Thinking about Heap Exploitation

BrieflyX
Blue-lotus / Redbud
brieflyx.me
2018.4

OUTLINE

- “Chunkflow” of ptmalloc
- Common primitives
- Way to libc
- Crafting chunks to overlap
- Controlling the pointers
- Colorful fastbin attack
- More colorful non-fastbin corruption

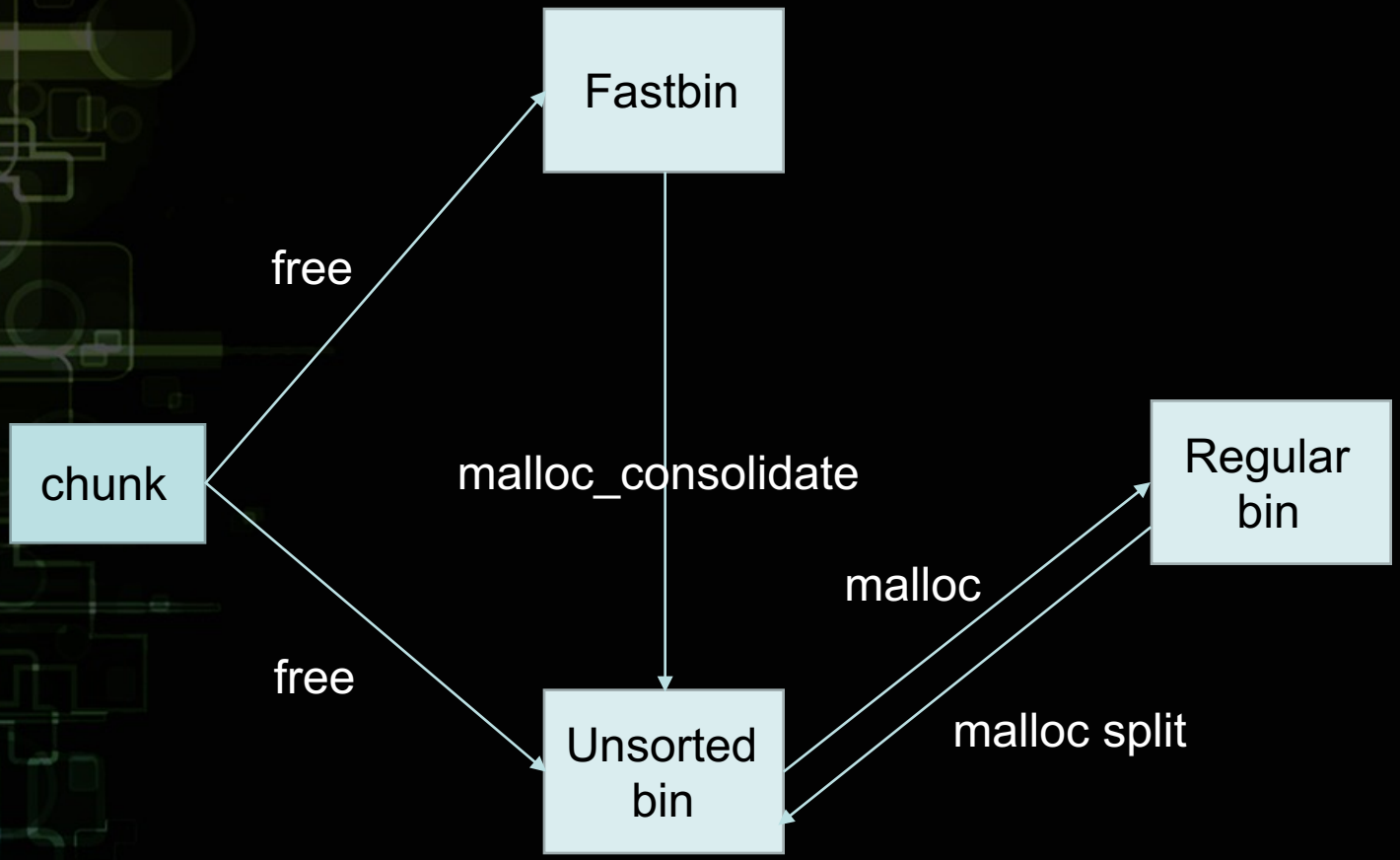
Purpose

- Turn corruptions on the heap to overwrite in libc !
- Unless there are datum on the heap to hijack control flow…

Chunkflow

- Learn about how malloc/free/realloc will affect chunks on the heap & chunks in bins

Chunkflow



Common primitives

- Arbitrary overflow in heap
- Use after free
- Off-by-one
- Out of bound
- ……

Way to libc

- Mostly we need to leak libc addr.
- Put chunks into non-fastbin!

