



SP2024 Week 04 • 2024-02-11

# PWN III - Heap Exploitation

Sam Ruggiero

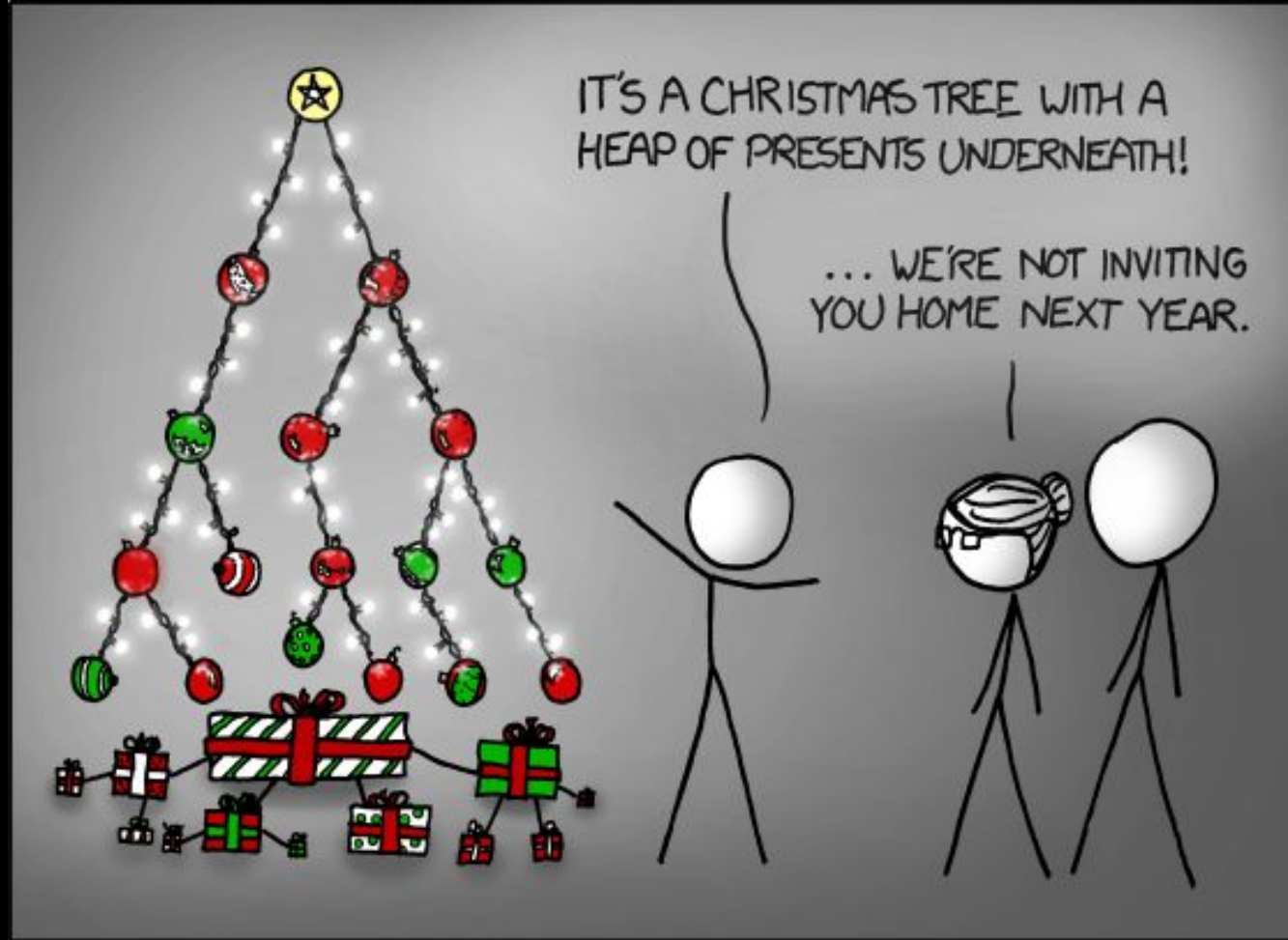
# Announcements

- Next Weekend, Feb 16 @ 10pm: LACTF Starts
  - We will be in person again! Free pizza/food for participants!



ctf.sigpwny.com

sigpwny{house\_of\_house\_of\_house}



# Memory Layout

Bottom of memory  
(0x0000000000000000)



Memory Region

.text  
(instructions)

