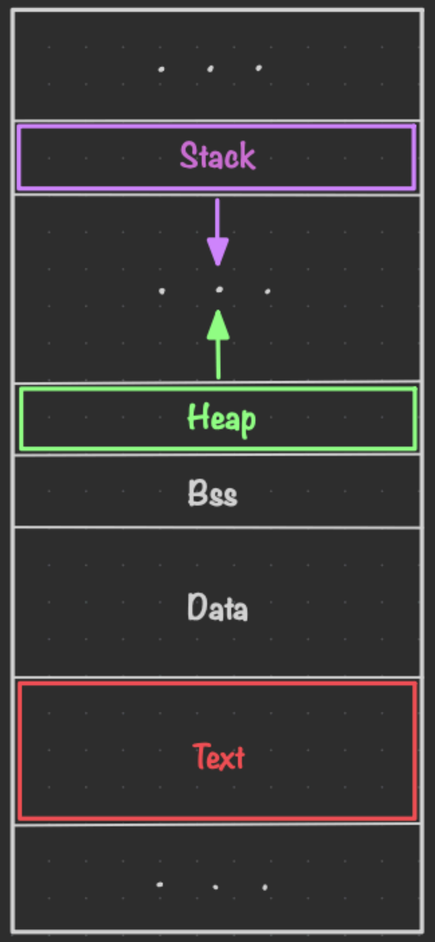
Theory :

- Fundamentals
- Buffer Overflow
- Stack Overflow
- Format String
- Heap Overflow

