Use after frees in Cyber Warfare

WACTF Exploitation 5

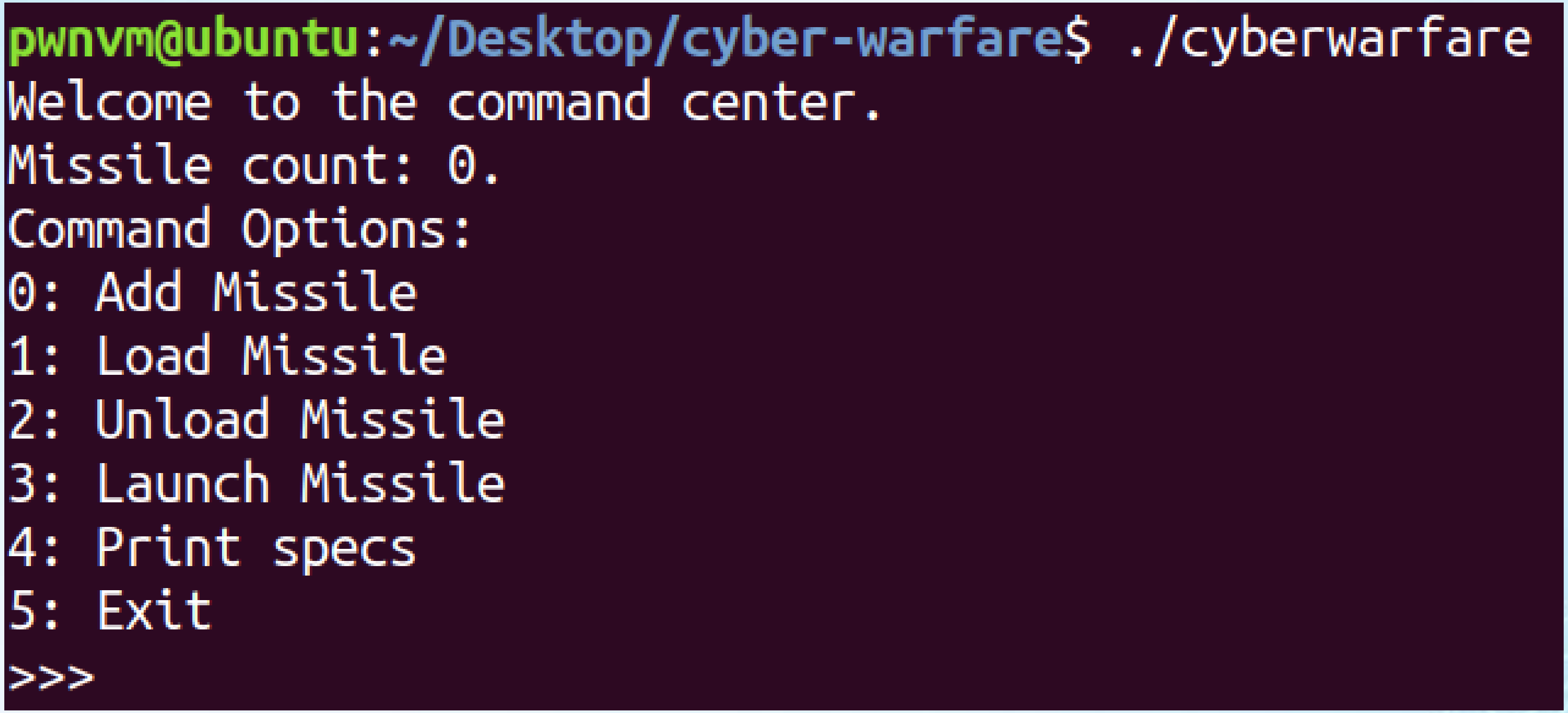
Who am I?

- Cyber Security Undergraduate at Curtin University.
- Browser Security Research Intern at InfoSect Canberra.
- Love finding and exploiting vulnerabilities in low level software.
- Love playing CTFs.

pwntools

- Python library built for binary exploitation
- Has built in capabilities for socket programming, interaction with a program’s stdin and stdout, and etc
- Allows us to automate a lot of the exploit