Way to libc II

- All fastbins?
- Trigger malloc_consolidate
  - When requesting a large bin chunk (>0x400 in 64-bit) (malloc)
  - When a chunk is merged into top (free)

Way to libc III

- Top chunk size?
- Not only house of force
- Trigger sysmalloc
- Top chunk will be freed

```
assert ((old_top == initial_top (av) && old_size == 0) ||
        ((unsigned long) (old_size) >= MINSIZE &&
         prev_inuse (old_top) &&
         ((unsigned long) old_end & (pagesize - 1)) == 0));
```

Crafting overlapped chunks

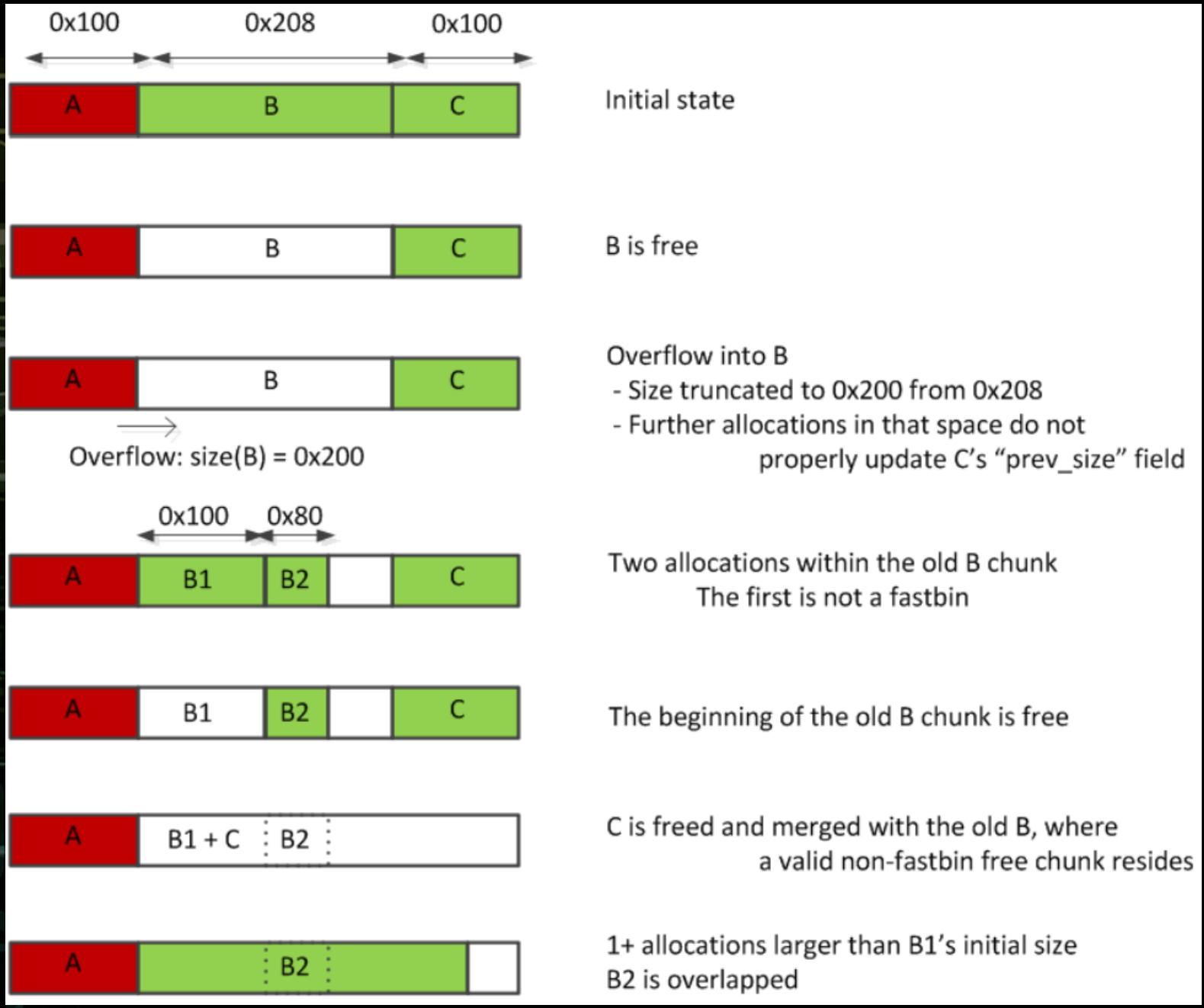
- Core skills in heap exploit
- Turn use-after-free or off-by-one into arbitrary overflow
- Familiar with security checks
- https://heap-exploitation.dhavalkapil.com/diving_into_glibc_heap/security_checks.html

Crafting overlapped chunks

- Use after free
- Arbitrary size : fastbin attack
- Limited size
  - free -> consolidate -> malloc back

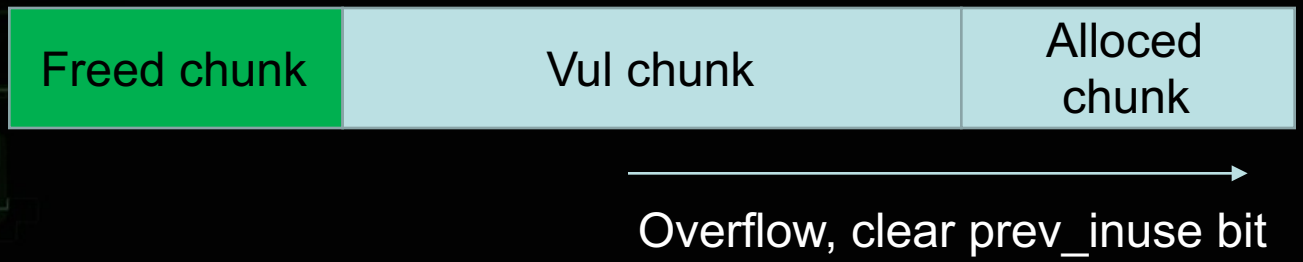
Crafting overlapped chunks

- Off-by-one
- Quite common in input functions
- Traditional method : shrink free chunk