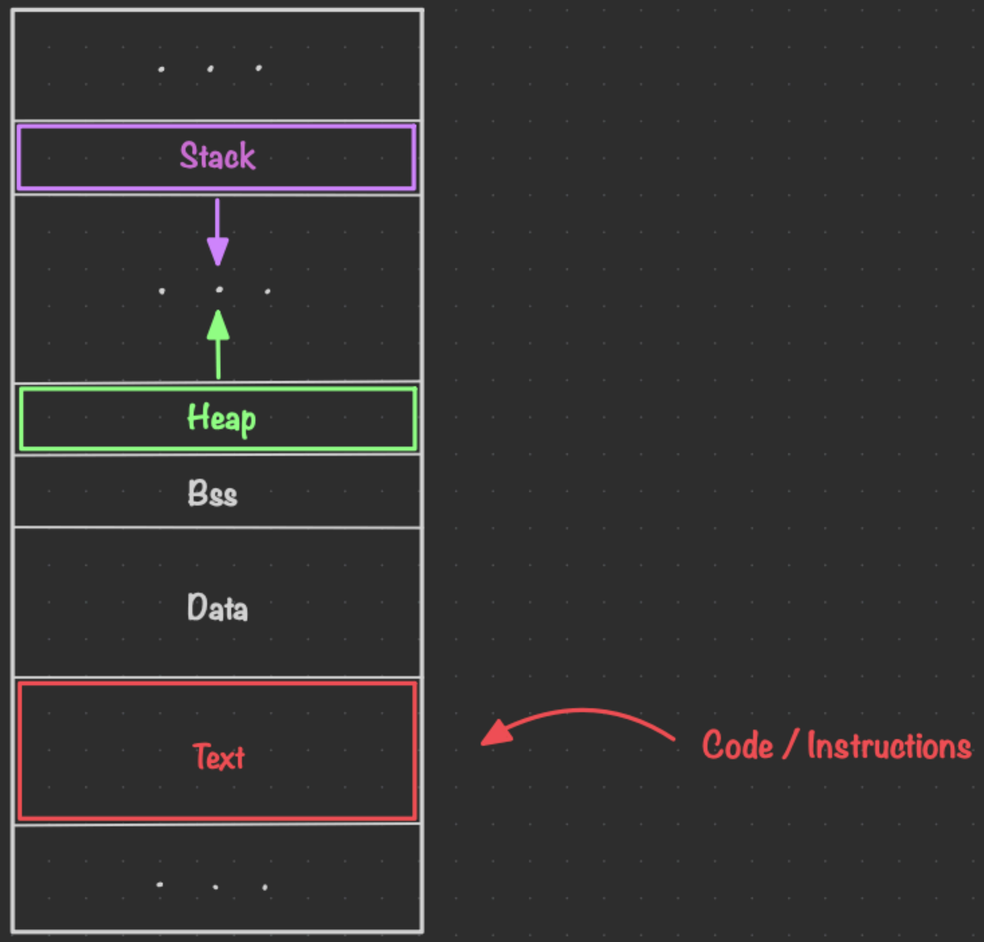
Practice... 😈

Buffer Overflow

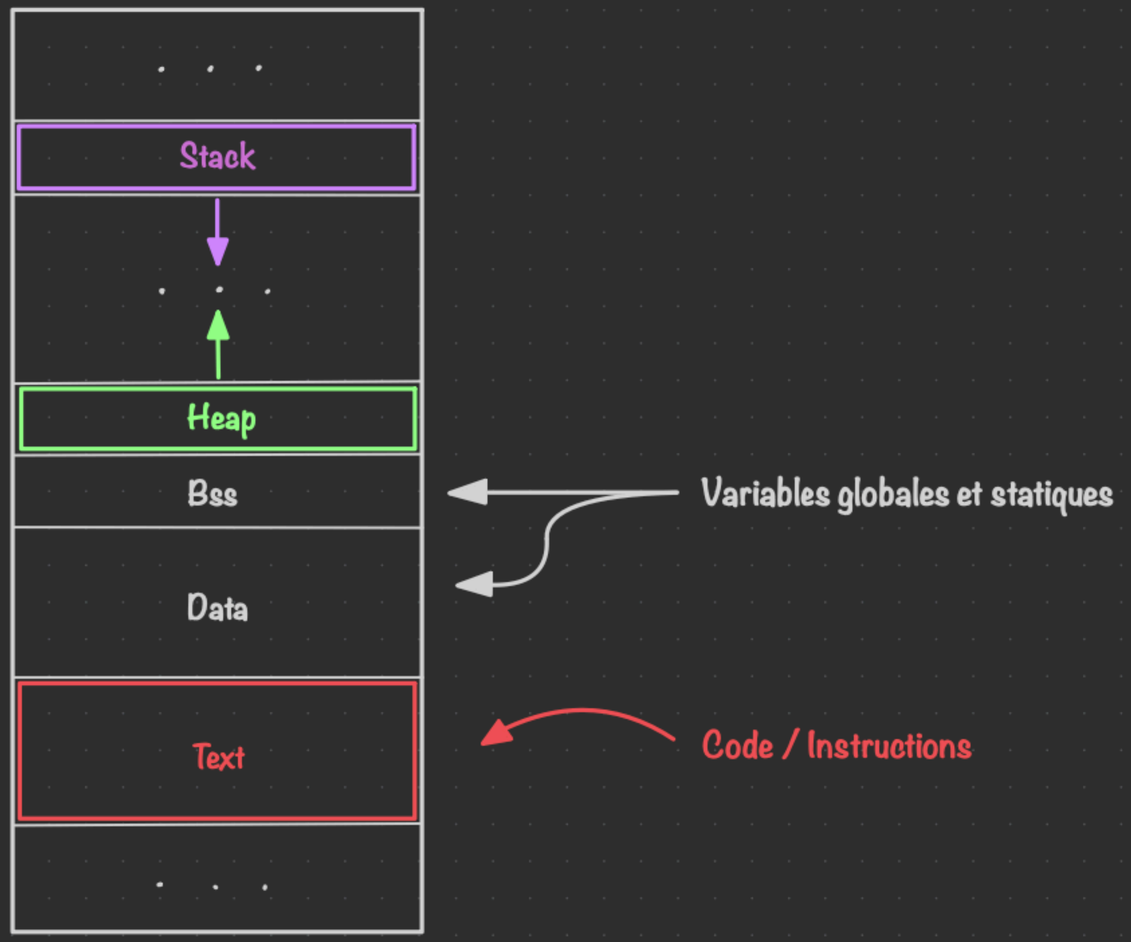
La Mémoire



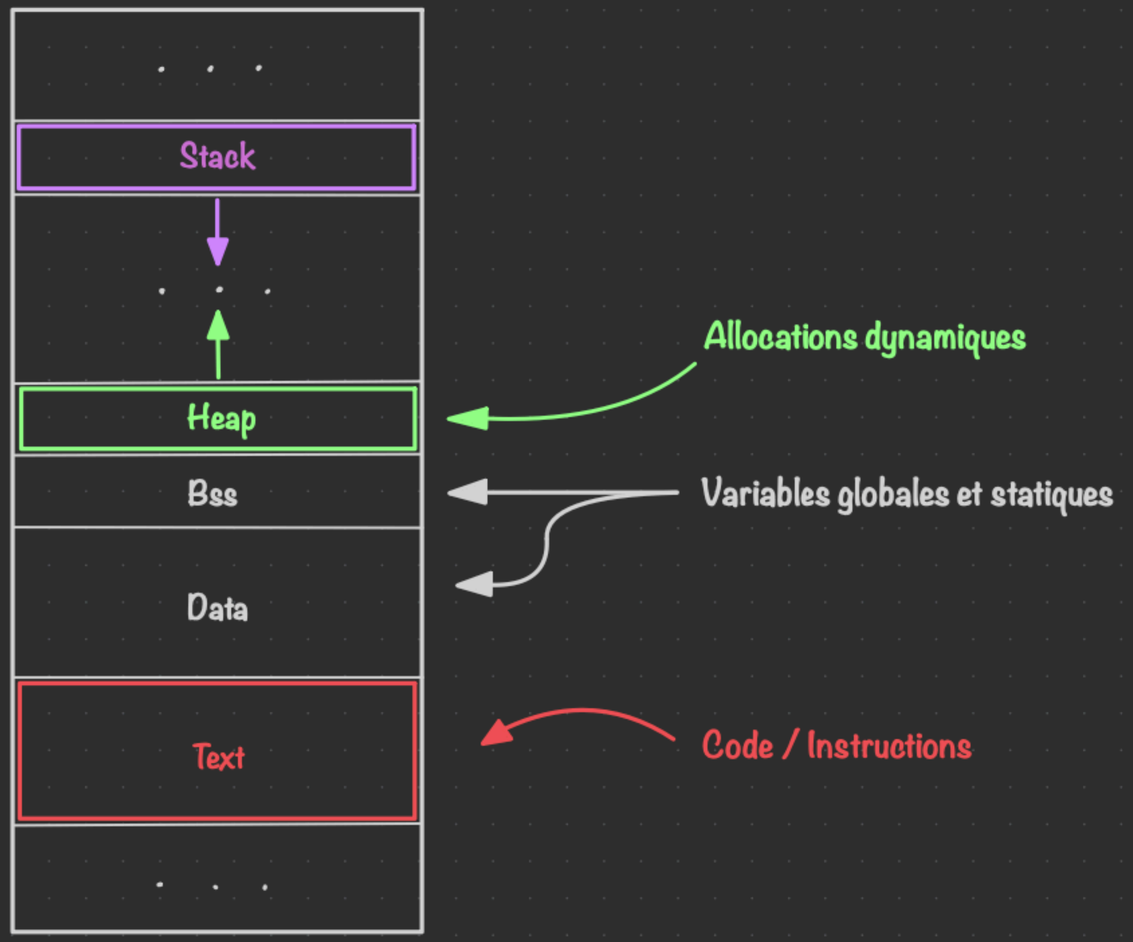
La Mémoire



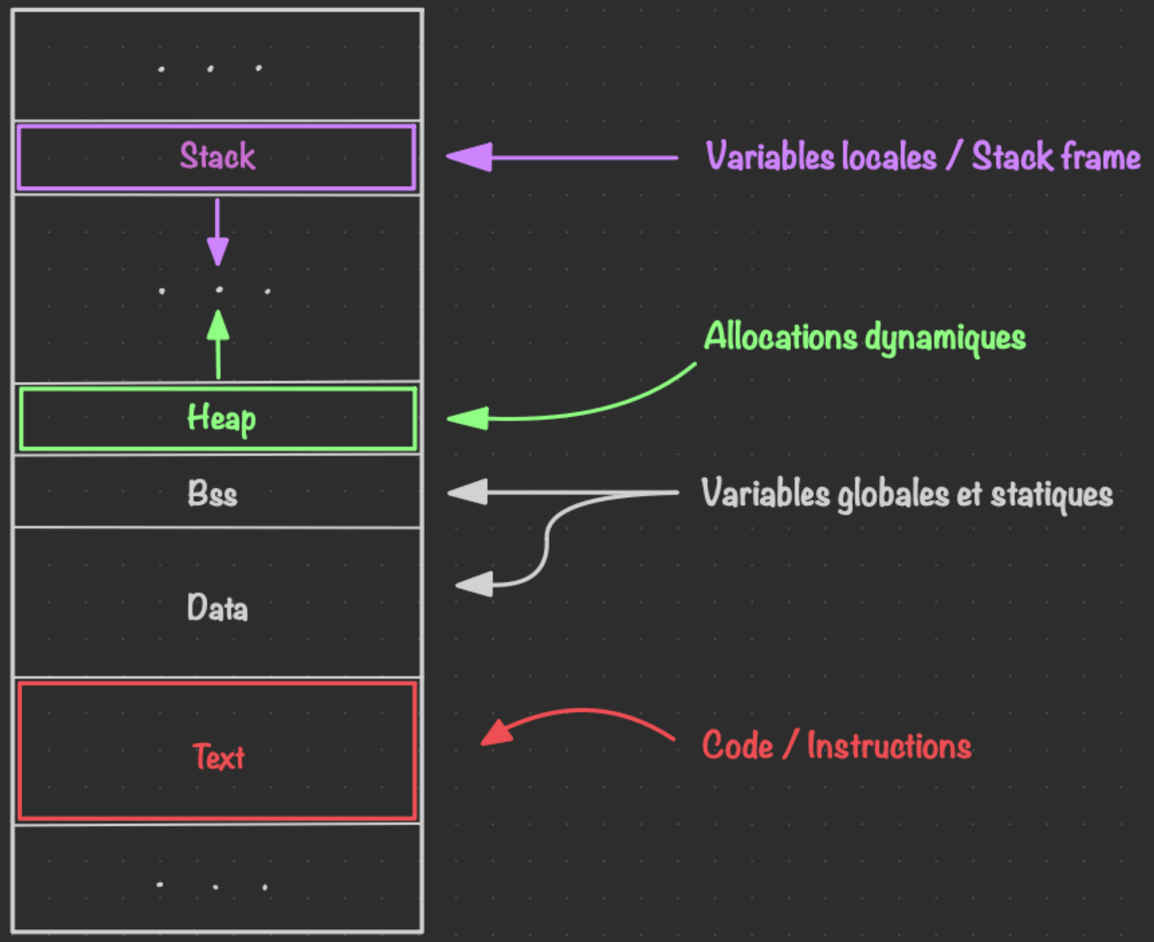
La Mémoire


Stack
Heap
Bss
Data
Text
Variables globales et statiques
Code / Instructions

La Mémoire

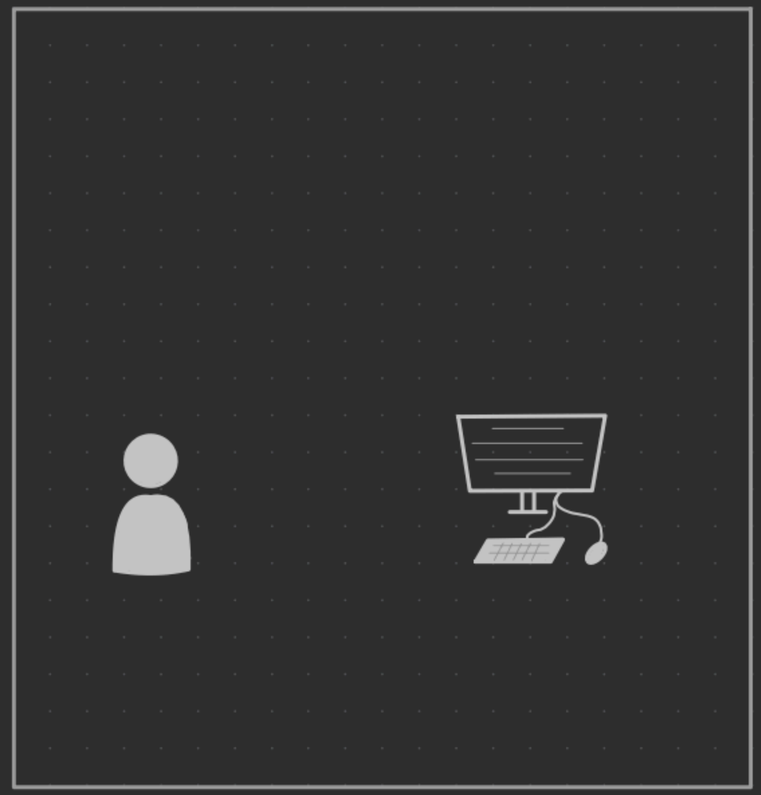

Stack
Heap
Bss
Data
Text
Allocations dynamiques
Variables globales et statiques
Code / Instructions

La Mémoire


Stack
Heap
Bss
Data
Text
Variables locales / Stack frame
Allocations dynamiques
Variables globales et statiques
Code / Instructions

Buffer overflow

```
char buffer[10] = {0};
puts("What's you name?");
scanf("%s", buffer);
```

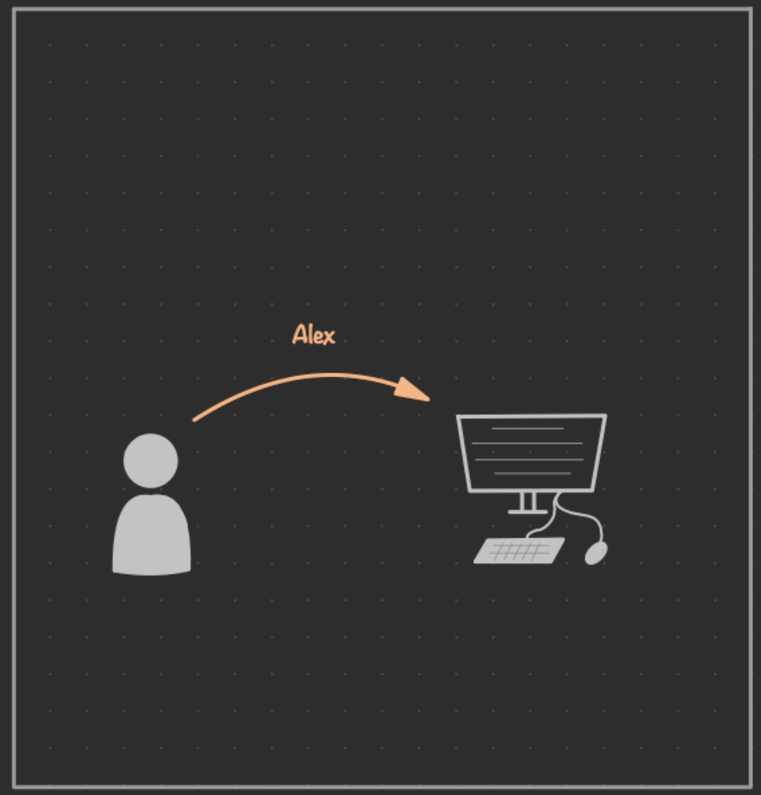


Buffer overflow

```
char buffer[10] = {0};
puts("What's you name?");
scanf("%s", buffer);
```

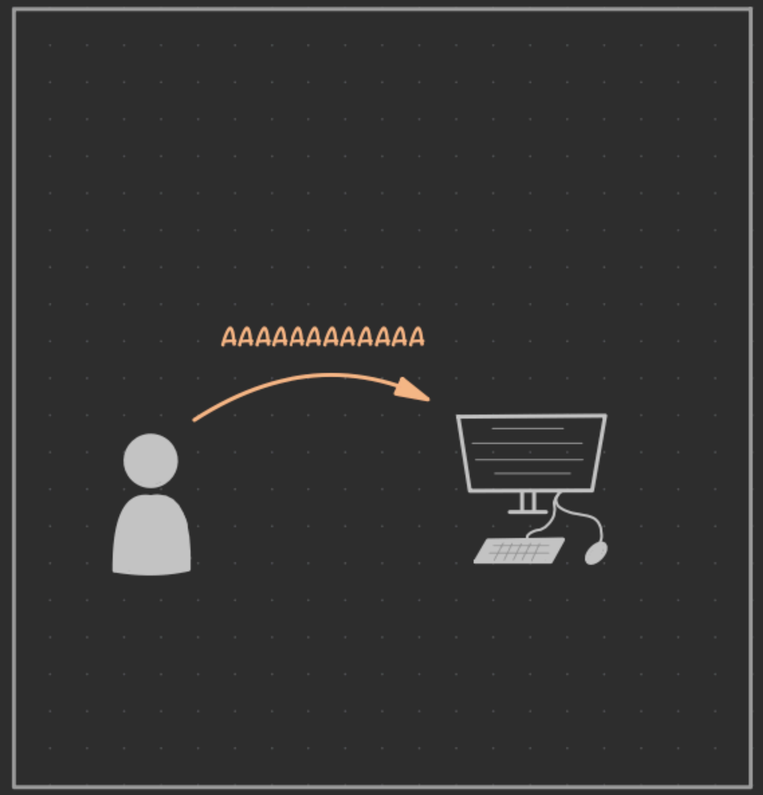
Buffer overflow

```
char buffer[10] = {0};
puts("What's you name?");
scanf("%s", buffer);
```



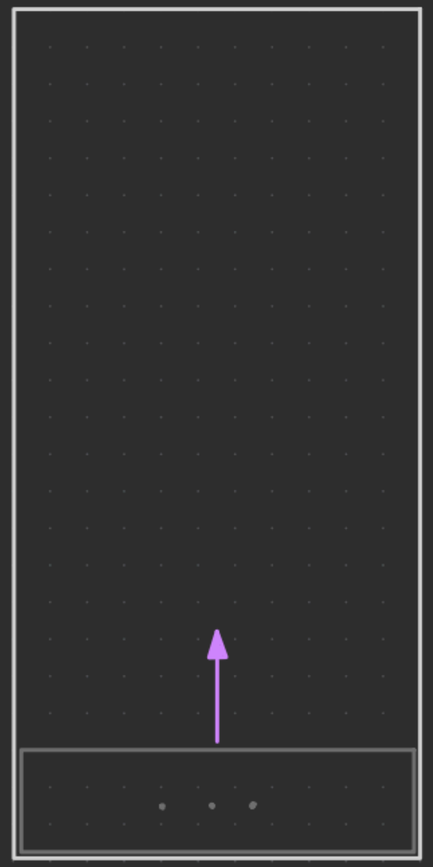
Buffer overflow

```
char buffer[10] = {0};
puts("What's you name?");
scanf("%s", buffer);
```