The challenge binary

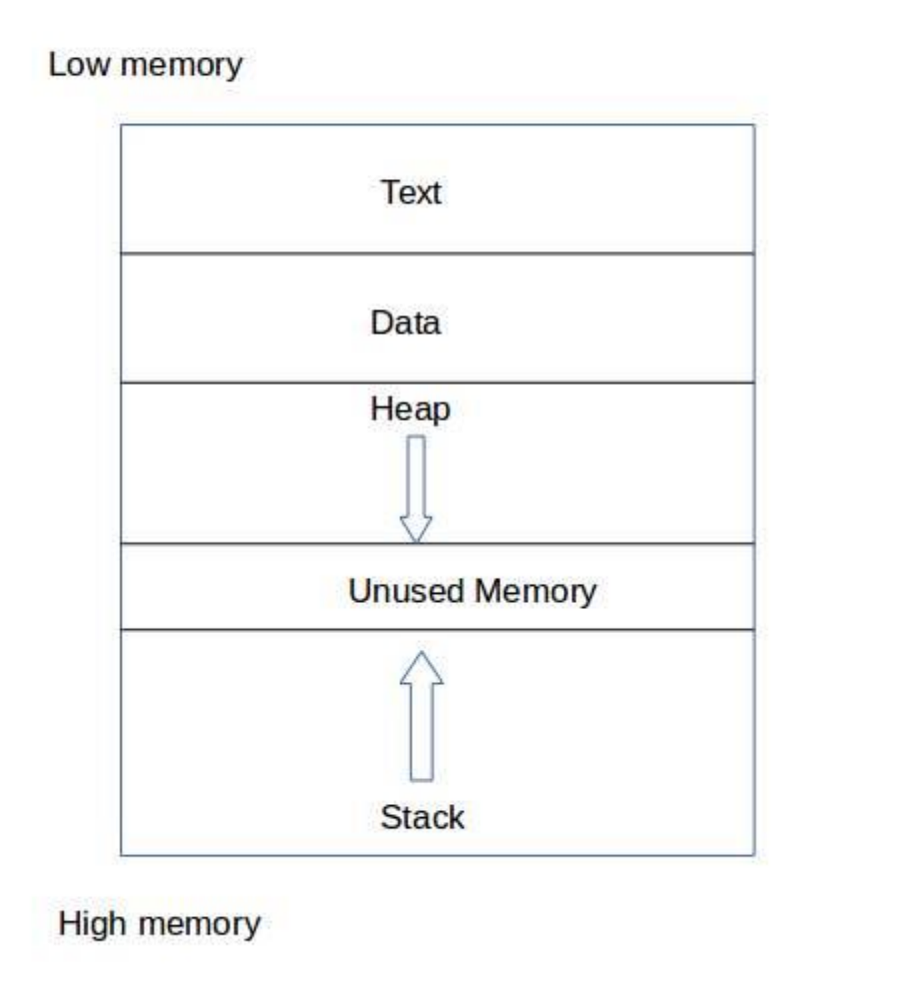


```
pwnvm@ubuntu:~/Desktop/cyber-warfare$ ./cyberwarfare
Welcome to the command center.
Missile count: 0.
Command Options:
0: Add Missile
1: Load Missile
2: Unload Missile
3: Launch Missile
4: Print specs
5: Exit
>>>
```

The challenge binary

- Full RELRO – Full Relocations Read Only
- Stack canaries enabled
- NX disabled
- PIE – Position Independent Executable
- ASLR – Address Space Layout Randomization