0x100
0x208
0x100
A
B
C
Initial state
A
B
C
B is free
A
B
C
Overflow: size(B) = 0x200
Overflow into B
- Size truncated to 0x200 from 0x208
- Further allocations in that space do not properly update C's "prev_size" field
0x100
0x80
A
B1
B2
C
Two allocations within the old B chunk
The first is not a fastbin
A
B1
B2
C
The beginning of the old B chunk is free
A
B1 + C
B2
C is freed and merged with the old B, where a valid non-fastbin free chunk resides
A
B2
1+ allocations larger than B1's initial size
B2 is overlapped

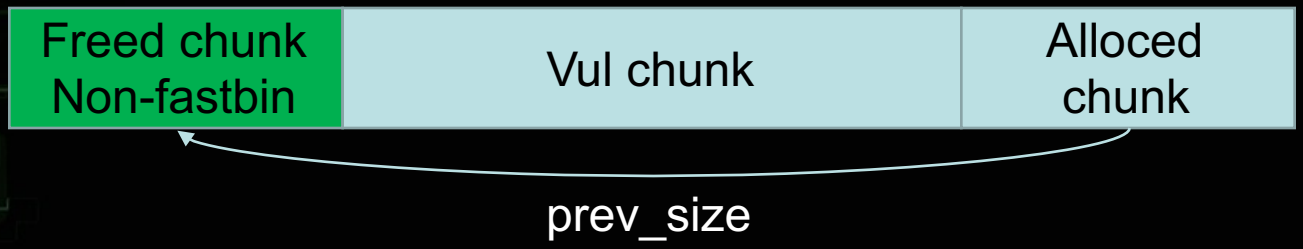
Crafting overlapped chunks

- Off-by-one
- Popular method : House of Einherjar
- Fake prev_size trick



Crafting overlapped chunks

- Off-by-one
- Popular method : House of Einherjar
- Fake prev_size trick

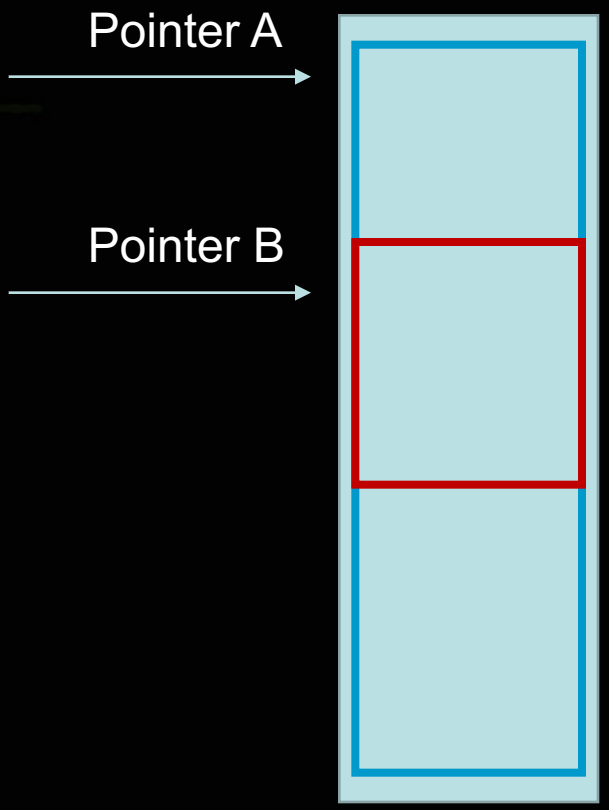


Crafting overlapped chunks

- New check in unlink of glibc 2.23
- chunksize(P) != prev_size (next_chunk(P))
- “corrupted size vs. prev_size”
- Not very difficult to bypass

Crafting overlapped chunks

- Final pattern of overlapping



Controlled data(A) goes beyond an address pointed by another pointer (B)

You almost own the heap

Controlling the pointers

- Put a chunk into fastbin
  - Chunk size
  - Next chunk size
- Put a chunk into unsorted bin
  - Chunk size & prev_inuse
  - Next chunk size & prev_inuse
  - Next next chunk's prev_inuse

Controlling the pointers

- When a chunk is in bins, and there are ‘fd’ or ‘bk’ on the heap. Here comes chances to jump out of the heap.

Colorful Fastbin

- Typical technique in past ctfs

- Double free : fastbin dup
- UAF : corrupted fd

Colorful Fastbin

- Selecting targets
  - An ‘int’ size (4-bytes)
- realloc_hook / malloc_hook
- GOT
- Utilizing misalignment

Colorful Fastbin

- Just overwriting malloc hook?
- Also, main_arena can be controlled

Colorful Fastbin

- Overwriting top
  - Crafting another size in main_arena
  - Huge global_max_fast