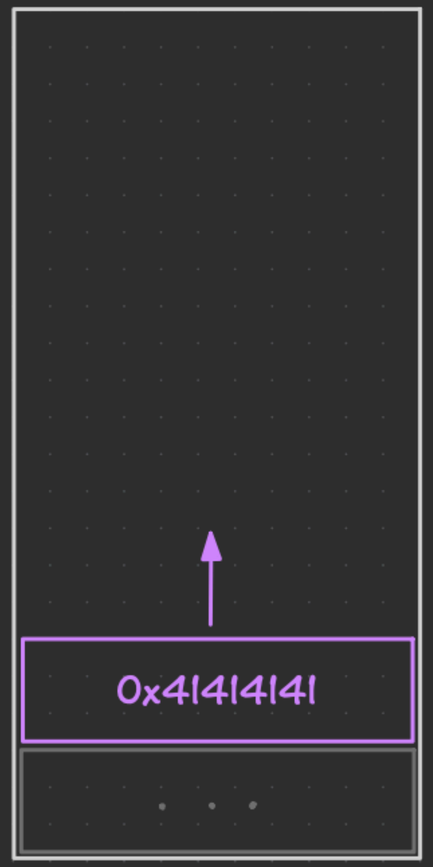
Stack Overflow

La Stack



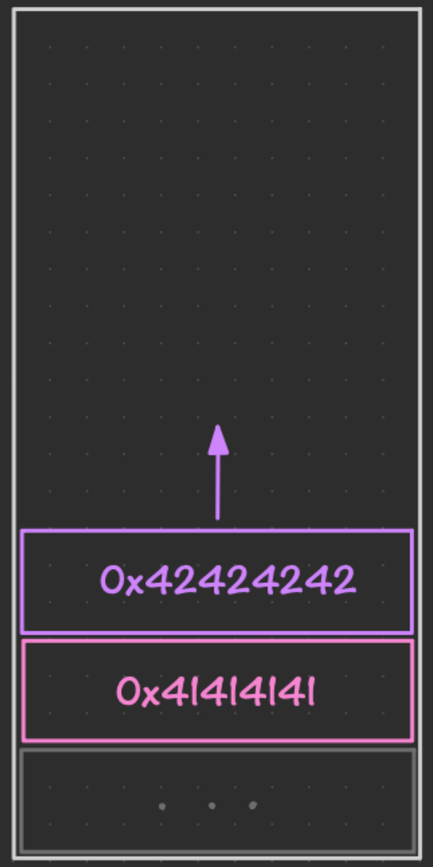
```
push 0x41414141
push 0x42424242
pop  eax
push 0x43434343
```

La Stack



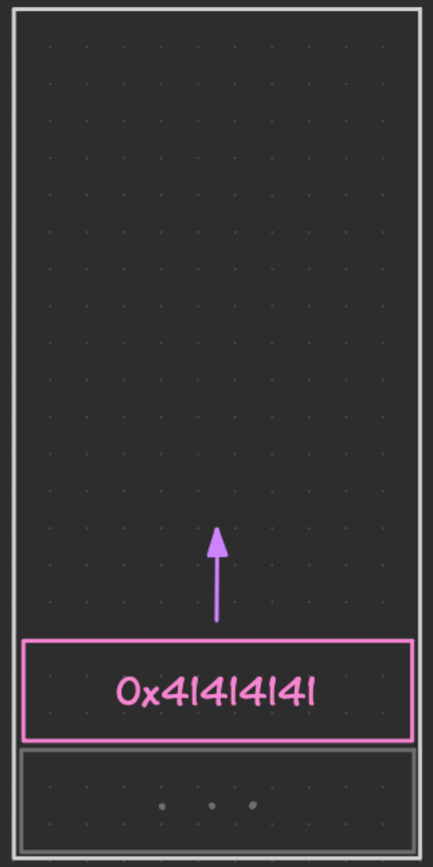

```
push 0x41414141
push 0x42424242
pop  eax
push 0x43434343
```

La Stack




```
push 0x41414141
push 0x42424242
pop  eax
push 0x43434343
```

La Stack




```
push 0x41414141
push 0x42424242
pop  eax
push 0x43434343
```

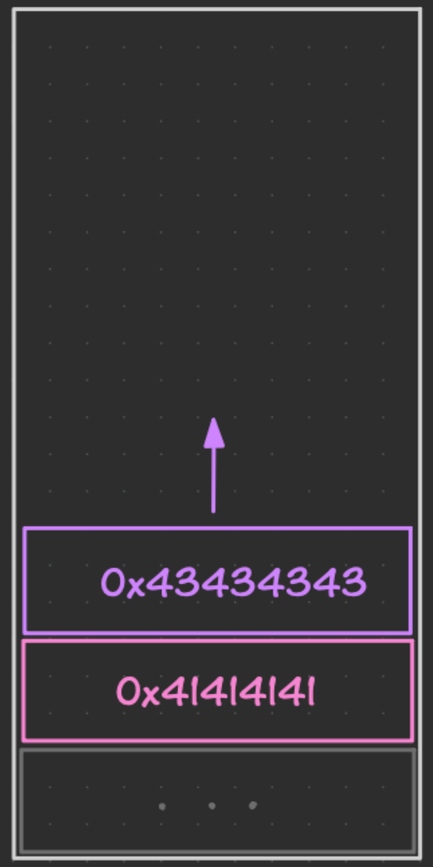
eax = 0x42424242

ebx = ...

ecx = ...

...

La Stack




```
push 0x41414141
push 0x42424242
pop  eax
push 0x43434343
```

eax = 0x42424242

ebx = ...

ecx = ...

...

C calling convention

```
void hello(){ printf("Hello World!"); }

int main(){
    hello();
    return 0;
}
```


```
main:
    call   hello
    mov    rax, 0
    leave
    ret
```

C calling convention

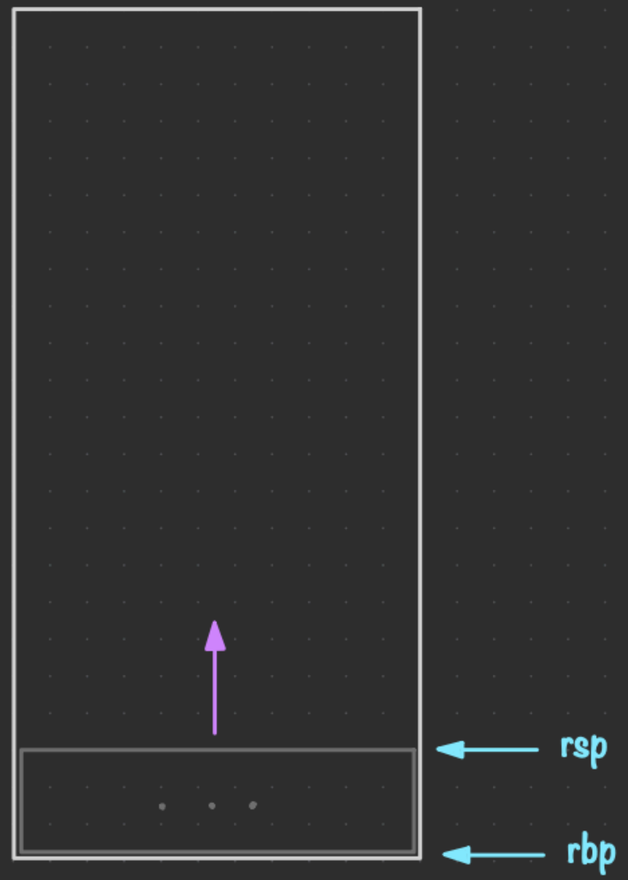
```
void hello(){ printf("Hello World!"); }

int main(){
    hello();
    return 0;
}
```