Linux memory mappings



Low memory

Text

Data

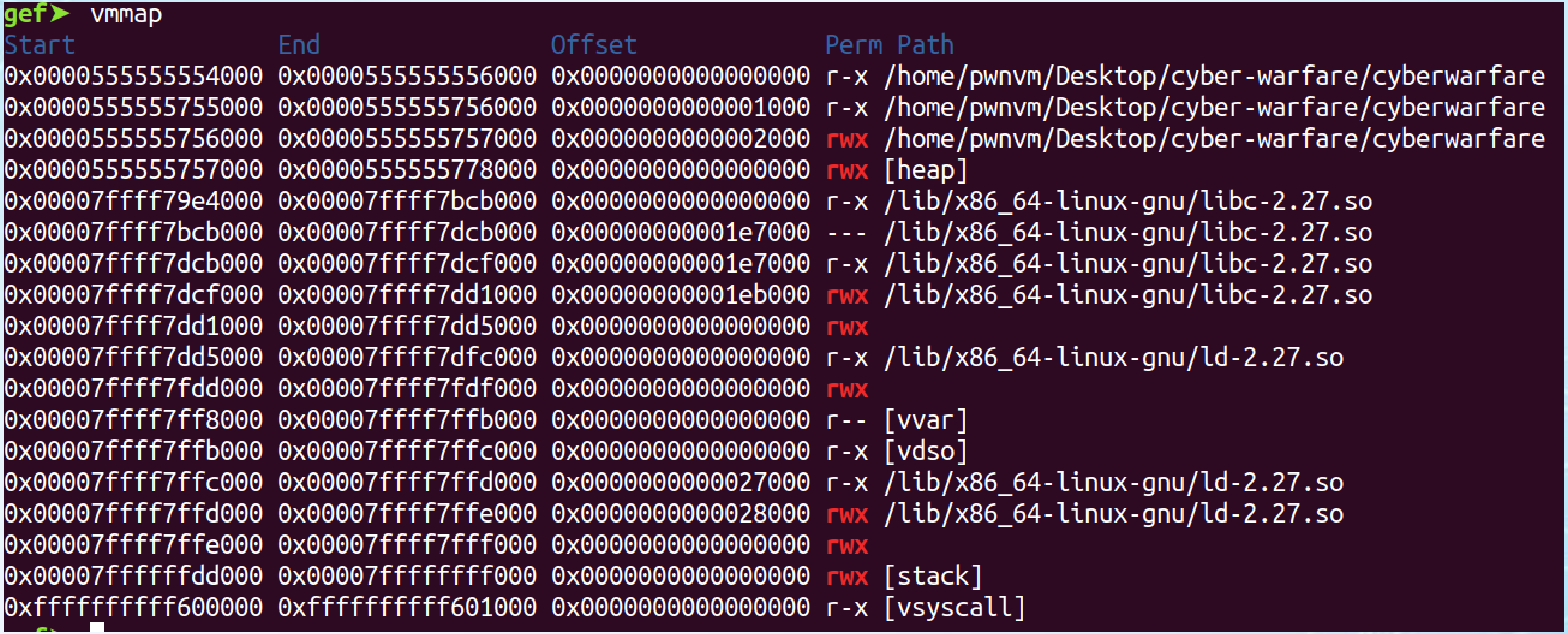
Heap

Unused Memory

Stack

High memory

Memory mapping



```
gef➤  vmmap
Start              End                Offset             Perm Path
0x0000555555554000 0x0000555555556000 0x0000000000000000 r-x /home/pwnvm/Desktop/cyber-warfare/cyberwarfare
0x0000555555755000 0x0000555555756000 0x0000000000001000 r-x /home/pwnvm/Desktop/cyber-warfare/cyberwarfare
0x0000555555756000 0x0000555555757000 0x0000000000002000 rwx /home/pwnvm/Desktop/cyber-warfare/cyberwarfare
0x0000555555757000 0x0000555555778000 0x0000000000000000 rwx [heap]
0x00007ffff79e4000 0x00007ffff7bcb000 0x0000000000000000 r-x /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7bcb000 0x00007ffff7dcb000 0x00000000001e7000 --- /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcb000 0x00007ffff7dcf000 0x00000000001e7000 r-x /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000 0x00007ffff7dd1000 0x00000000001eb000 rwx /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000 0x00007ffff7dd5000 0x0000000000000000 rwx
0x00007ffff7dd5000 0x00007ffff7dfc000 0x0000000000000000 r-x /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7fdd000 0x00007ffff7fdf000 0x0000000000000000 rwx
0x00007ffff7ff8000 0x00007ffff7ffb000 0x0000000000000000 r-- [vvar]
0x00007ffff7ffb000 0x00007ffff7ffc000 0x0000000000000000 r-x [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 0x0000000000027000 r-x /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000 0x00007ffff7ffe000 0x0000000000028000 rwx /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000 0x00007ffff7fff000 0x0000000000000000 rwx
0x00007ffffffdd000 0x00007ffffffff000 0x0000000000000000 rwx [stack]
0xffffffffff600000 0xffffffffff601000 0x0000000000000000 r-x [vsyscall]
```

Where are the two vulnerabilities?

Where are the two vulnerabilities?

- The Add Missile function – Format string vulnerability
  - `printf(missiles[missile_count]->name)`
- The Launch Missile function – Use after free vulnerability
  - Never unloads the missile

What is a format string vulnerability?

- Essentially a very easily avoidable programming error.
- Attacker controlled data being passed into the first argument of printf
- Recall that the definition of printf is as follows:
  int printf(const char *format, …)
- printf(missiles[missile_count]->name)
- printf(“%p %p %p”)

Leaking addresses from the stack

- When passing in “%p” to printf without any other arguments, printf actually starts looking for values on the stack to print out.
- The stack can contain addresses to the code section, libc, and the heap.
- This lets us bypass two mitigations
  - Address Space Layout Randomization (ASLR)
  - Position Independent Executable (PIE)

What is a use after free vulnerability?

- Being able to use a pointer after it’s been freed
- Much harder to exploit
- In the context of this challenge the use after free allows us to either:
  - Print out the name and description of the missile
  - Free it again, causing a double free.