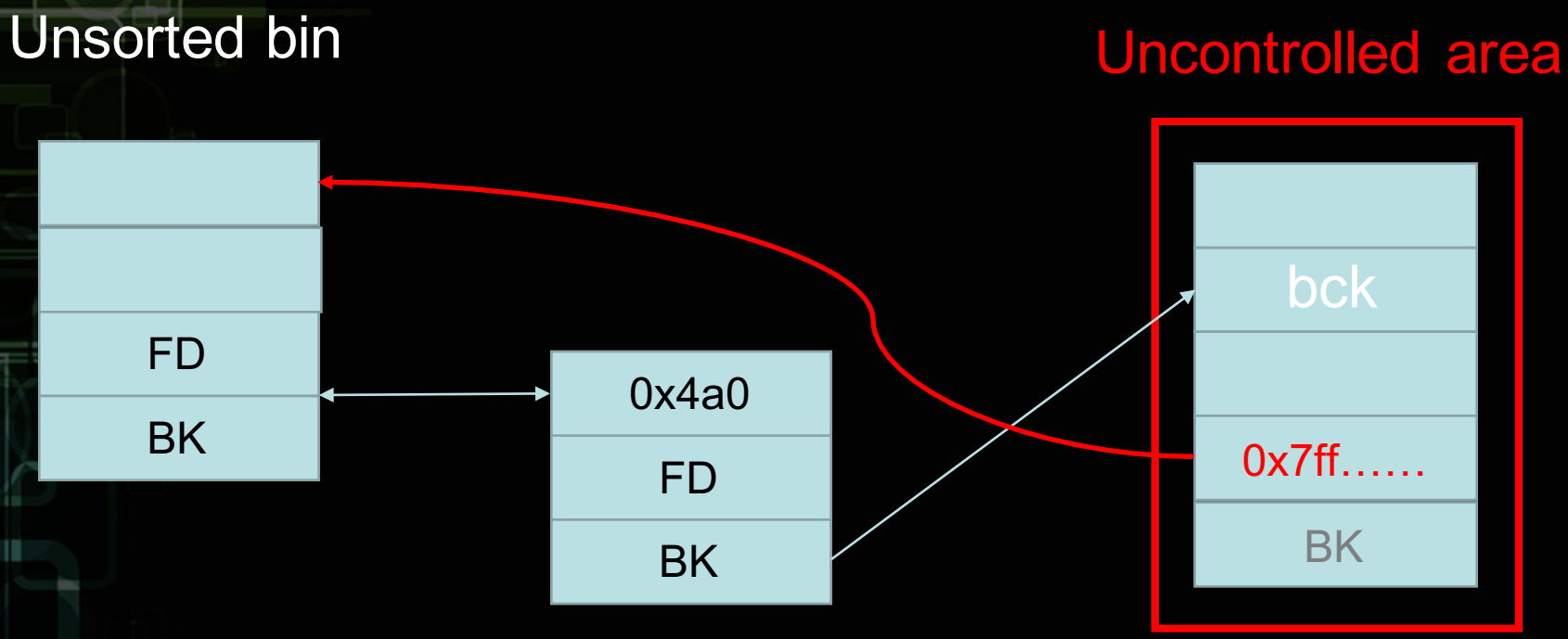
```
struct malloc_state
{
  /* Serialize access.  */
  mutex_t mutex;

  /* Flags (formerly in max_fast).  */
  int flags;

  /* Fastbins */
  mfastbinptr fastbinsY[NFASTBINS];

  /* Base of the topmost chunk -- not otherwise kept in a bin */
  mchunkptr top;
```

Powerful Unsorted Bin

- Simple unsorted bin attack
  - Corrupt ‘bk’ pointer of an unsorted bin chunk
  - Request the exact size,
  - Lead to an abitrary memory write (bk + 0x10 in 64bit) with unsorted bin addr.
- But, due to illegal ‘bk’, next malloc might crash