.data  
(initialized  
globals)

.bss  
(uninitialized  
globals)

heap  
↓  
↑  
stack  
(runtime data)

Top of memory  
(0xFFFFFFFFFFFFFFFF)



# Memory Allocation

- We've seen static stack allocation
- C does support Variable Length Arrays
- How to we return an address to the buffers?

```
int* buffers(int size){  
    char buf[256];  
    char buf2[size];  
    return buf2;  
}
```



# The Heap

- The Heap  $\neq$  Heap ADT
- Allows Dynamic Memory Allocation
- Allocations are preserved across function calls
  
- `malloc(int size)`: Returns a pointer to allocated data
- `free(int* ptr)` : Frees the allocation located at ptr.
  
- Regardless what you put, the minimum allocation will make a 32 byte chunk with 24 bytes of usable memory (64-bit).

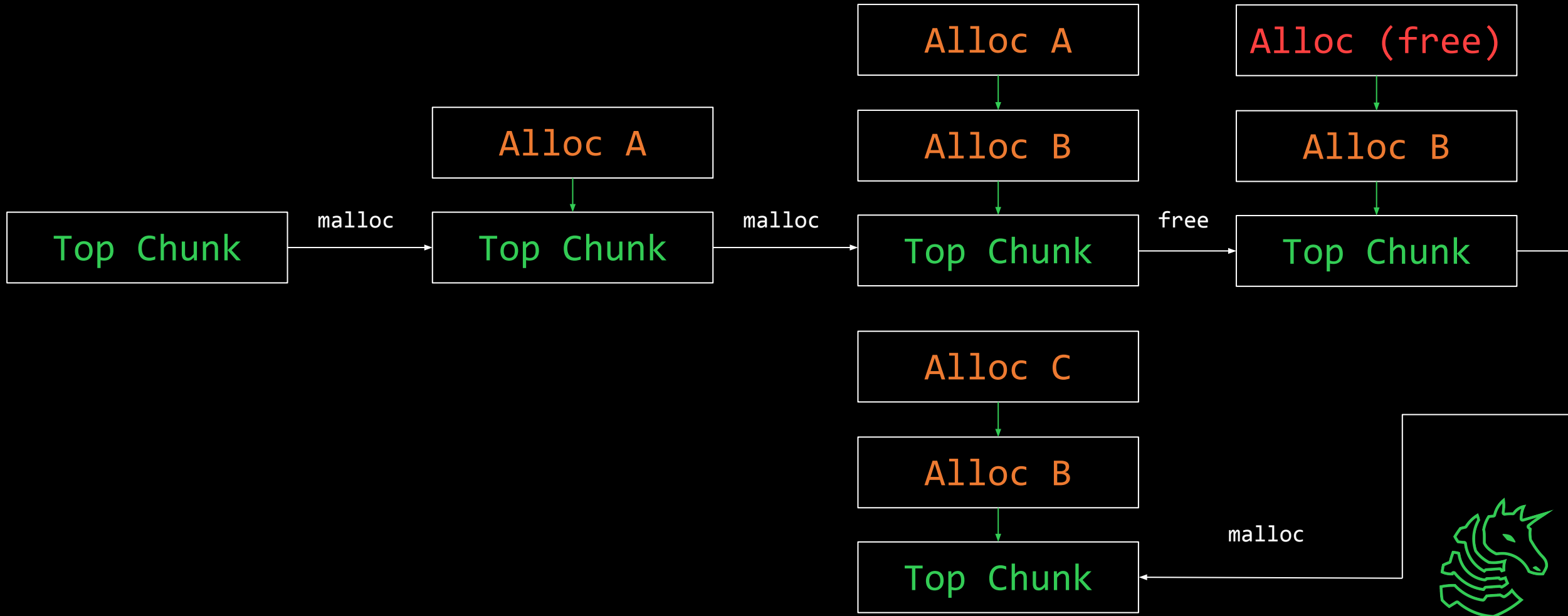


# Using the Heap

```
void example(int size){
    int* buf = malloc(sizeof(int)*size);
    for (int i=0; i<size; ++i){
        buf[i] = i;
    }
    free(buf);
}
```