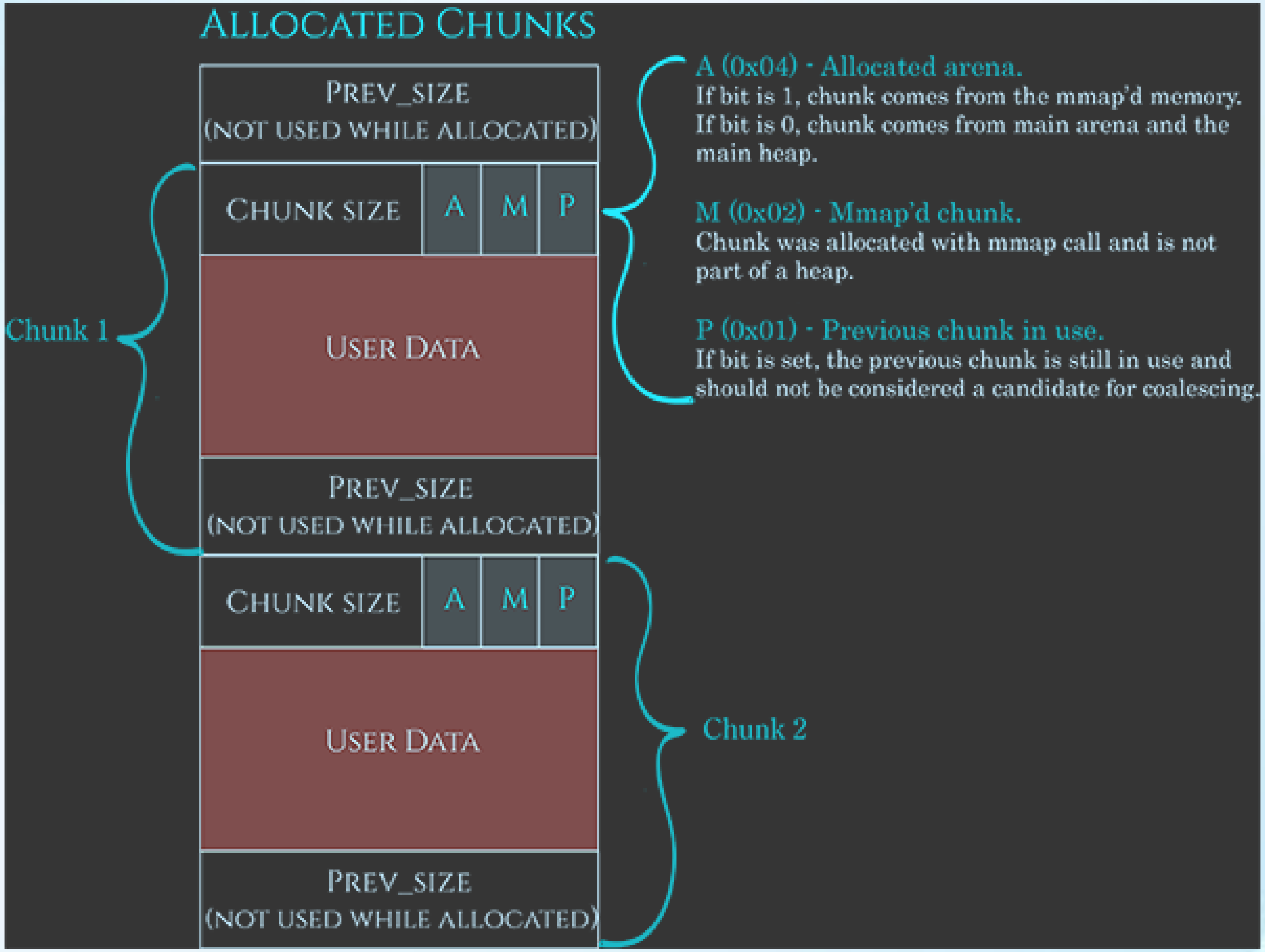
Malloc internals

- Every allocated chunk has a “chunk header” just above it.
- “Malloc doesn’t always return the exact number of bytes you ask for”
- This chunk header contains some important metadata information that the allocator uses.

Bins

- Bins are essentially free-lists
- Multiple bins for multiple types of chunks
- Freed chunks are stored in a LIFO linked list
- Chunks are inserted at the head
- fwd pointers are stored in the first 8 bytes of the chunk’s user data

