# The Heap

- the heap memory is managed by the programmer itself
- The Heap is very complex
- Malloc & Free control heap

## Heap Use Rules

- Must call Free()
  - ↳ "Memory Leak"
- Call Free() once
  - ↳ "Double Free Bug"
- Only call Free() with allocated Pointers
  - ↳ "Invalid free"
- Don't write/read past the end of

allocation

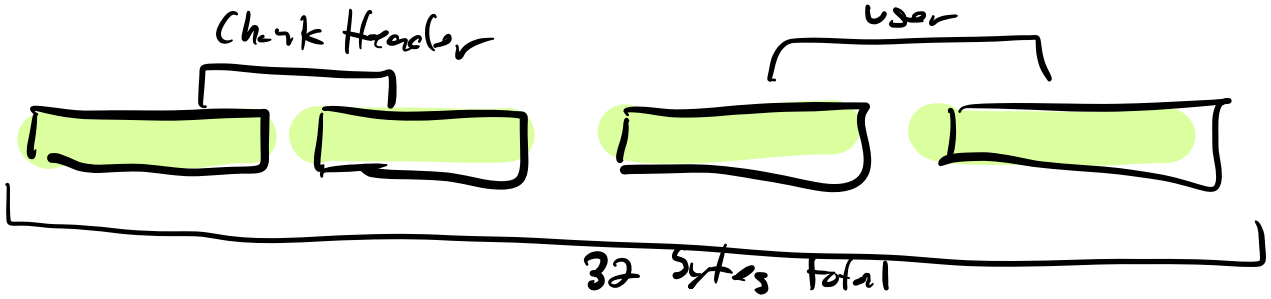
↳ Buffer Overflow

- Do not read / use uninitialized memory
  - ↳ "Uninitialized Reads"
  - "Information Leak"

★ these rules are useful To know In order to take advantage of Bad Programming

## How Heap Works

- Allocation contains Meta Data Region & Allocation contains User Data

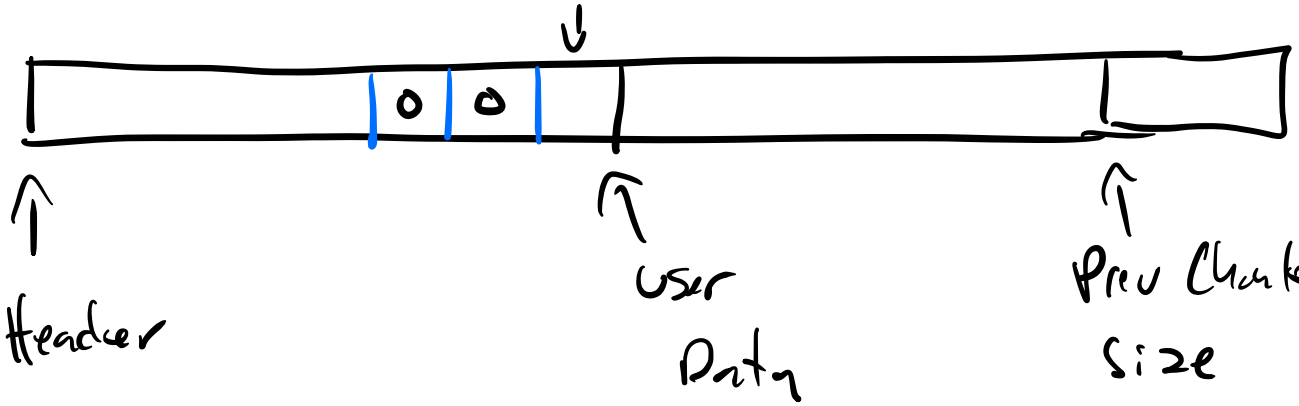


- Heap is placed after the text section when ASLR is not enabled
- Addresses inside Into Proc Map also will have random addrs inside stack / text sections
- 0x55555 = P.I.E enabled Binary
- Non P.I.E Binarys with ASLR dont make a difference
- Heap is created only when Malloc is called
- Malloc ensures User Data portion is aligned to 16 bytes

**GEF Command:** heap chunks

Meta Data

Prev Chunk
In use = 1




size_t = 8 -> 64
4 -> 32

- When chunks are "freed" they're not actually free'd
- they get put in a list that says remove these later -> A Bin

## GEF command: heap bins

- Malloc will reuse chunks that get moved to the bins
- If chunk in bin isn't correct size for next malloc, it will chop them,

off the top chunk

- 7 chunks can be held in a t-cache Bin
- All bins are implemented as a Linked Lists

# Heap 0 - 64

* Deployed on Shell 2 *

* Bug is on line 32 *

Can overflow d by manipulating the memcopy of strlength(argv[1])

Challenge contains a canary placed on the heap

Need to overflow the function pointer to call system

Puts & System take the same string argument

Run Program 256 times for each byte until you find get_call() addr

Set overflow to overwrite the first value of get_code()

puts can come in handy