# Basic Model of the Heap



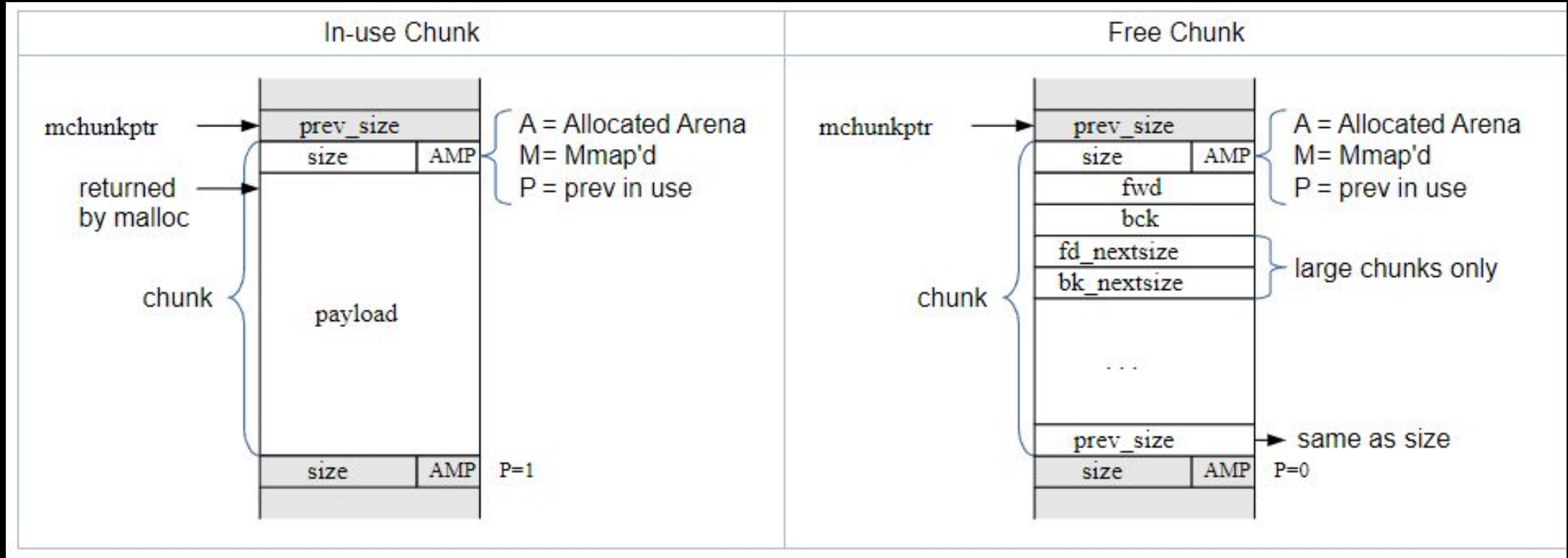


# What happens when you free

- When you free a chunk, it gets added to a free list
- If you malloc with a similar size, malloc checks the free list first
- glibc provides many optimizations to make allocations and reallocations fast!
- When you hear bins, think a double/single linked list.\
- Extra metadata gets placed into a chunk to maintain the list