Allocated chunks

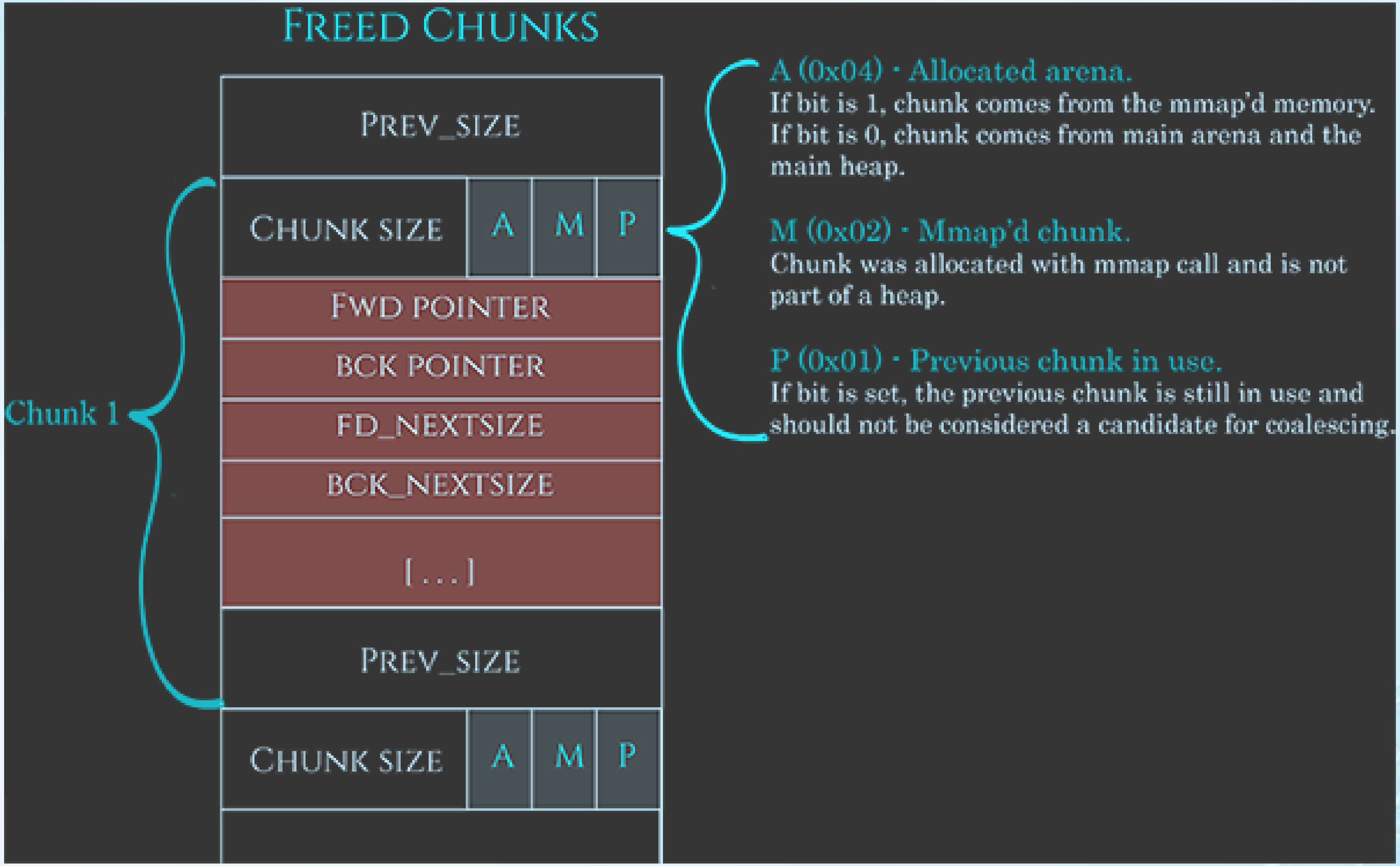


A (0x04) - Allocated arena.
If bit is 1, chunk comes from the mmap'd memory.
If bit is 0, chunk comes from main arena and the main heap.

M (0x02) - Mmap'd chunk.
Chunk was allocated with mmap call and is not part of a heap.

P (0x01) - Previous chunk in use.
If bit is set, the previous chunk is still in use and should not be considered a candidate for coalescing.

Freed chunks

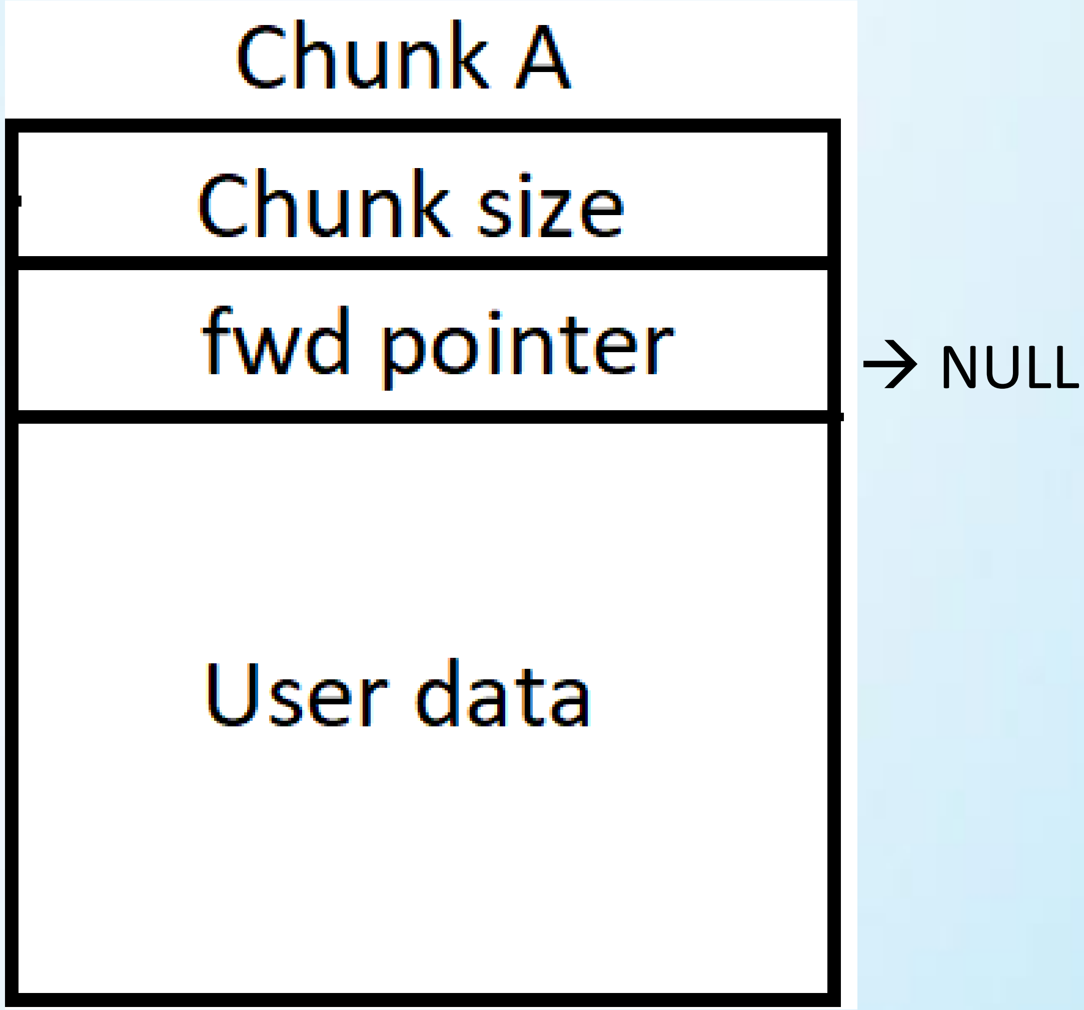


A (0x04) - Allocated arena.
If bit is 1, chunk comes from the mmap'd memory. If bit is 0, chunk comes from main arena and the main heap.