Compilation

```
main:
    call   hello
    mov    rax, 0
    leave
    ret
```

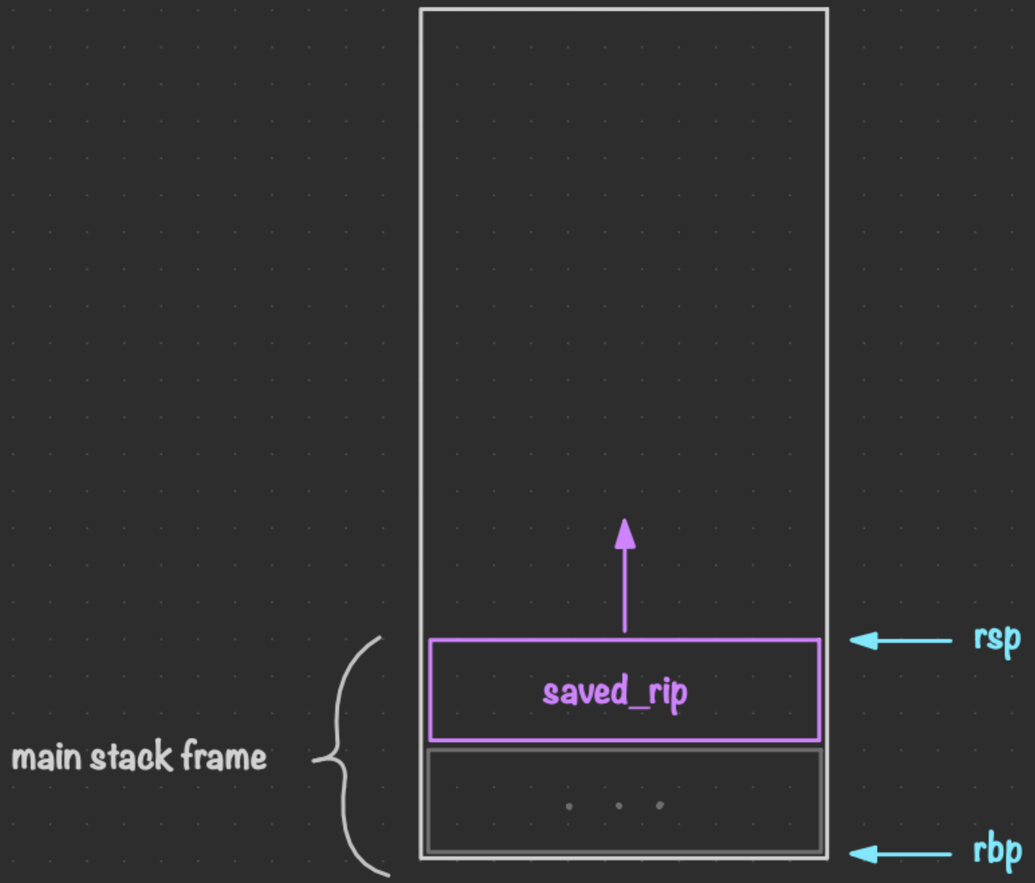
```
push   $rip
jmp    <hello_addr>
```

C calling convention



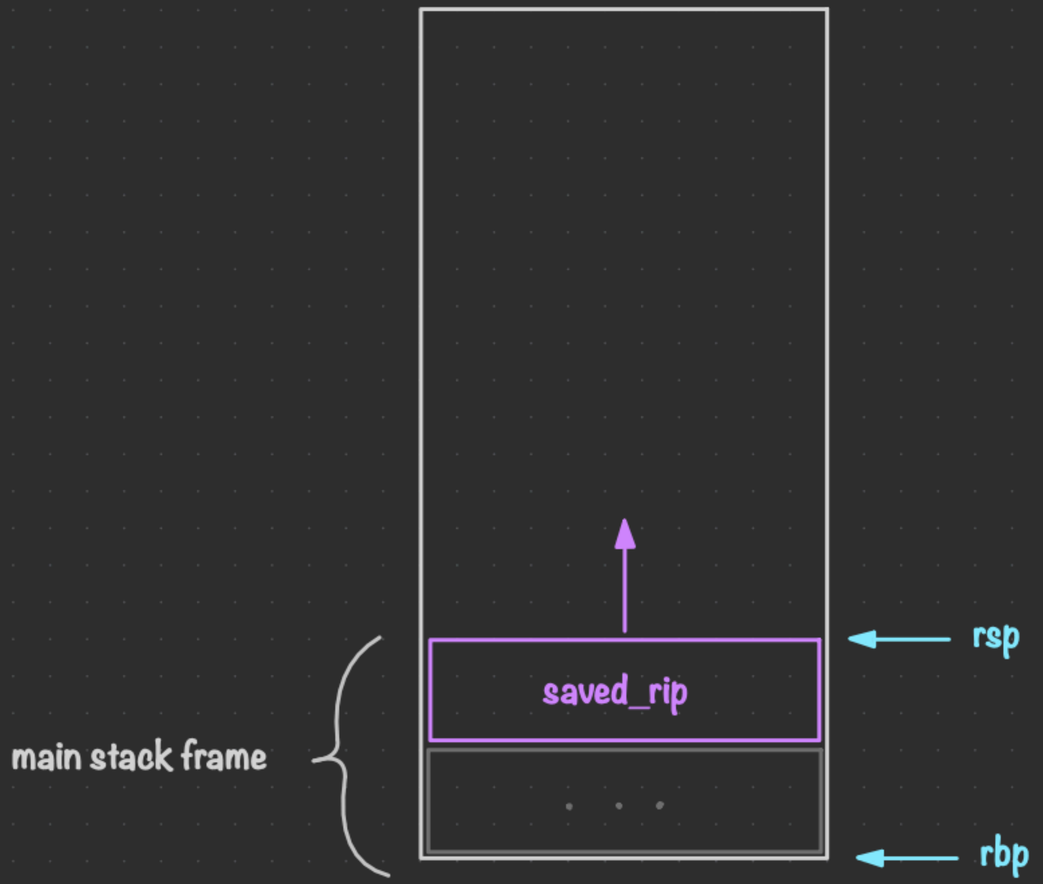

```
main:
    push    $rip
    jmp     <hello_addr>


hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

C calling convention



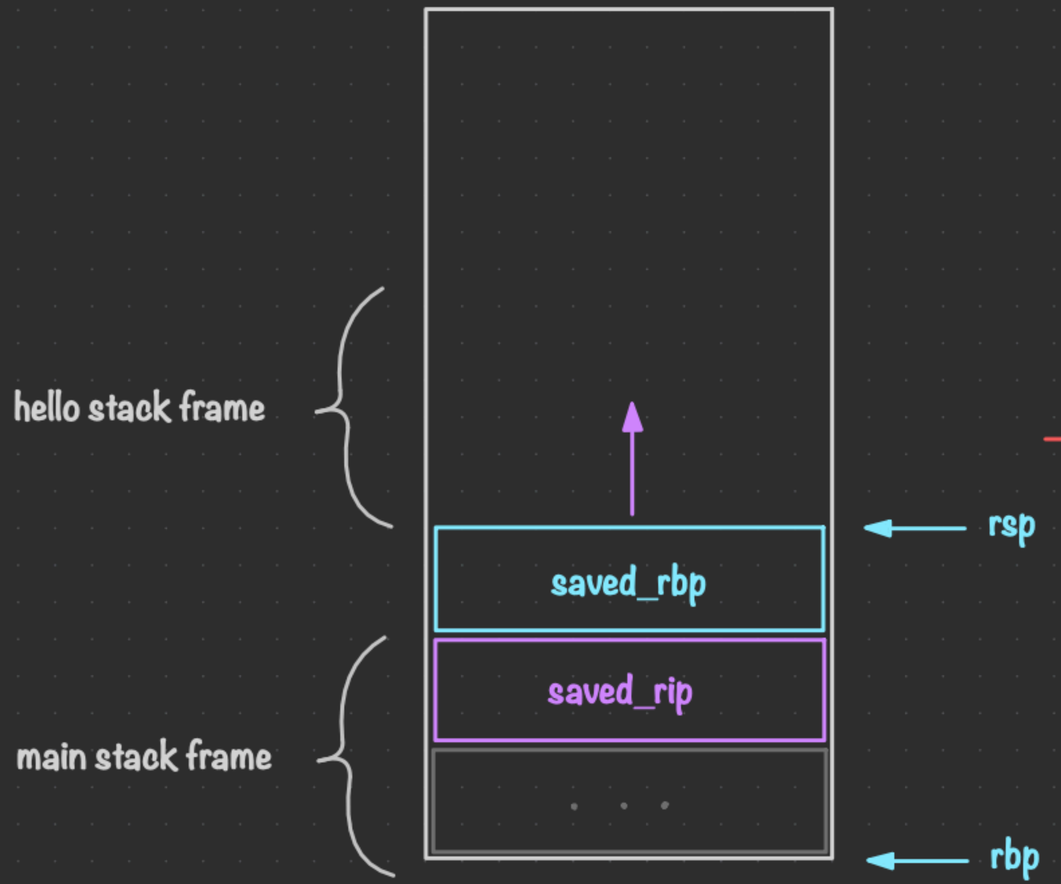

```
main:
    push    $rip
    jmp     <hello_addr>


hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

C calling convention



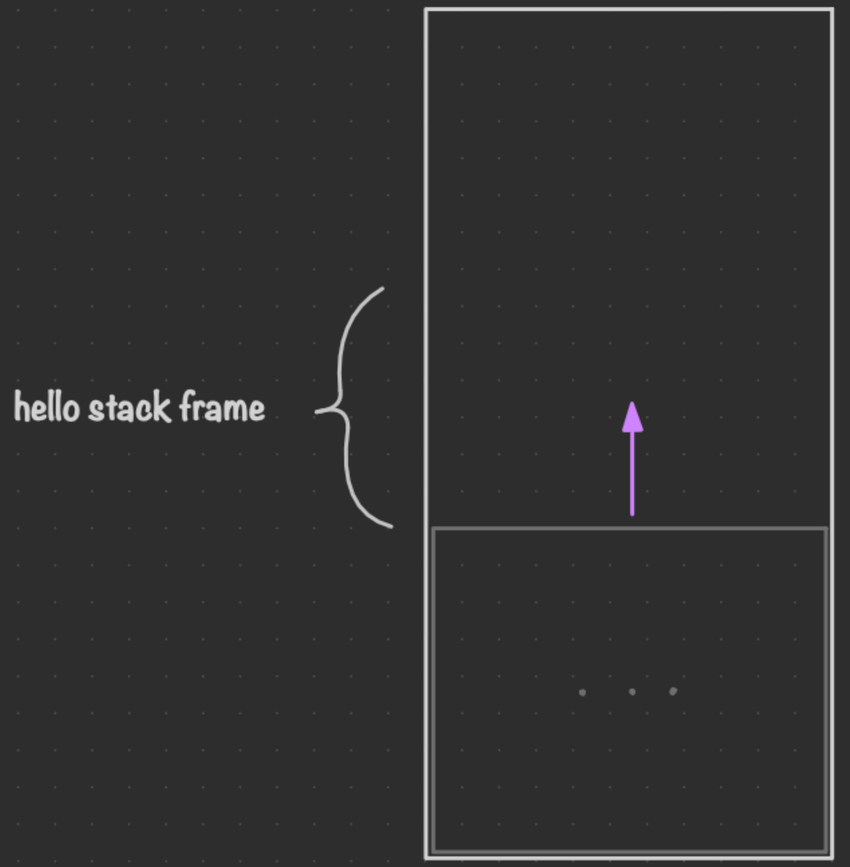

```
main:
    push    $rip
    jmp     <hello_addr>

hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

C calling convention