# Free Chunk



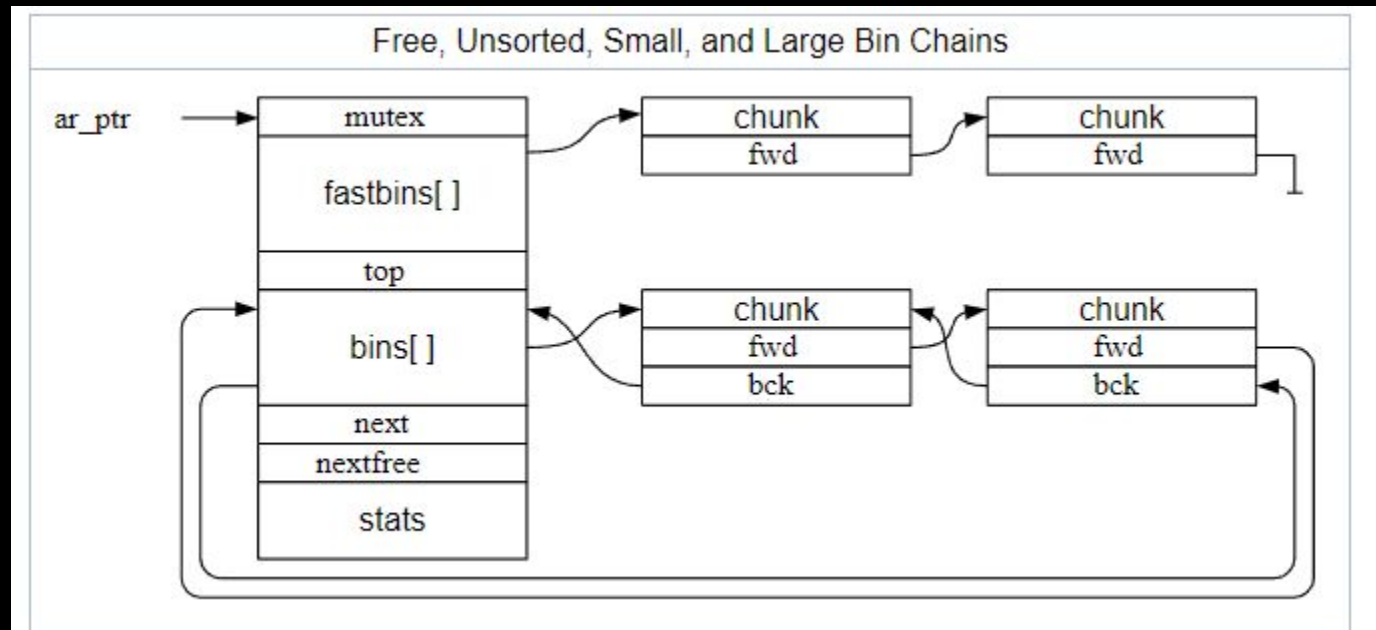
# Coalesce and Split

- Coalescing free chunks into one big chunk
  - Done when several adjacent chunks are free and not reused
  - Some bins prevent immediate coalescing
- Splitting a large chunk into a remainder chunk
  - Sometimes an alloc is smaller than a free chunk available
  - malloc will split the chunk into one of the correct size and a remainder chunk
- A large chunk may coalesce with the top chunk if adjacent
- When the top chunk gets too large, malloc will release the memory to the system.



# Bins

- Malloc will reserve memory from the system when it needs
- It will try to keep that memory available for reuse for as long as possible
- Free chunks have to be efficiently available for all chunk sizes



# Tcache Bins

- Introduced glibc2.26: Thread-specific bins for small allocations
- 64 bins, max count of 7, min size of 0x20, max size of 0x410
  - Subtract 8 for metadata, so min size for bin is 24 bytes or less.
- Each new thread gets a new tcache
  - Don't worry about this for now, multithread heap pwn is super advanced
- Singly-Linked, Only forward ptrs
- Because of their speed, there is lacking security checks within them



# Fast Bins

- The "overflow" for tcache
- 10 bins, between 0x20 and 0xb0
  - Sometimes only 7 are active...
- The "in-use" bit is maintained to prevent coalescing
  - The fast bin may be flushed and coalesced if large chunks are available
- Singly Linked



# Unsorted Bin

- The laundry pile of memory chunks
- When you free, and it doesn't fall in tcache/fast, it goes here
- malloc waits to see if you immediately reuse this chunk
- If not, it will sort it into a **small** or **large** bin



# Small Bins

- Like fast bins, fixed size in every bin
- 61 bins, starting at size 32 and up to 1024
- Adjacent chunks can be combined and moved to different bins
- Circular Doubly Linked