Powerful Unsorted Bin

```
/* remove from unsorted list */
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```



Powerful Unsorted Bin

- House of orange
  - FSOP
  - Corrupt '_IO_list_all' with unsorted bin address
  - Force fp->chain to point to heap (0x60 small bin)
  - Craft fake vtable to bypass the boudary check
- Once knowing libc, one unsorted bin attack leads to shell

Powerful Unsorted Bin

- Only once?
- No!
- We could repair the unsorted bin after its corruption

Powerful Unsorted Bin

- Plan α
  - After corrupting global_max_fast, use index overflow to overwrite unsorted bin
  - Shortcomings : difficult to repeat the attack

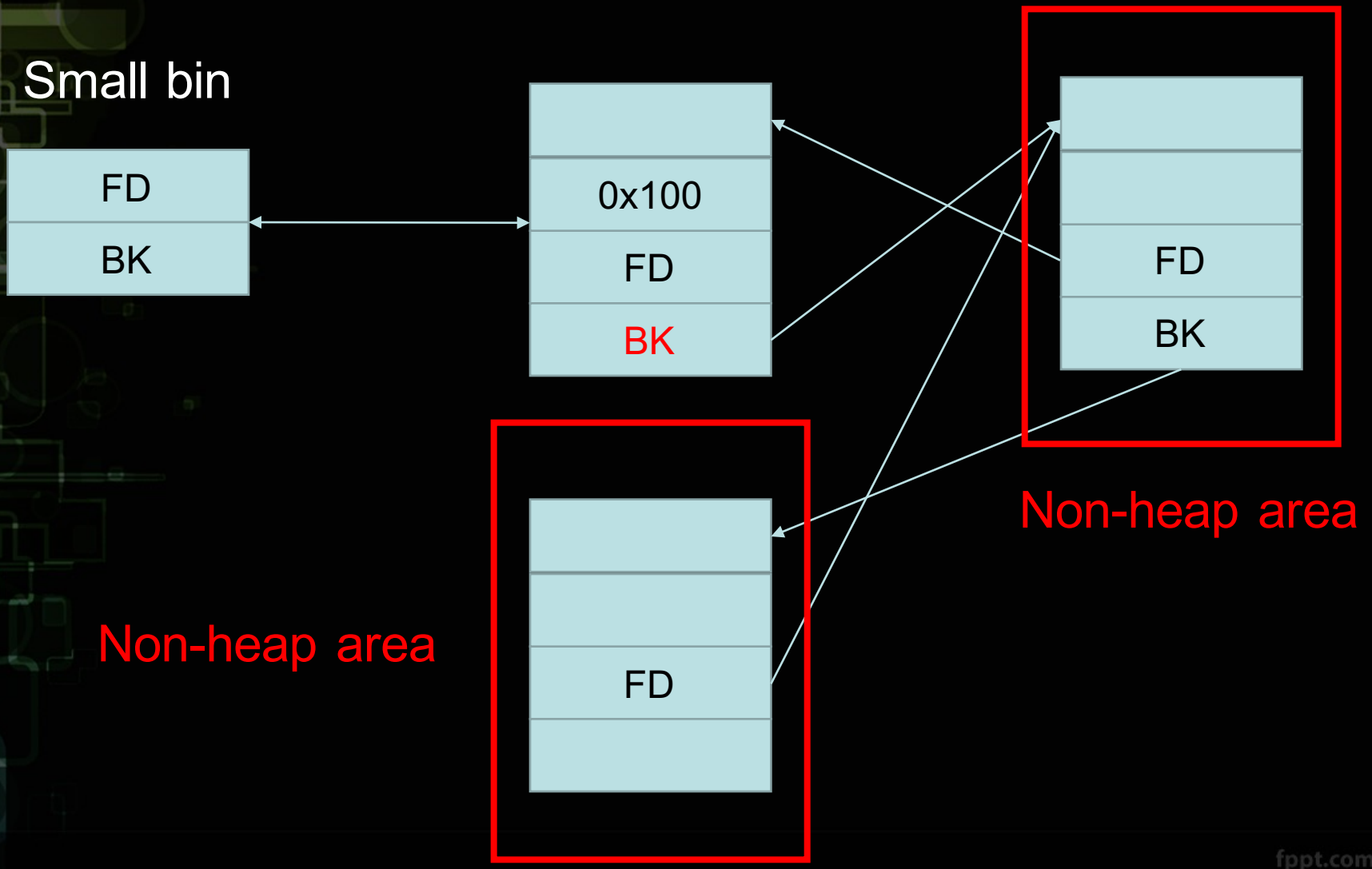
Powerful Unsorted Bin

- Plan β
  - Use large bin code
  - Almost unlimited arbitrary memory write with heap address

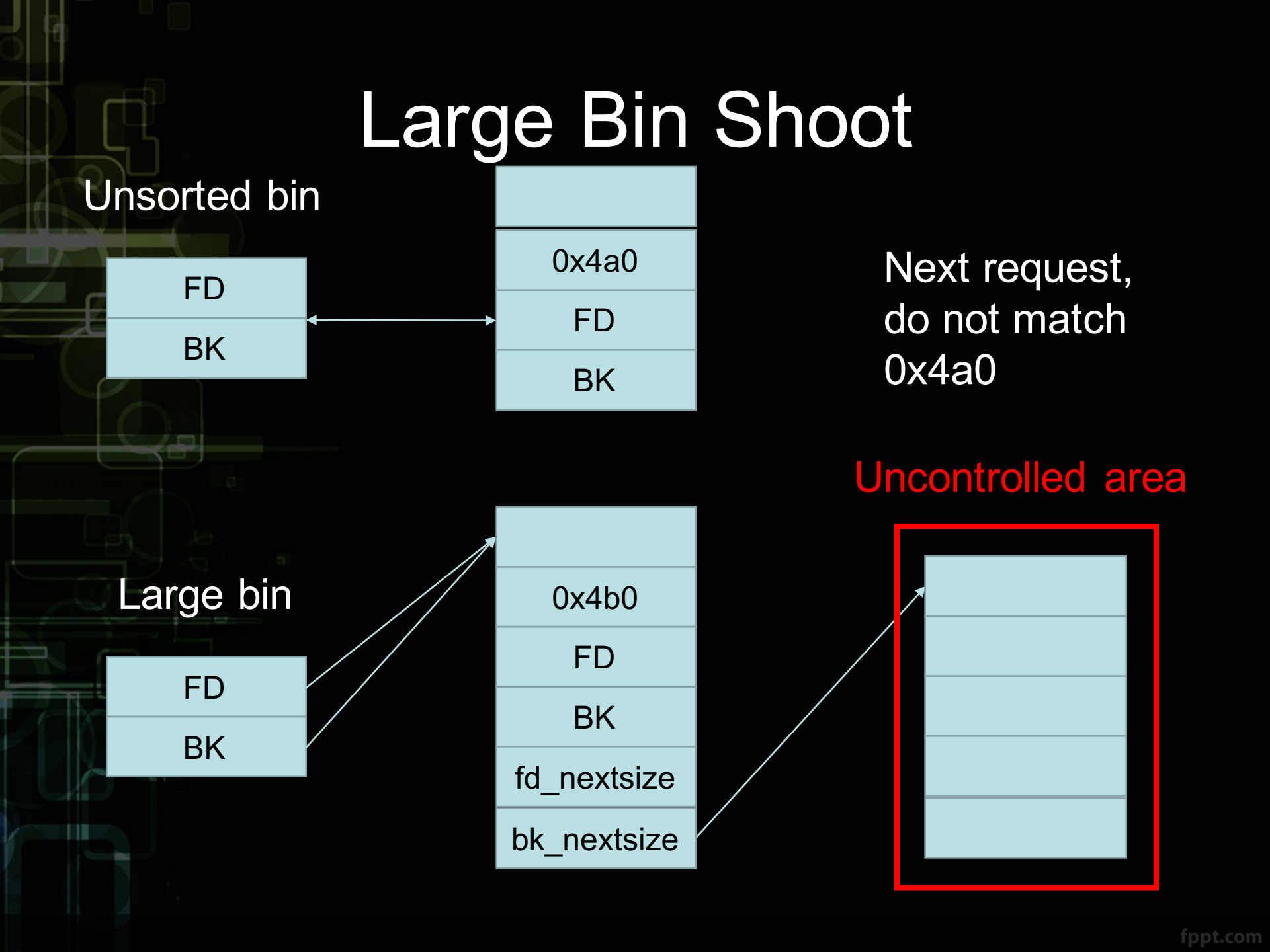
Small Bin Attack

- House of lore
- Need to create 2 address-known fake chunk, also need heap address

Small Bin Attack



Large Bin Shoot

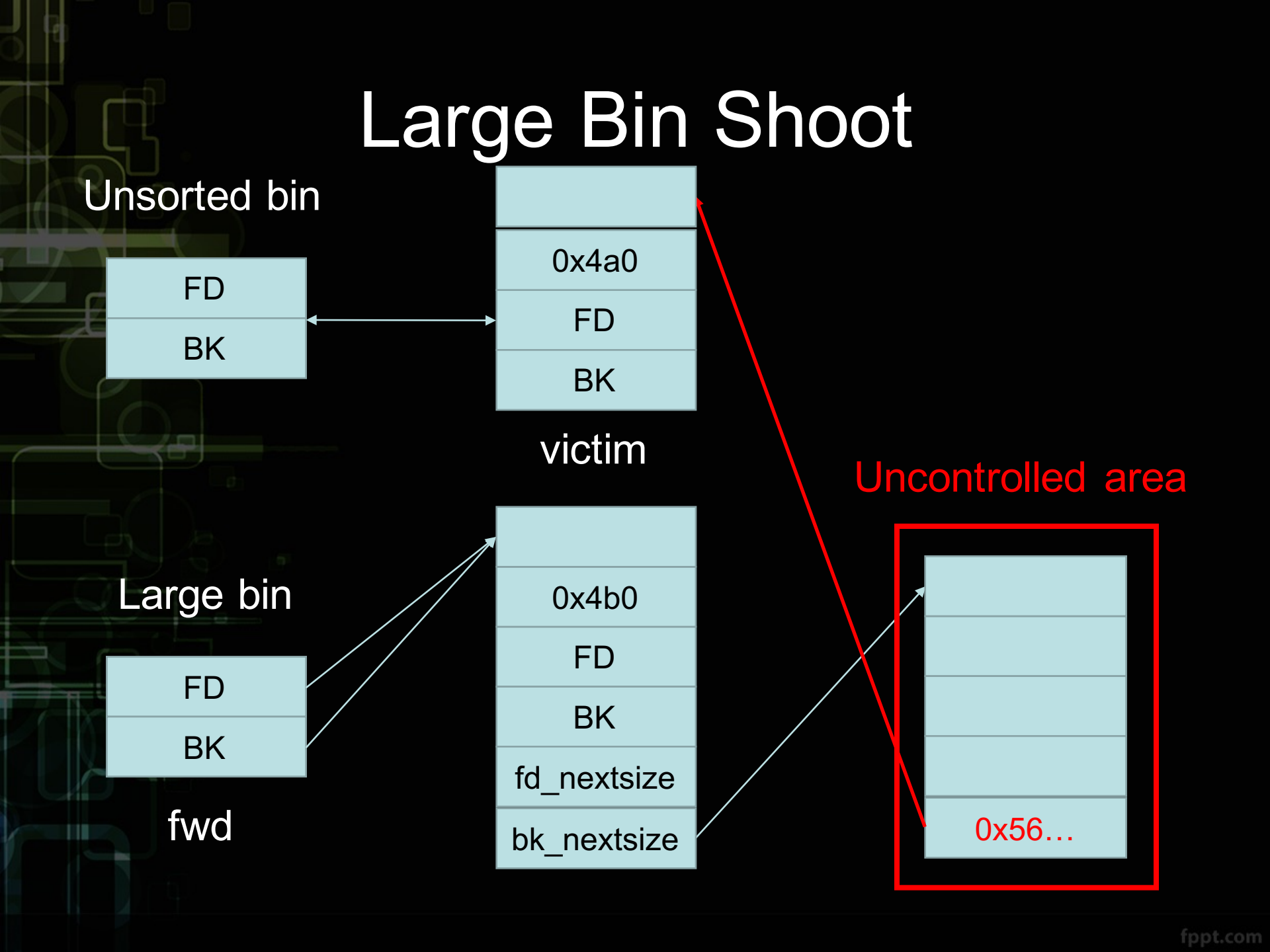


Large Bin Shoot

