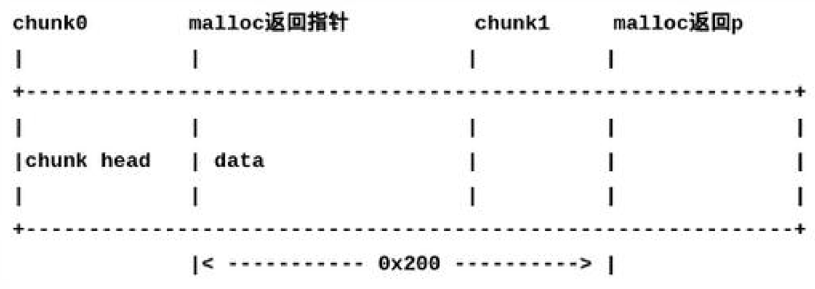
- Finally, p = ptr - 0x18

double free

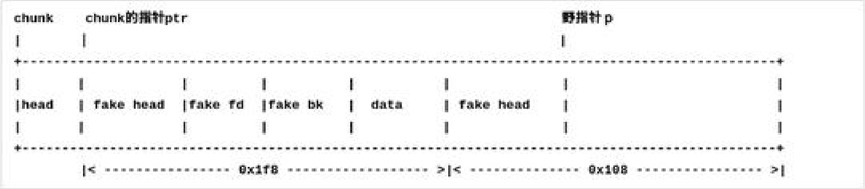
- free a pointer twice
- using unlink essentially

- p1 = malloc(n)
- p2 = malloc(m)
- free both of them
- a wild pointer p2

- then malloc(x)
- x > n + 16(32bit)
- so as to make fake data to use the wild pointer p



```
chunk0        malloc返回指针          chunk1     malloc返回p
|             |                      |          |
+--------------------------------------------------------------+
|             |                      |          |              |
|chunk head   | data                 |          |              |
|             |                      |          |              |
+--------------------------------------------------------------+
              |< ----------- 0x200 ----------> |
```



```
chunk   chunk的指针ptr                                          野指针p
|       |                                                       |
+----------------------------------------------------------------------------------------+
|       |          |         |         |          |             |                        |
|head   | fake head |fake fd  |fake bk  |   data    | fake head   |                        |
|       |          |         |         |          |             |                        |
+----------------------------------------------------------------------------------------+
        |< --------------- 0x1f8 ----------------- >|< -------------- 0x108 ---------------- >|
```

- Understanding the heap by breaking it - Black Hat
- http://www.blackhat.com/presentations/bh-usa-07/Ferguson/Whitepaper/bh-usa-07-ferguson-WP.pdf

use after free

- using memory which is freed
- leading to arbitrary code execution
- common in c++

- https://sploitfun.wordpress.com/2015/06/16/use-after-free/

off by one

- chunk overlapping
  - extending free chunks
  - extending allocated chunks
  - shrinking free chunks
- unlink

- Glibc Adventures: The Forgotten Chunks
- https://www.contextis.com/documents/120/Glibc_Adventures-The_Forgotten_Chunks.pdf

- unsorted bin attack
- fast bin attack
- ……