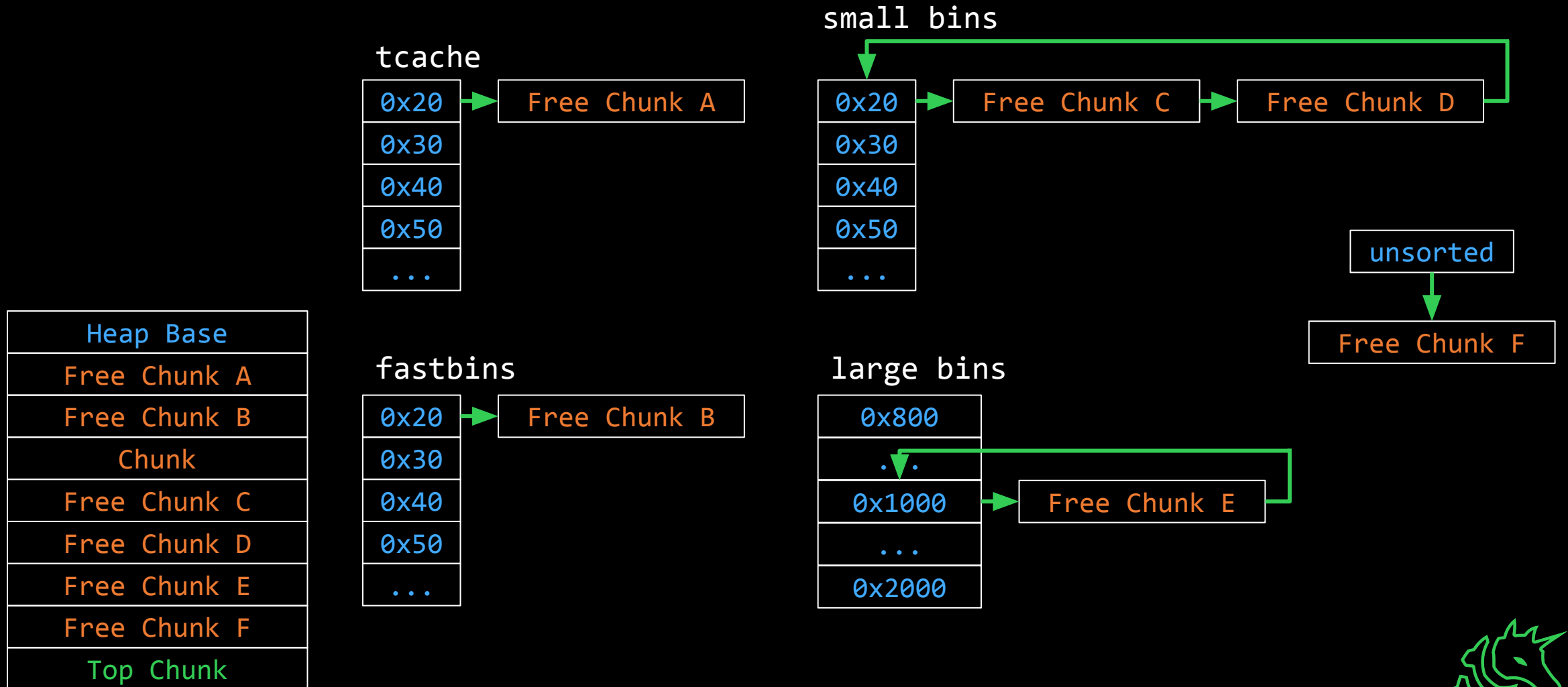


# Large Bins

- Stores a *range* of memory allocations in each bin
- 63 bins
- Memory allocations are inserted in a bin *in sorted order*
- An exponentially smaller number of bins are used for exponentially larger allocations
  - there are 32 bins that store allocations within 64 byte differences from each other
  - there are 2 bins that store allocations within 256kb differences
  - there is 1 "everything else" bin.
- Circular Doubly Linked



# Still a Basic Model of the Heap



# Malloc's Brothers

- `calloc(items, size)` allocates `items*size` bytes, and clears it to zero.
  - Does NOT use the tcache!
- `realloc(ptr, size)` changes the allocation size of `ptr` to `size`.
  - A horrifying amalgamation of `free` and `malloc`.
  - Will cause coalescing and splits if you increase/decrease size



# Malloc is Horrible to Understand

- There's still many optimizations, behaviors, and interactions not covered
- Looking at the glibc source code is informative but not easy
- It also changes version to version...
- The best way to understand malloc's memory patterns is to just experiment
  - Especially when doing challenges, as glibc versions change.