```
main:
    push    $rip
    jmp     <hello_addr>

hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

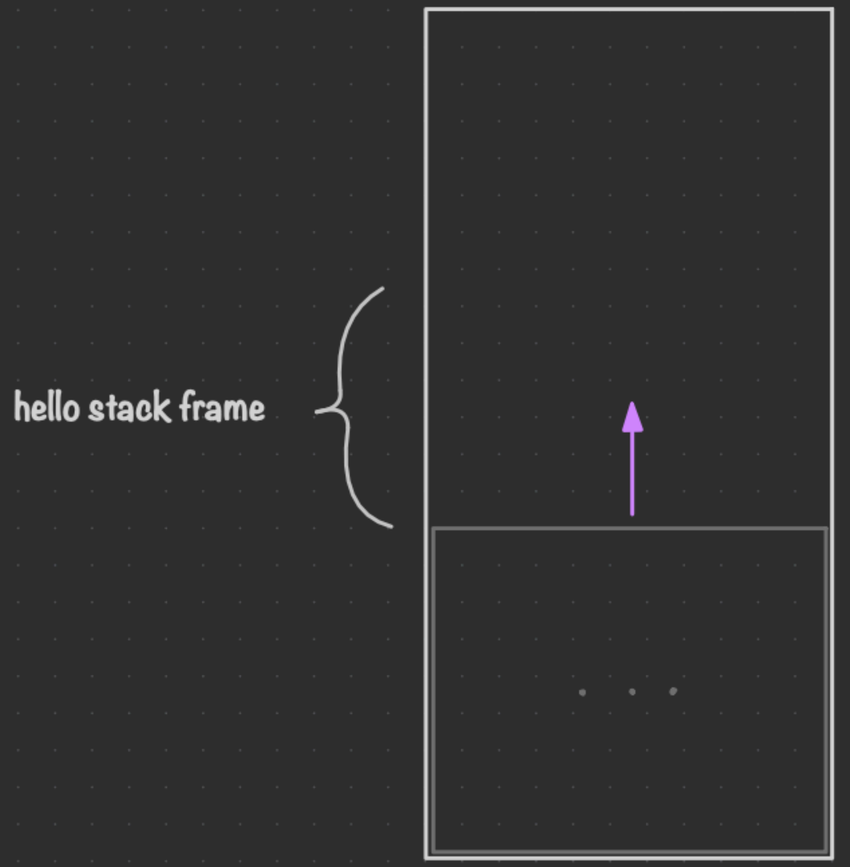
C calling convention



rsp, rbp

```
main:
    push    $rip
    jmp     <hello_addr>

hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

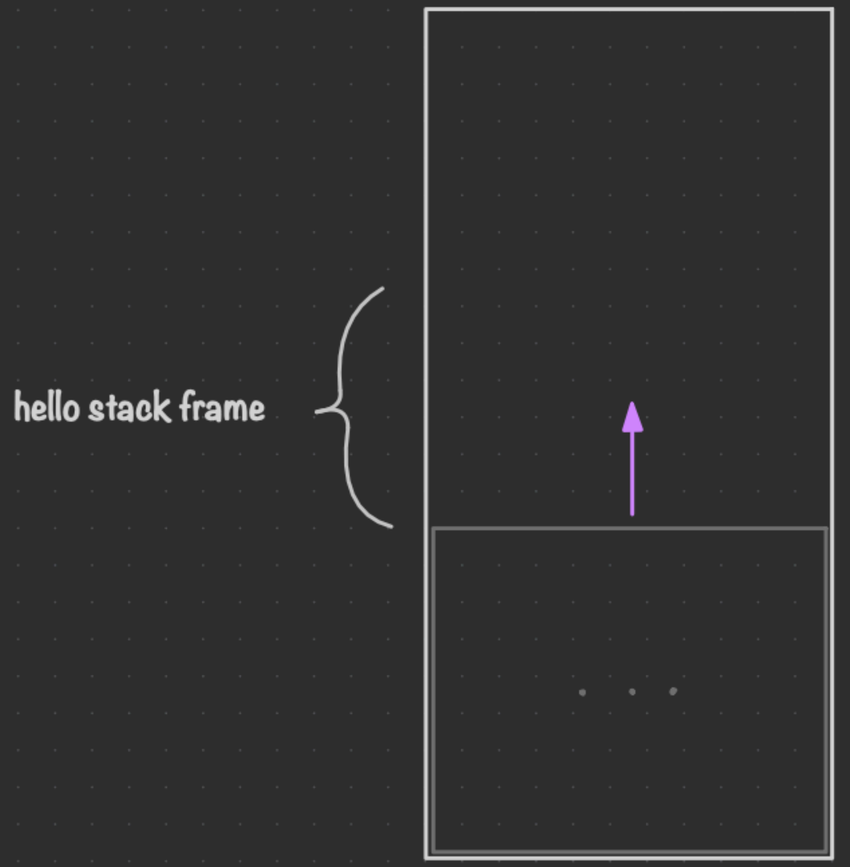
C calling convention



rsp, rbp

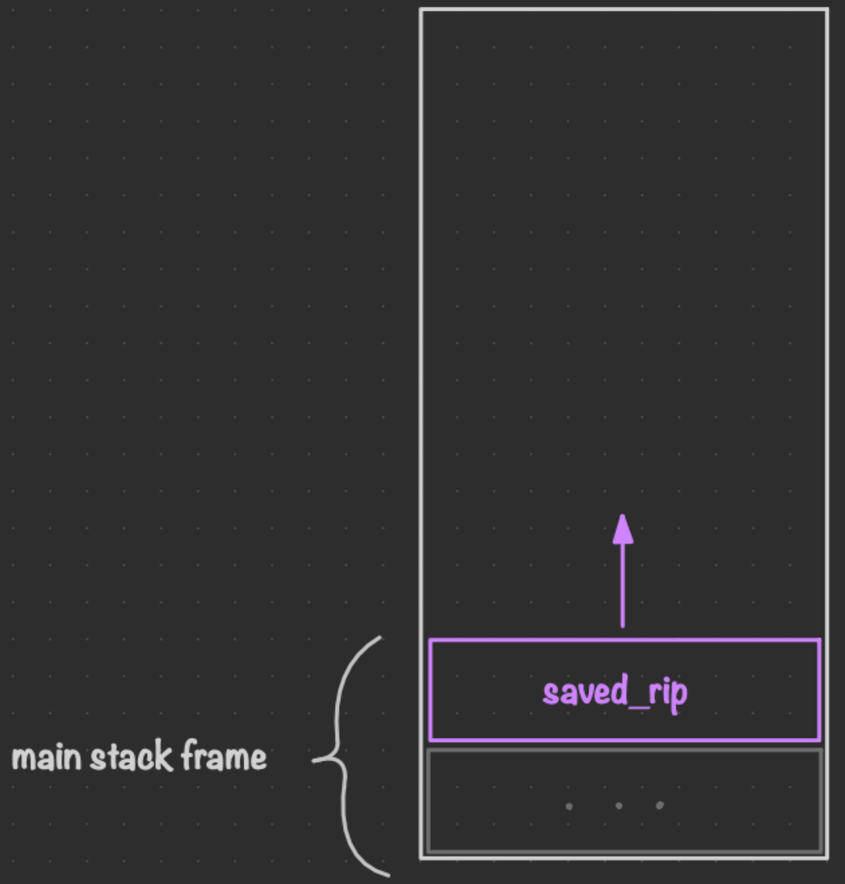
```
main:
    push    $rip
    jmp     <hello_addr>

hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

C calling convention



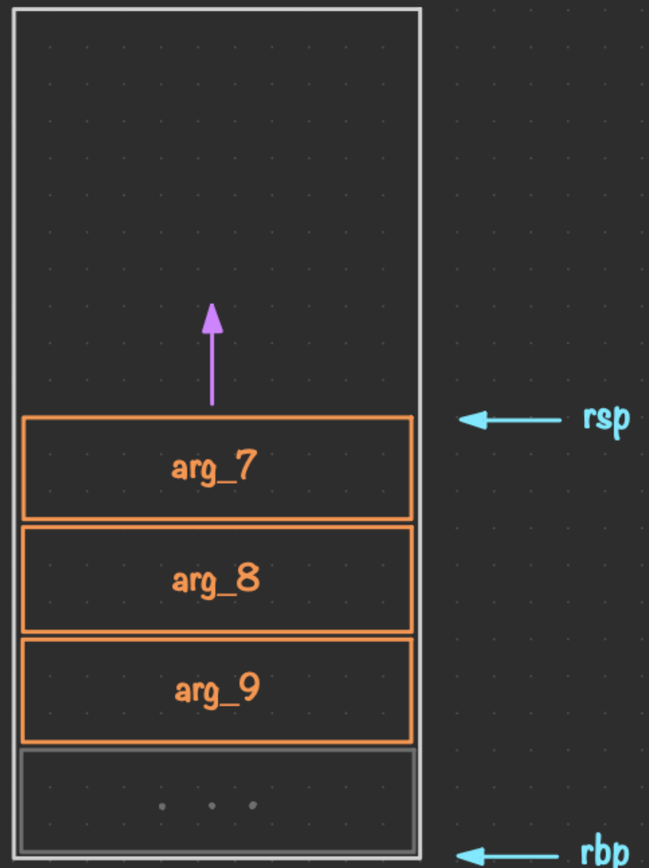

```
main:
    push    $rip
    jmp     <hello_addr>

hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

C calling convention