```
victim_index = largebin_index (size);
bck = bin_at (av, victim_index);
fwd = bck->fd;

/* maintain large bins in sorted order */
if (fwd != bck)
  {
    /* Or with inuse bit to speed comparisons */
    size |= PREV_INUSE;
    /* if smaller than smallest, bypass loop below */
    assert ((bck->bk->size & NON_MAIN_ARENA) == 0);
    if ((unsigned long) (size) < (unsigned long) (bck->bk->size))
      {
        fwd = bck;
        bck = bck->bk;

        victim->fd_nextsize = fwd->fd;
        victim->bk_nextsize = fwd->fd->bk_nextsize;
        fwd->fd->bk_nextsize = victim->bk_nextsize->fd_nextsize = victim;
```

Put into large bin & sorted by size

Large Bin Shoot



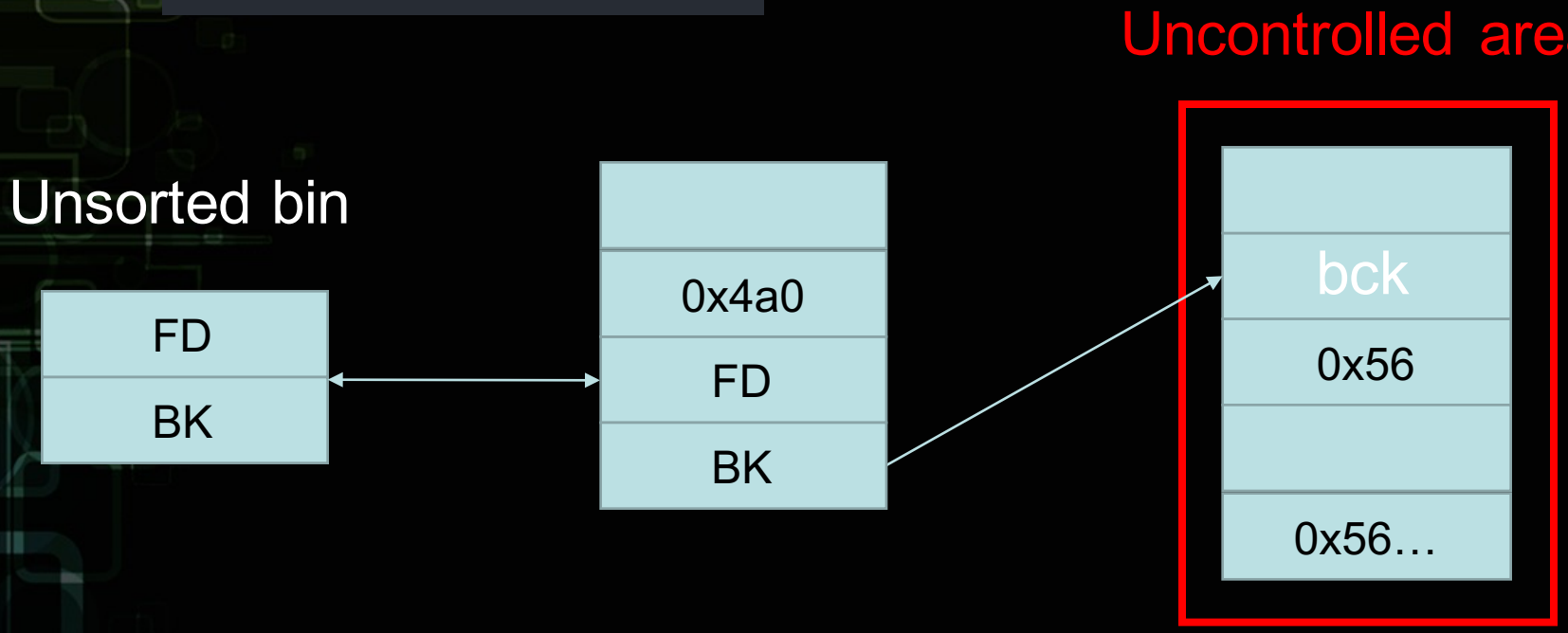