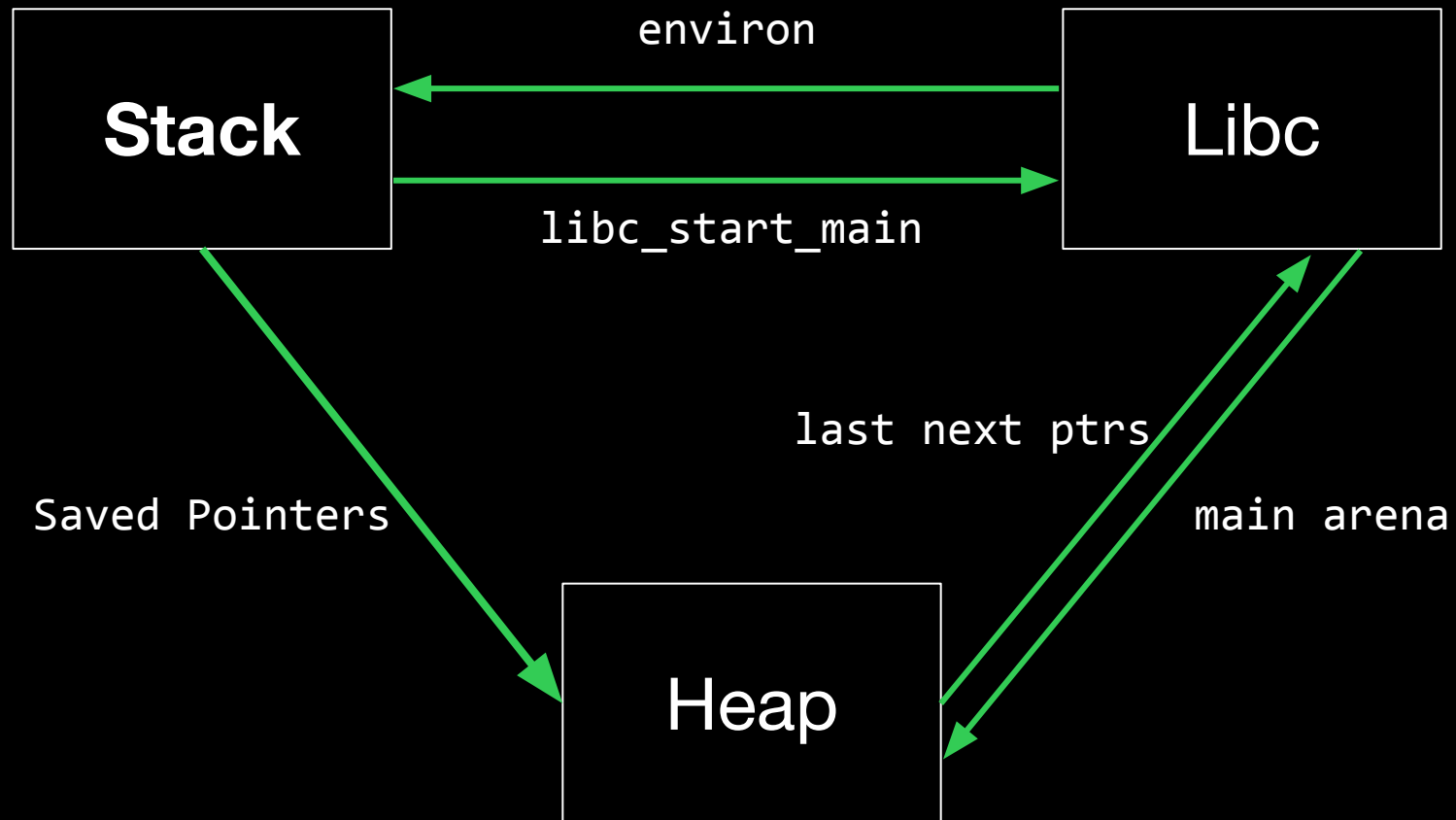


# Exploiting Programs

- We want arbitrary code execution:
  - Control the return address on the stack
  - Return to glibc library for useful functions
- Two Primitives:
  - Arbitrary Read
  - Arbitrary Write
- If we can get an arbitrary write to the stack, we can control program flow
- If we can get an arbitrary read, we can leak libc addresses



# Leak Traversal



# Use After Free (UAF)

- When you free a pointer, you are responsible for clearing the variable storing the pointer (`ptr = NULL;`)
- Nothing stops you from reading/writing to that pointer post-free
- This allows you to affect many aspects of malloc's state
  - Preload chunks with data
  - Modify Chunk sizes
  - Modify fwd/bk pointers(!!)