```
main:
    push    $rip
    jmp     <hello_addr>

hello:
    push    rbp
    mov     rbp,  rsp
    mov     rdi,  <ptr_to_str>
    call    printf
    pop     rbp
    ret
```

C calling convention



rax = ...

rbx = ...

rcx = arg_4

rdx = arg_3

rdi = arg_1

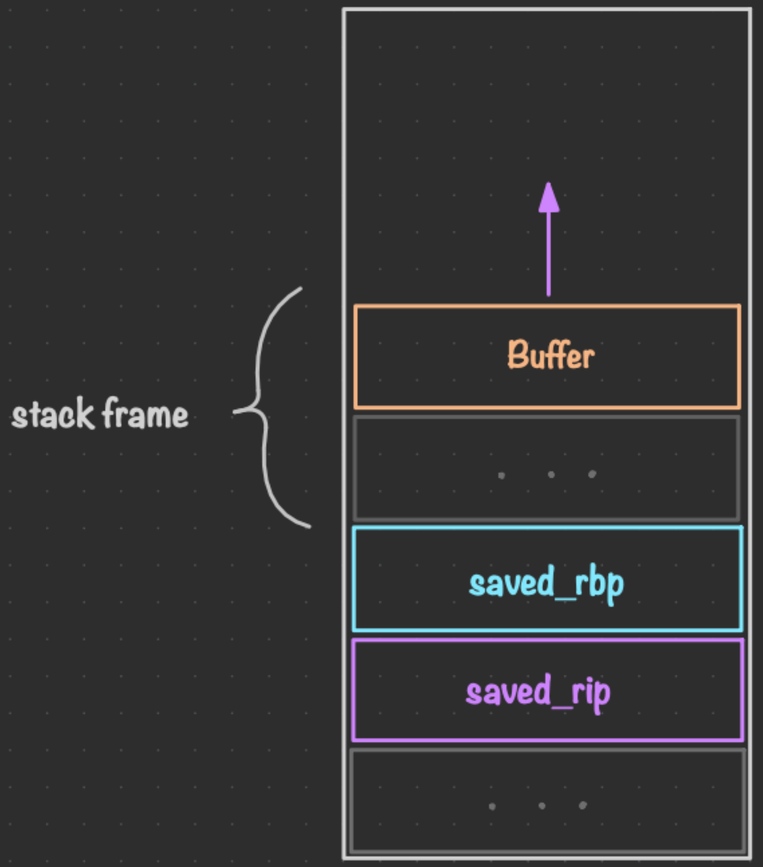
rsi = arg_2

r8 = arg_5

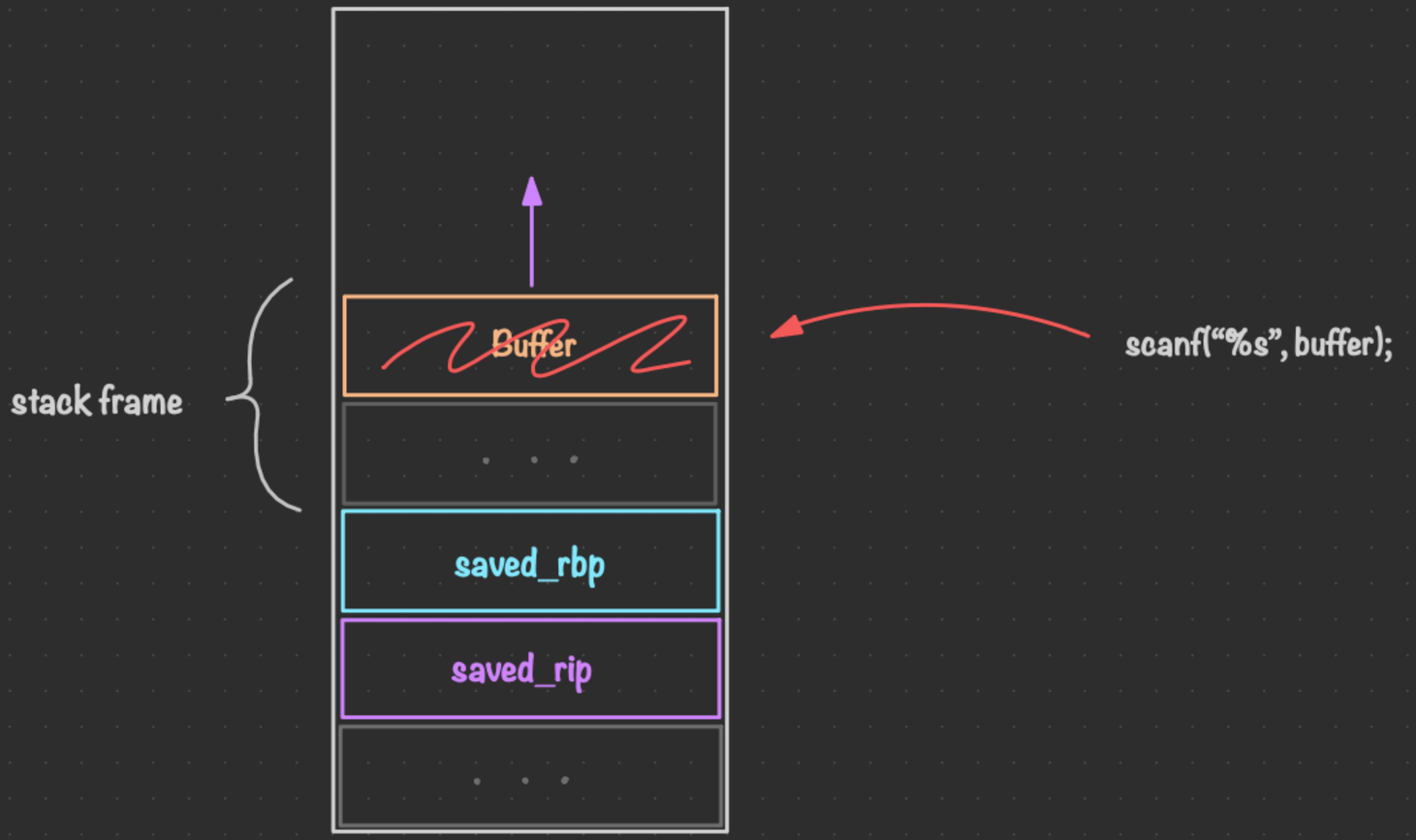
r9 = arg_6

...

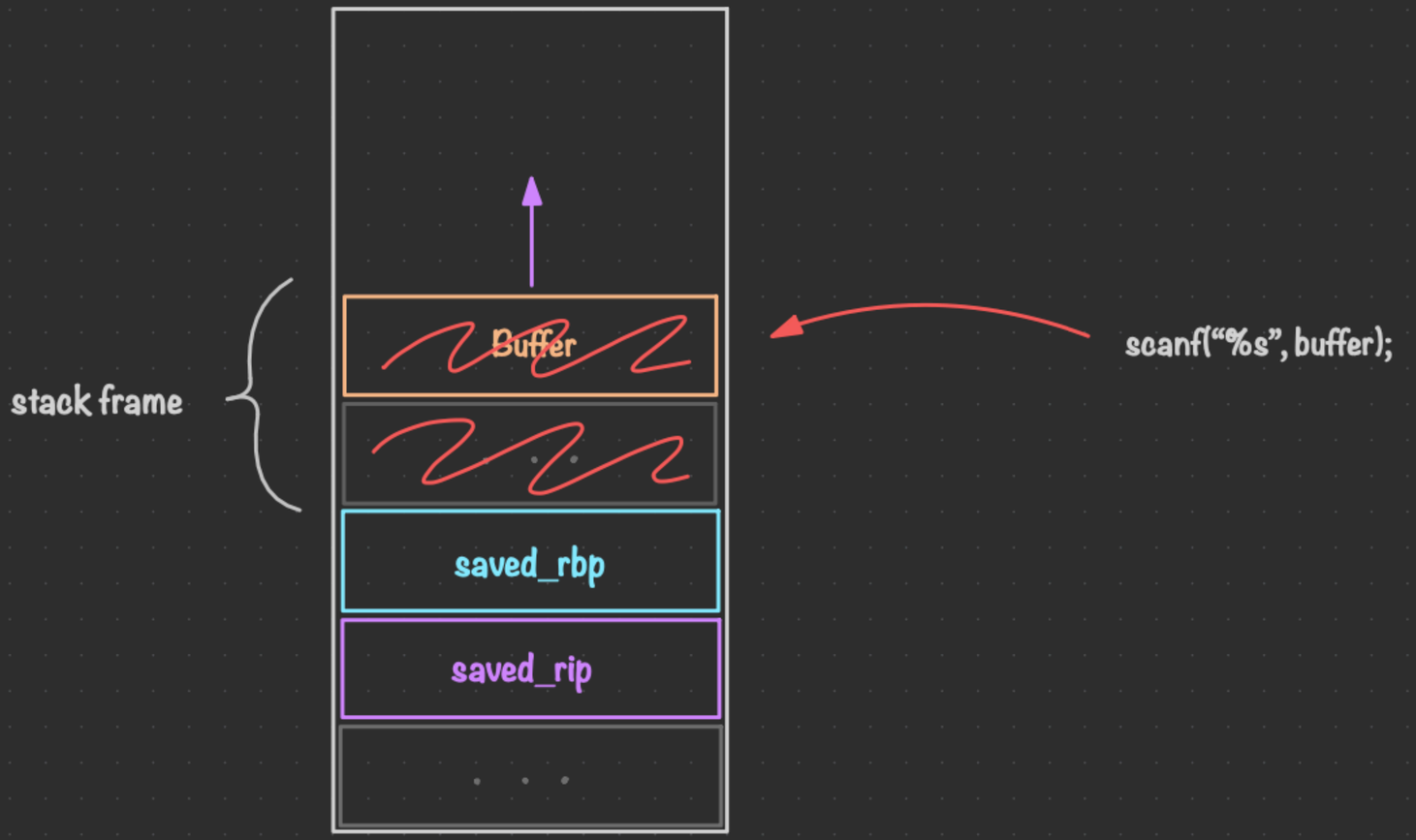
Stack buffer overflow



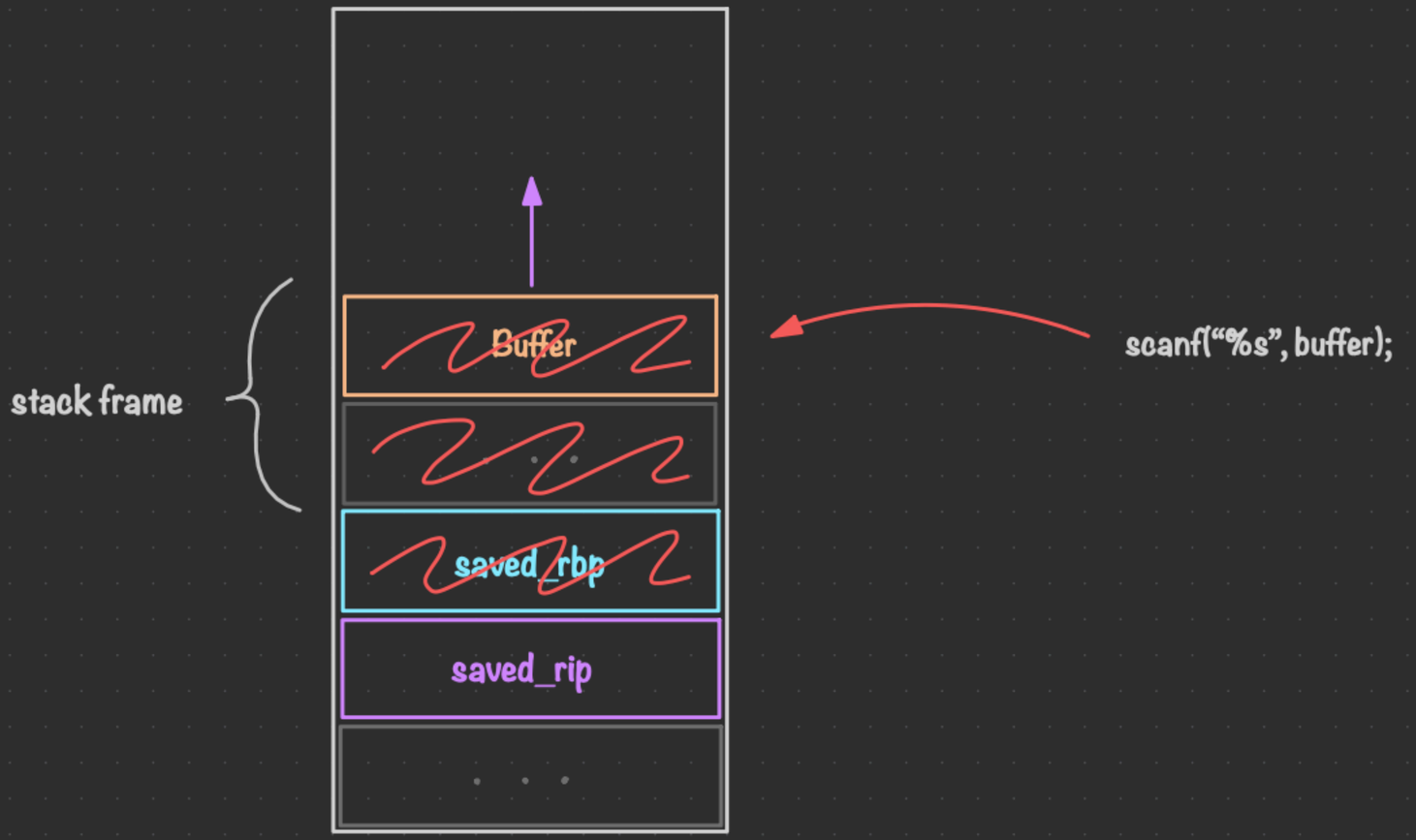
Stack buffer overflow



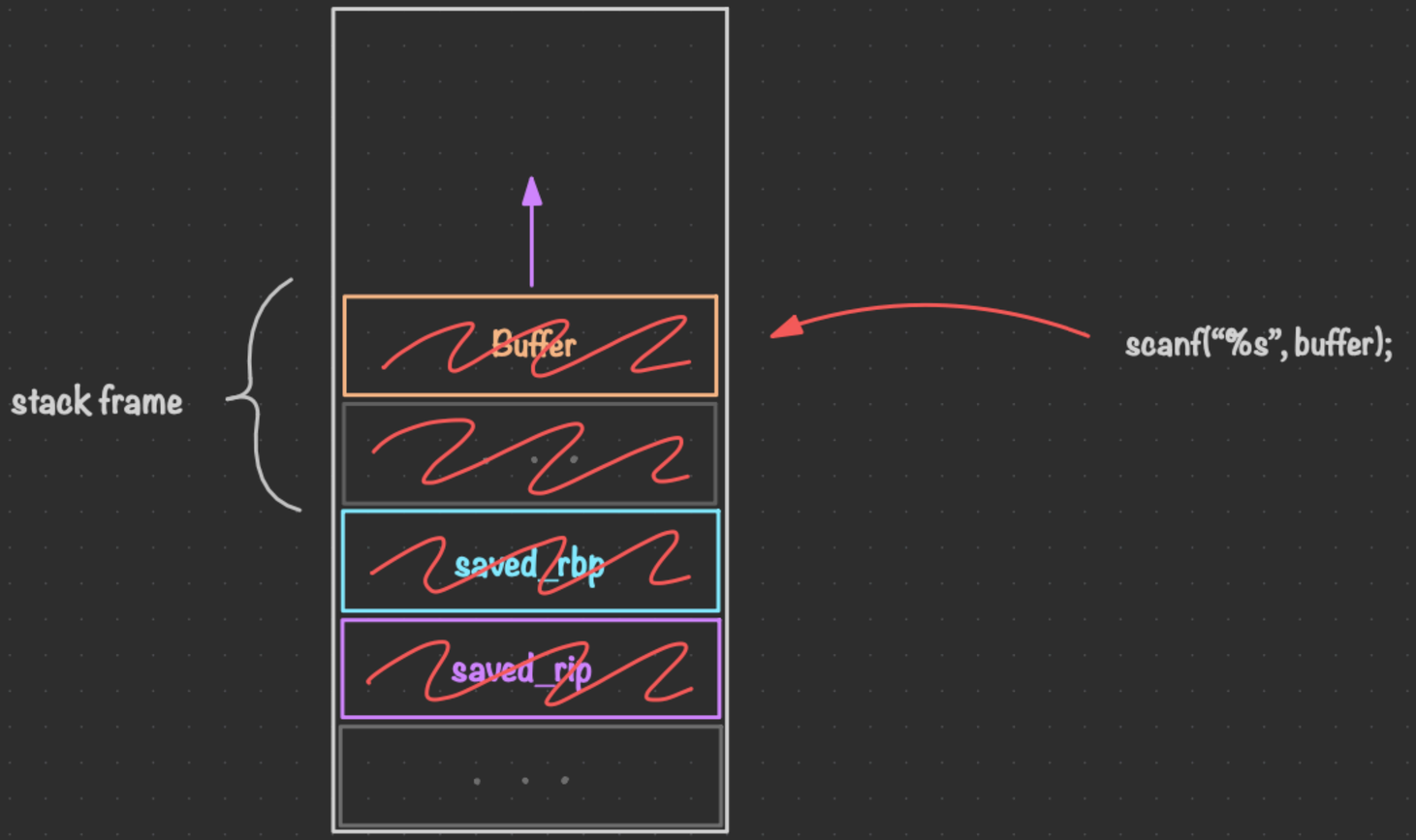
Stack buffer overflow



Stack buffer overflow

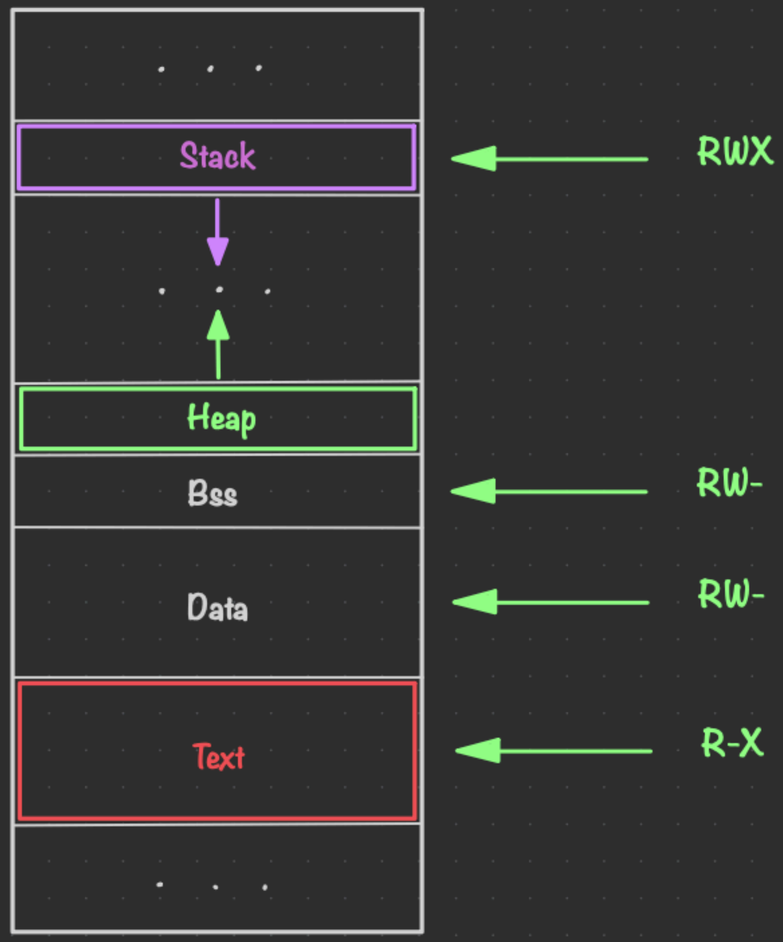


Stack buffer overflow

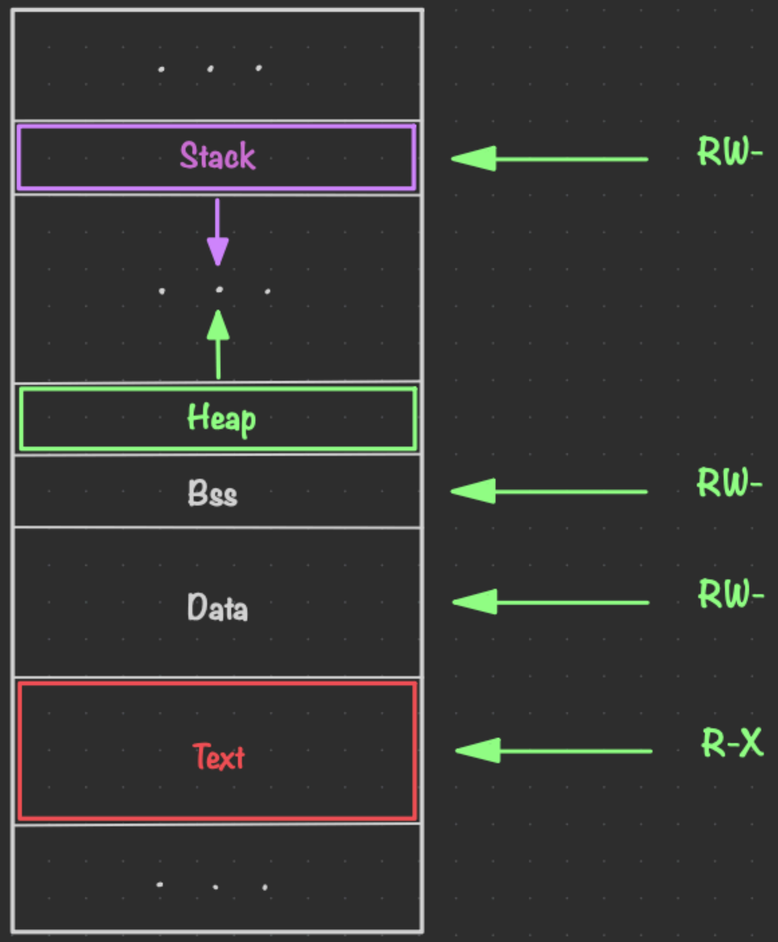


Binary Protections