M (0x02) - Mmap'd chunk.
Chunk was allocated with mmap call and is not part of a heap.

P (0x01) - Previous chunk in use.
If bit is set, the previous chunk is still in use and should not be considered a candidate for coalescing.

FWD pointers



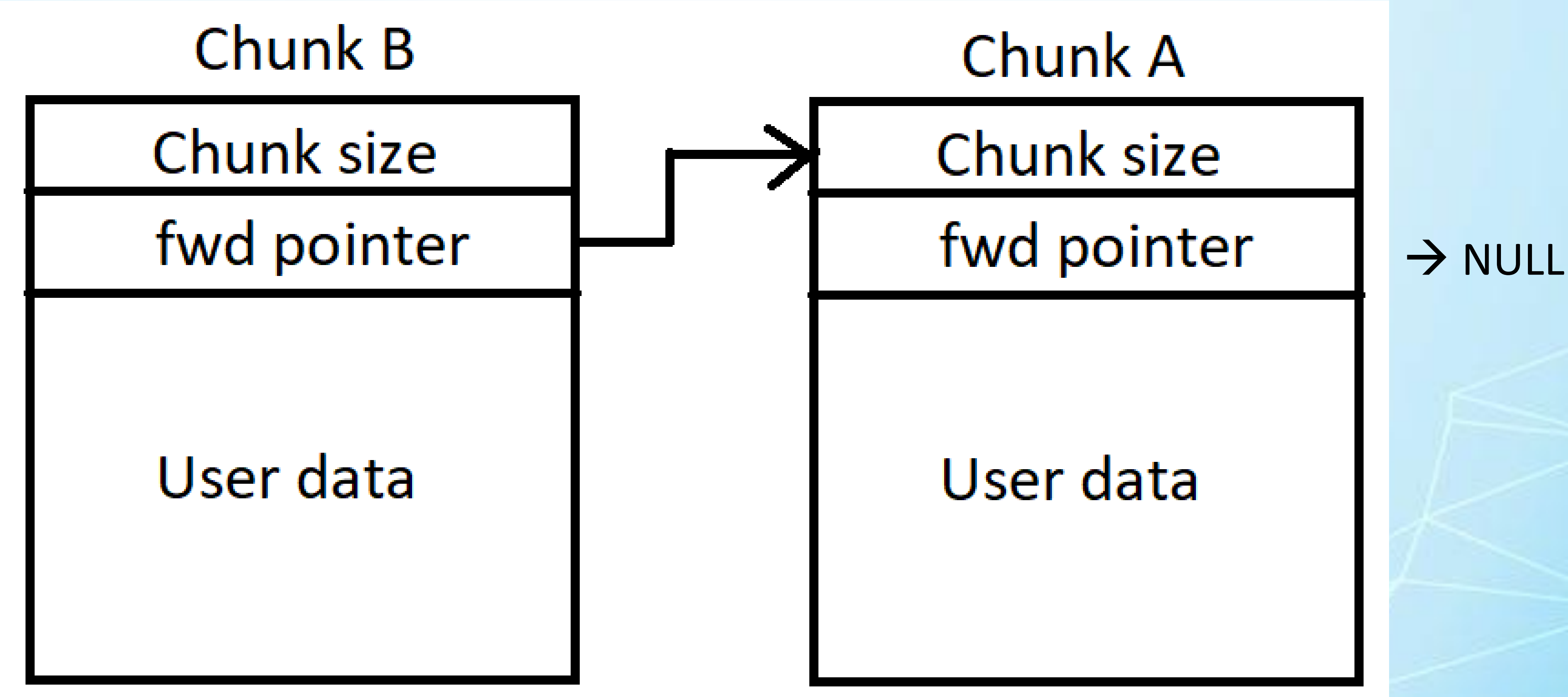
Chunk A

Chunk size

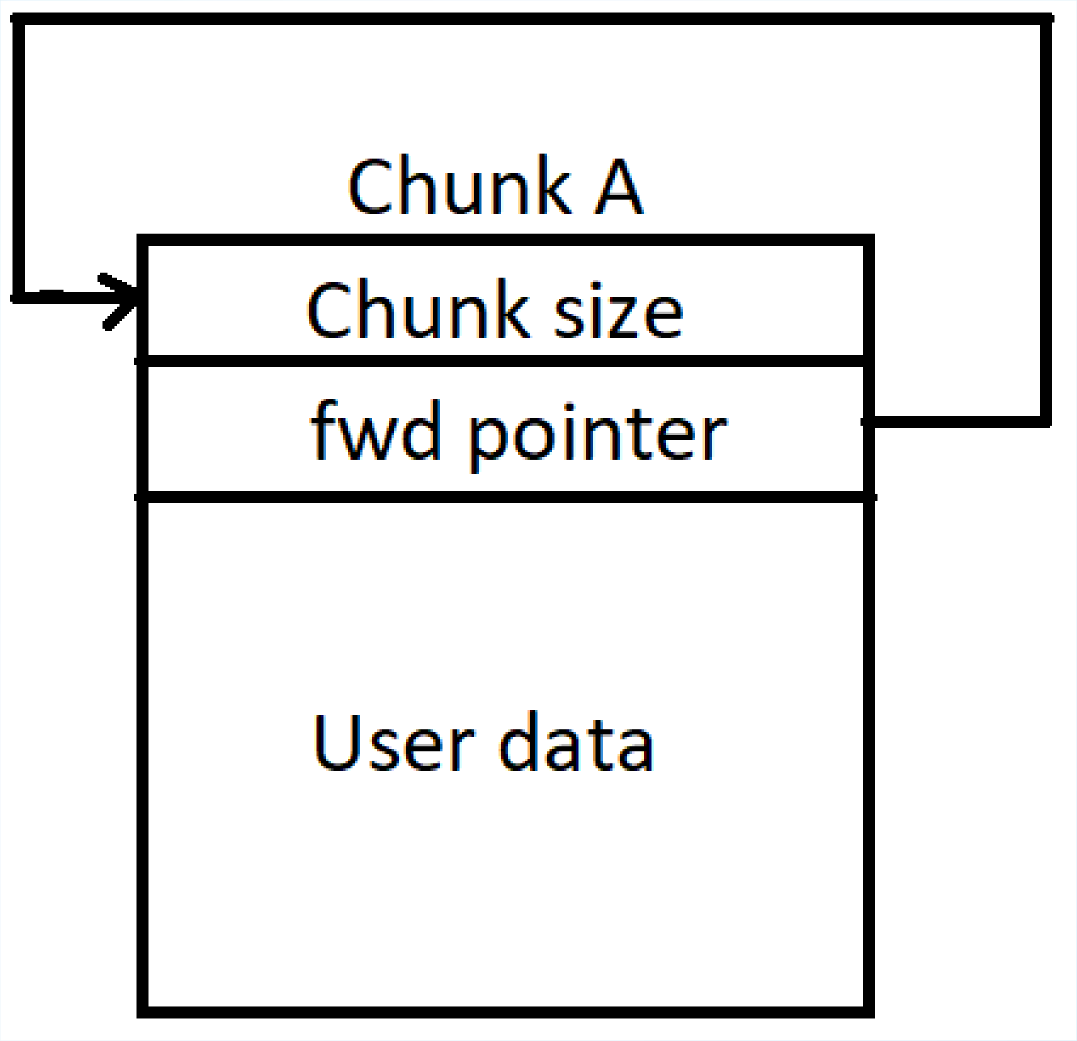