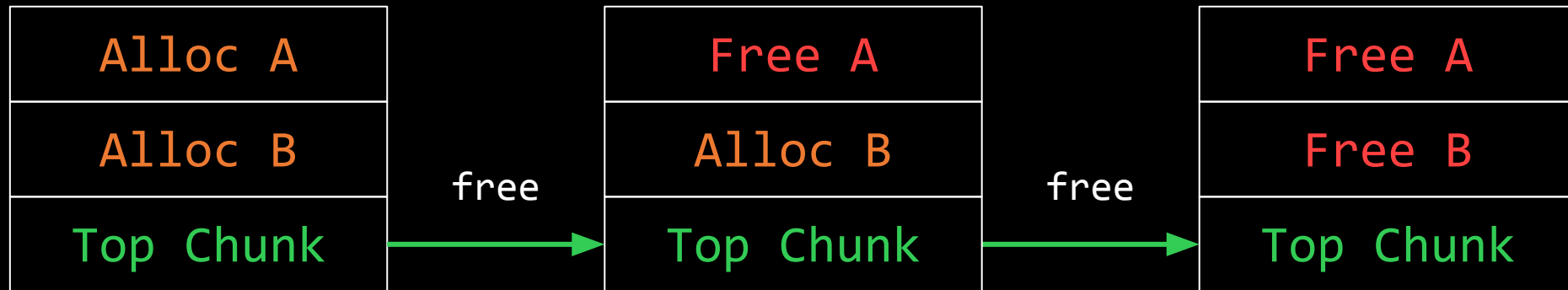


# Example: tcache poisoning

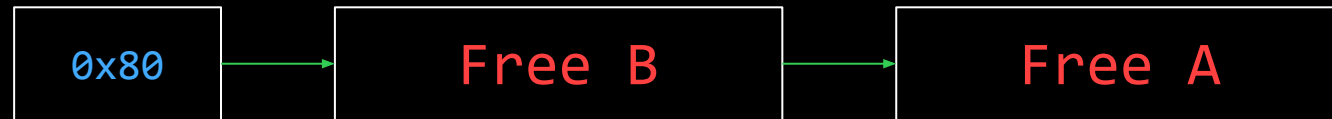




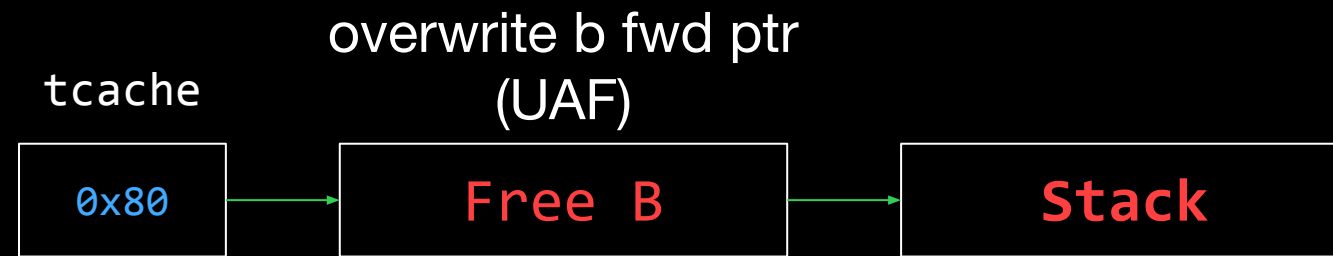
# Example: tcache poisoning



tcache



# Example: tcache poisoning



# Example: tcache poisoning

Stack Chunk



tcache



# Safe Linking

- malloc attempts to ""encrypt"" single-linked list pointers
- $(pos \gg 12) \wedge ptr$
- By taking the position of where the ptr is stored, you encrypt it with the ASLR bits of the position.
- But if the position and the ptr are in the same page, then you can get the heap base:

```
def deobfuscate(val):  
    mask = 0xfff << 52  
    while mask:  
        v = val & mask  
        val ^= (v >> 12)  
        mask >>= 12  
    return val
```



# Double Free

- The free bit does not always mean free
  - (the one thing an indicator bit should do)
- Calling free on an already free'd chunk *can* add it to a free list twice
  - malloc tries to prevent trivial double frees (2 in a row)
  - the tcache checks an entire list (only 7)
- Fast bins are vulnerable!
- Modify the free metadata of the received chunk