Large Bin Shoot

- Corrupt large chunk ‘bk_nextsize’ with address A
- Call ‘malloc’ to put a large chunk (victim) into large bin. And its size is smaller than chunk in large bin
- Then address A+0x20 will be overwritten by ‘victim’ pointer

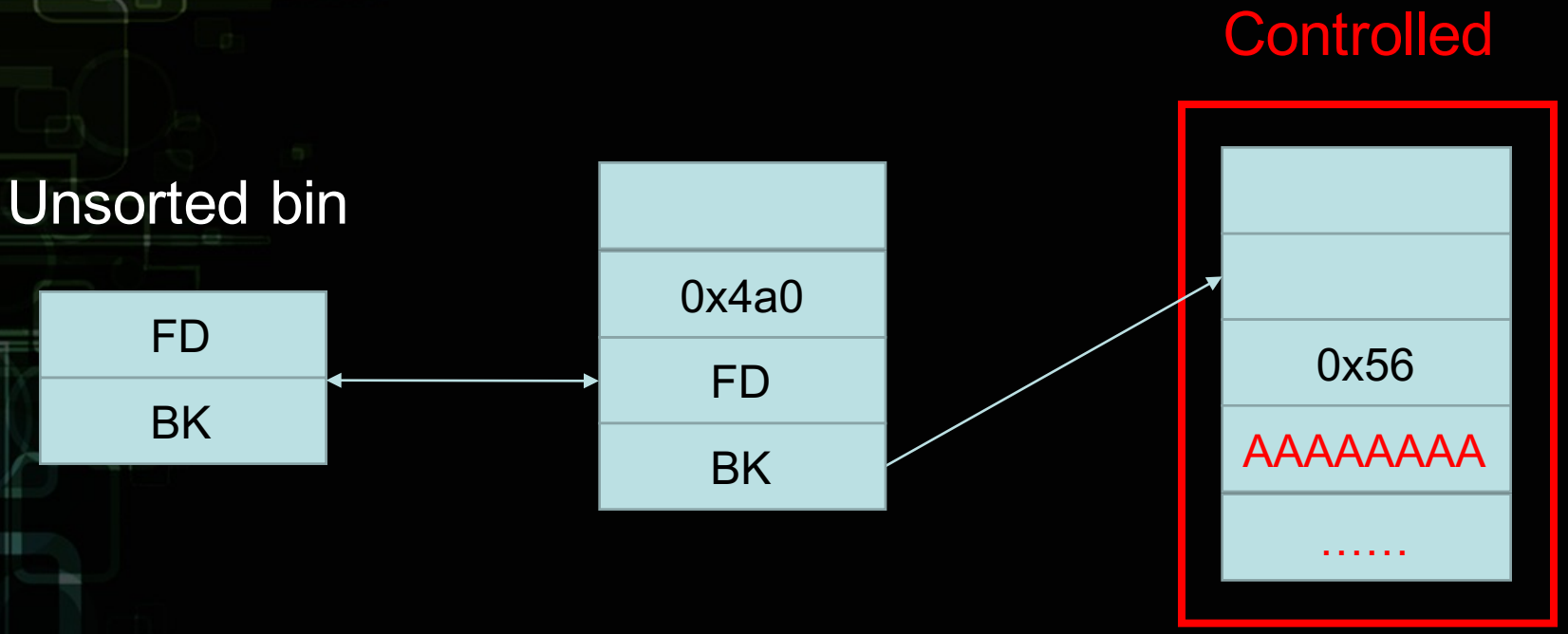
Unsorted Bin Attack

Next request for 0x50 size

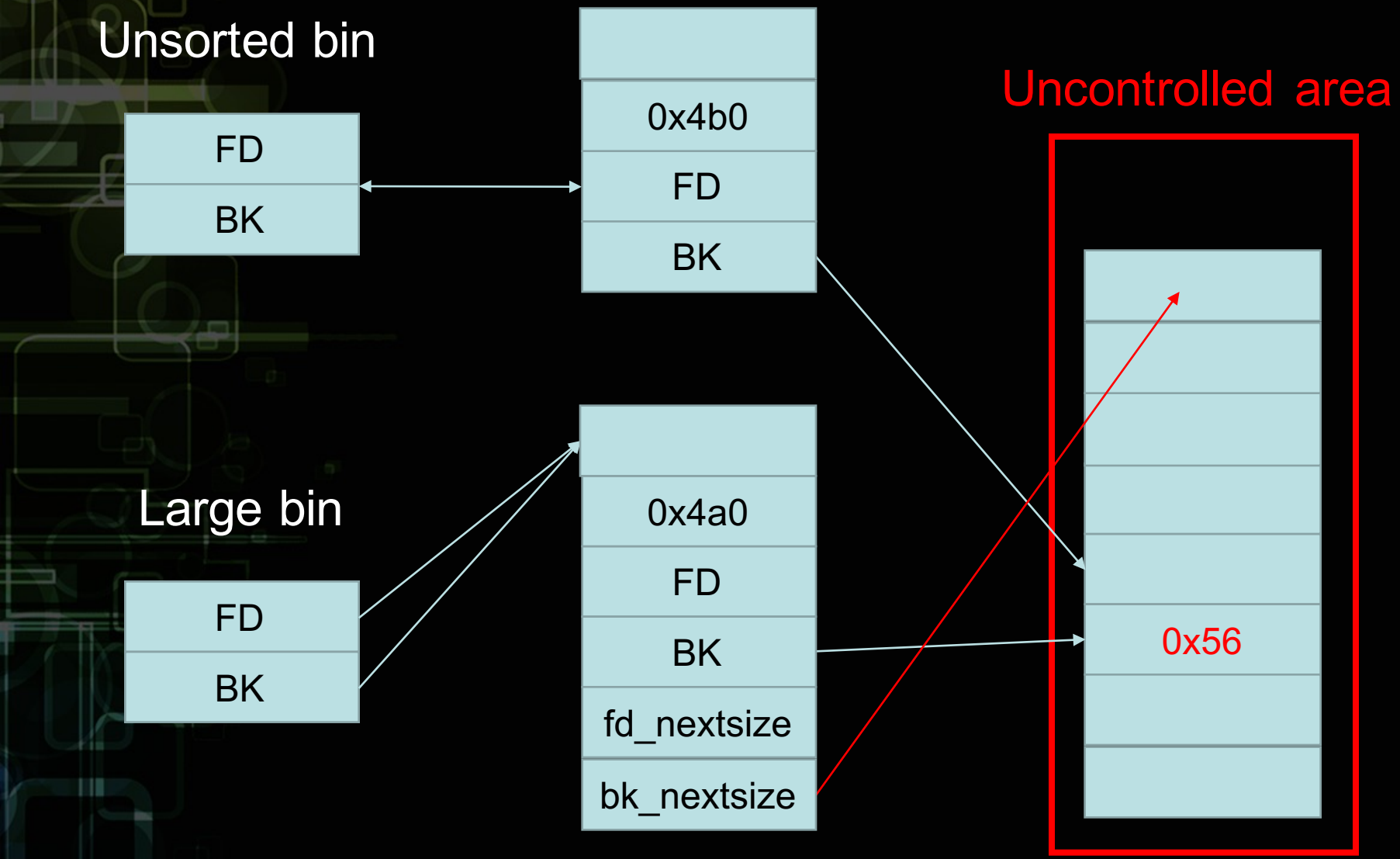
```
/* remove from unsorted list */
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```



Unsorted Bin Attack



More Attack Gesture



Summary

- Purpose : jump out of heap!
- Core skills : crafting overlapped chunks
- Dancing pointers : various bins tricks
- Misalignment tricks, hunting size in haystack

THANKS

HAVE FUN WITH HEAP!