fwd pointer → NULL

User data

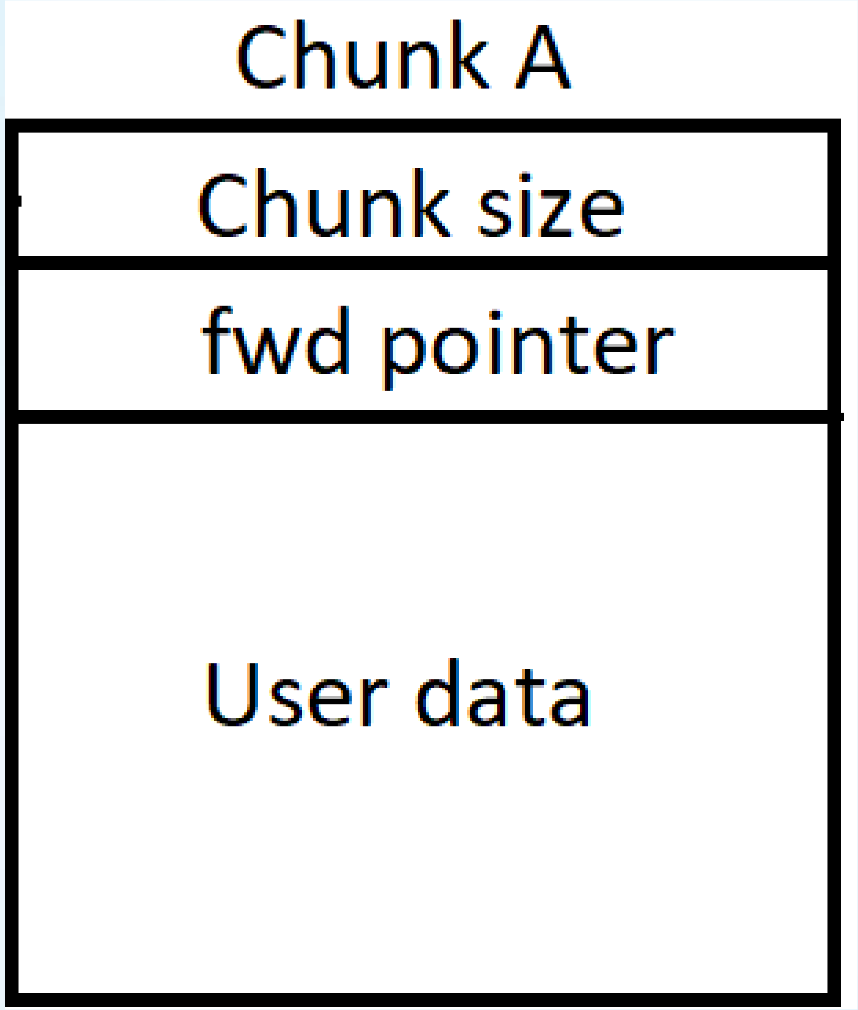
FWD pointers



Double free

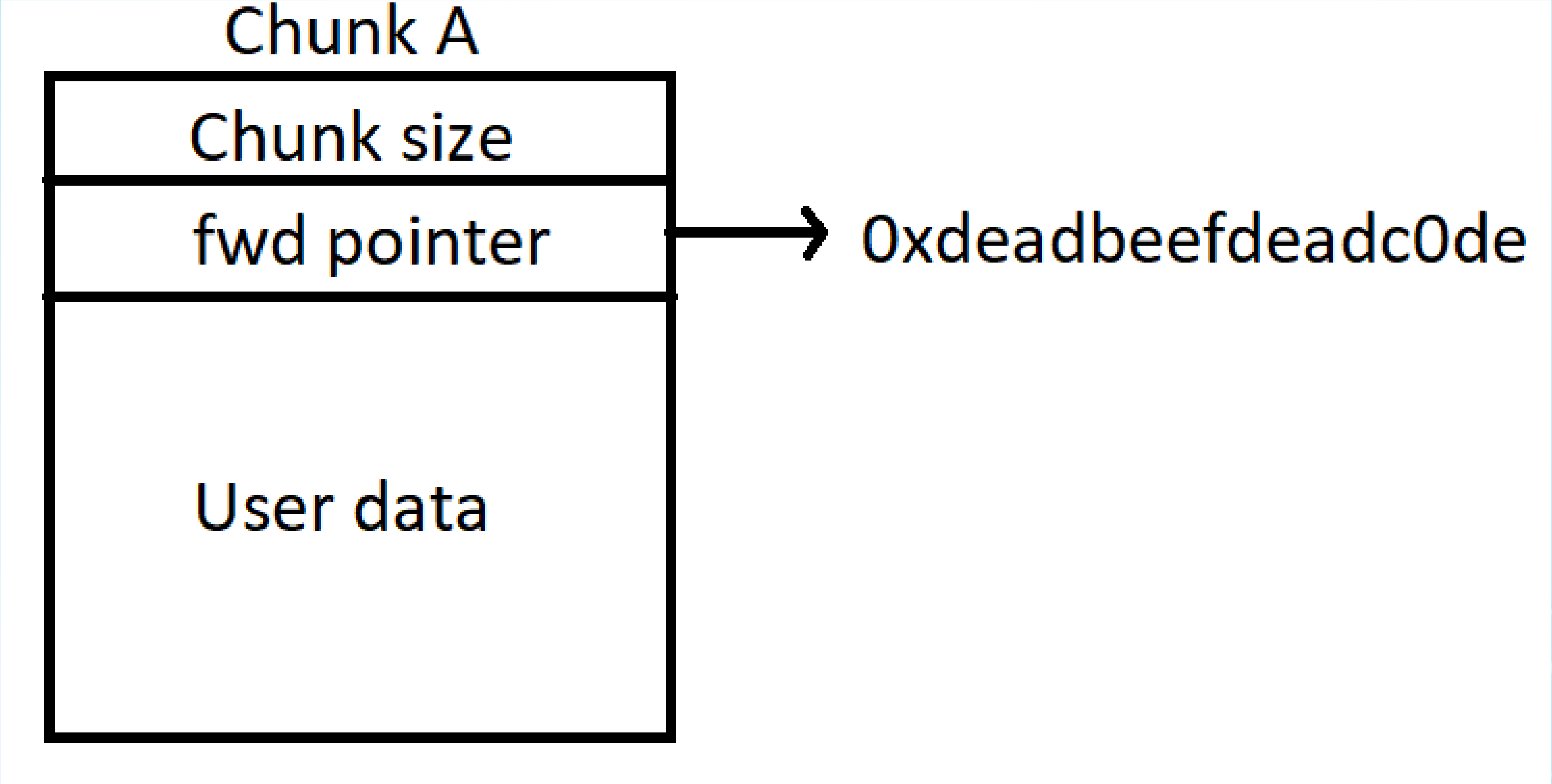


Double free – one allocation



<- Also user data!

Double free – one allocation



Chunk A

Chunk size

fwd pointer

0xdeadbeefdeadc0de

User data

Plan of attack

- Use the format string vulnerability to find the binary’s base address
  - Subsequently find the address of print_secret_codes
- Use the double free to leak an address on the heap
- Overwrite one of the menu function pointers on the heap with the secret function’s address using the double free

Time for a demo!

The ending

- Questions?

- Twitter: @farazsth98