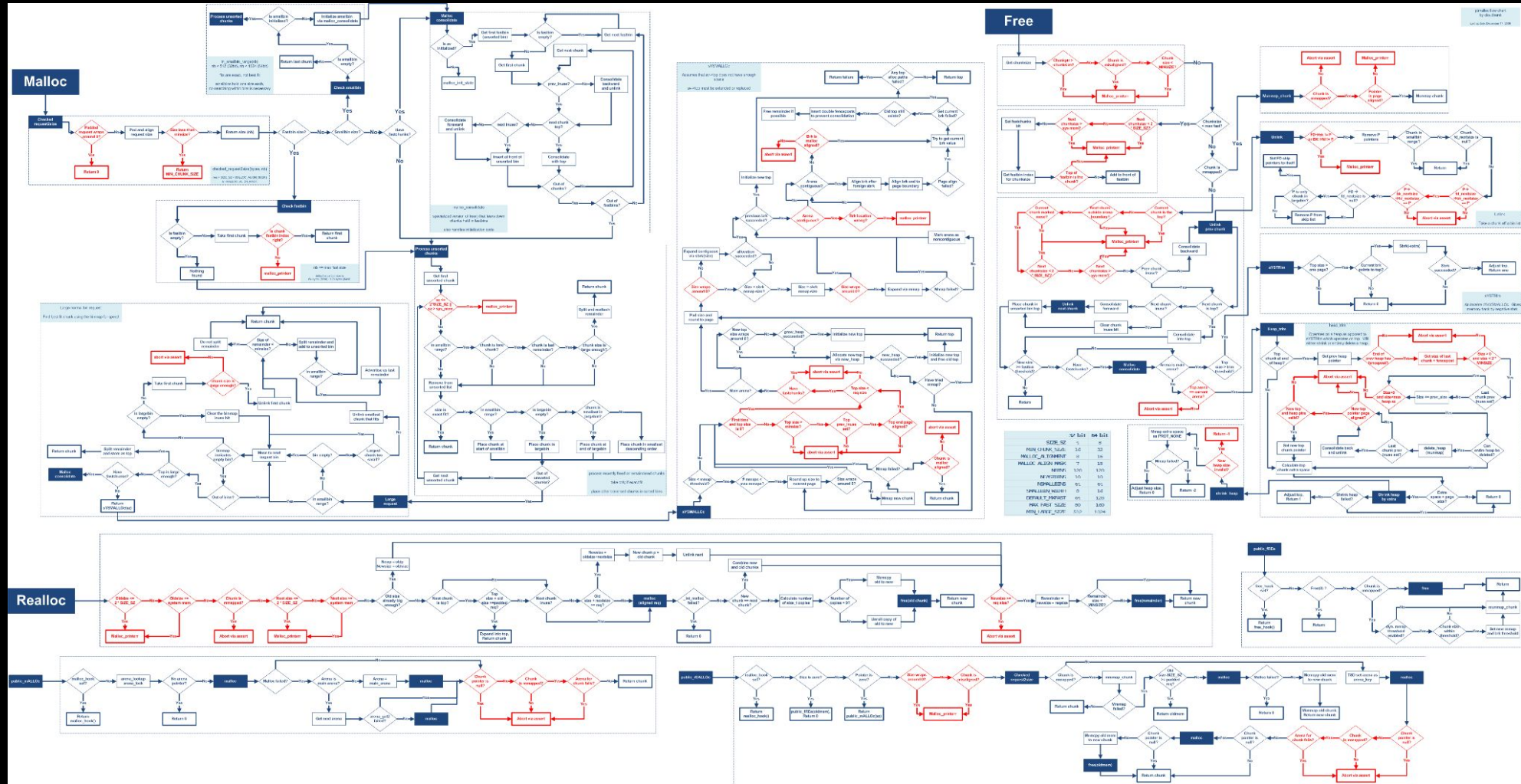


# Goals with Heap Exploitation

- Make malloc:
  - return a chunk somewhere interesting
  - write a value to an arbitrary address
  -
- Make libc merge invalid chunk



# A complete model of the Heap



# pwndbg tools

- heap
  - Displays the state & address of all heap chunks
- bins/tcachebins/fastbins/smallbins/largebins
  - Displays the state and chunks in each bin
- vis\_heap\_chunks
  - Shows a hex dump of the heap, color codes chunks, and marks chunks in bins





# Resources

<https://github.com/shellphish/how2heap>



# Resources

- <https://github.com/shellphish/how2heap>
  - has more resources linked!
- <https://azeria-labs.com/heap-exploitation-part-1-understanding-the-glibc-heap-implementation/>
- <https://heap-exploitation.dhavalkapil.com/>



# Next Meetings

## 2024-02-15 • This Thursday

- PWN IV: ROP with Akhil
- Learn how to complete PWN exploit chains and achieve RCE!

## 2024-02-16 • This Weekend

- LACTF
- UCLA's Major CTF Event! All are welcome!



ctf.sigpwny.com

**sigpwny{house\_of\_house\_of\_house}**

**Meeting content can be found at**  
**[sigpwny.com/meetings](https://sigpwny.com/meetings).**