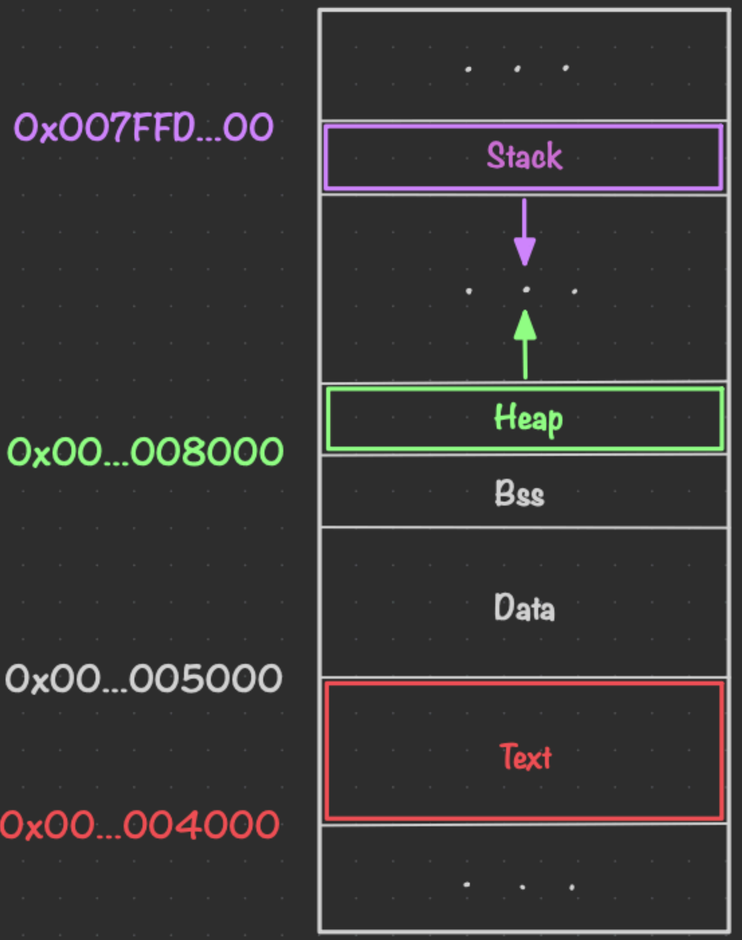
Binary Protections : w/o NX



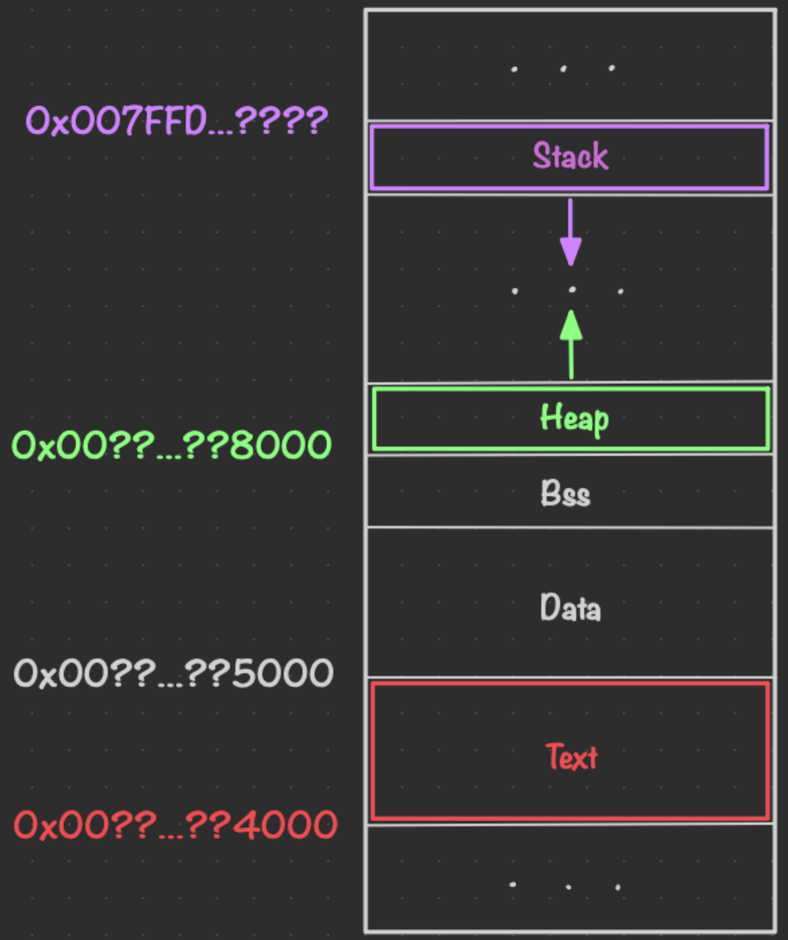
Binary Protections : w/ NX



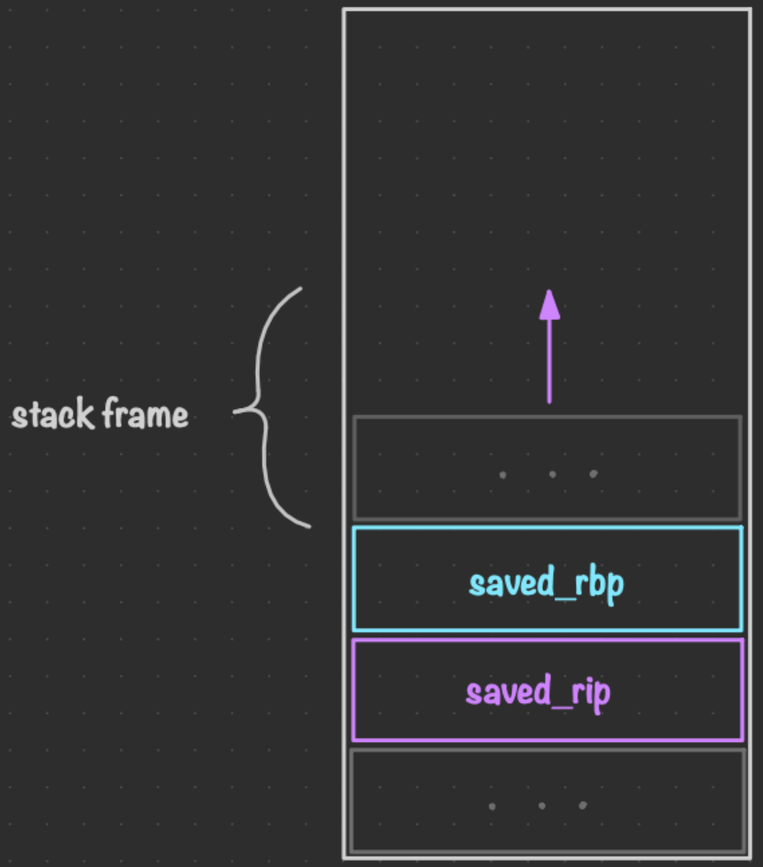
Binary Protections : w/o PIE & ASLR



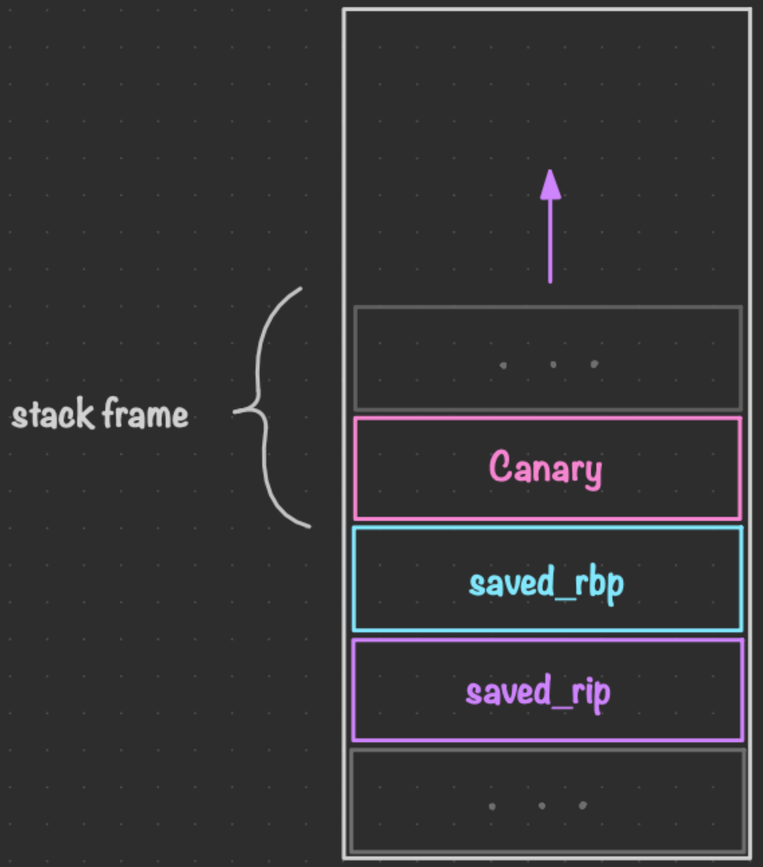
Binary Protections : w/ PIE & ASLR



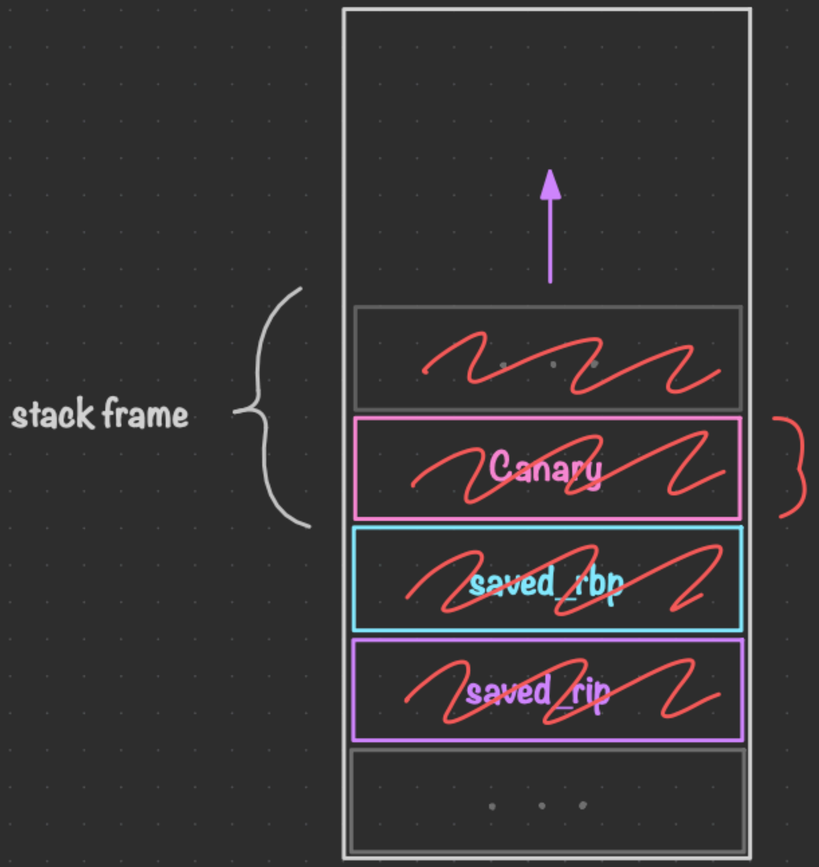
Binary protections : w/o stack canary



Binary protections : w/ stack canary

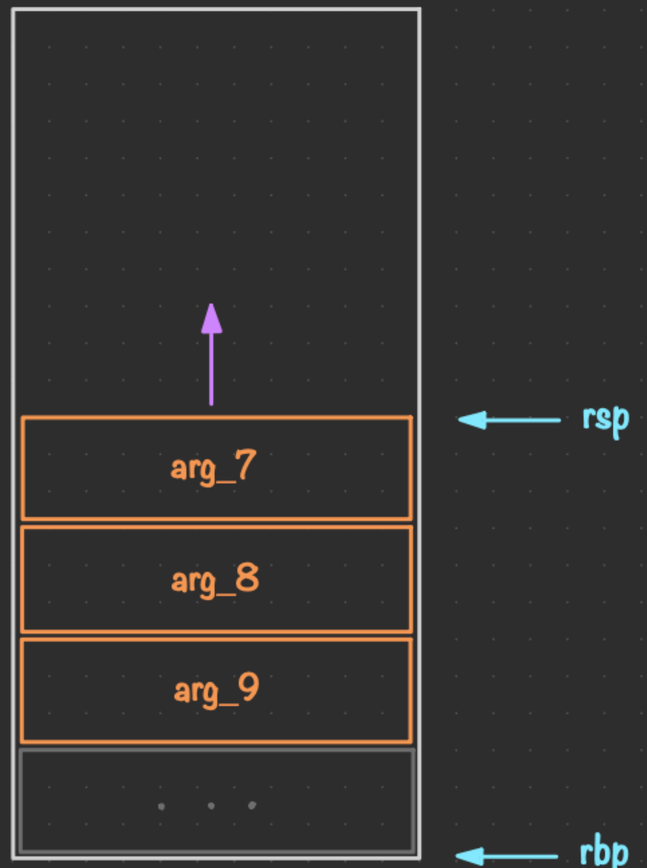


Binary protections : w/ stack canary



Format String

Format string vulnerability



rax = ...

rbx = ...

rcx = arg_4

rdx = arg_3

rdi = arg_1

rsi = arg_2

r8 = arg_5

r9 = arg_6

...

NAME
printf - format and print data

SYNOPSIS
printf FORMAT [ARGUMENT]...
printf OPTION

DESCRIPTION
Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

--help display this help and exit

--version
output version information and exit

FORMAT controls the output as in C printf. Interpreted sequences are:

\" double quote

\\ backslash

\a alert (BEL)

\b backspace

\c produce no further output

\e escape

\f form feed

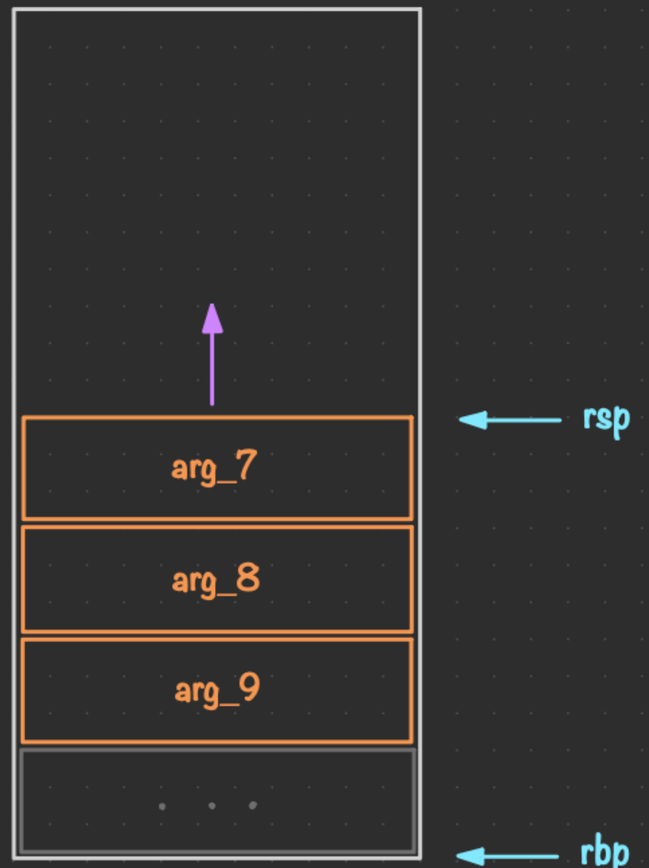
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    char name[4] = "john";

    printf("%d : %x : %p : %c : %s\n", 1234, 0x1337, 0xdeadbeef, 0x41, name);
    return 0;
}
```

NORMAL test.c : 27 Bot 17:1 23:28

```
❯ ./test
1234 : 1337 : 0xdeadbeef : A : john
❯
❯
❯
```

~/Secu/Cours_pwn 11:27:34

Format string vulnerability



rax = ...

rbx = ...

rcx = arg_4

rdx = arg_3

rdi = arg_1

rsi = arg_2

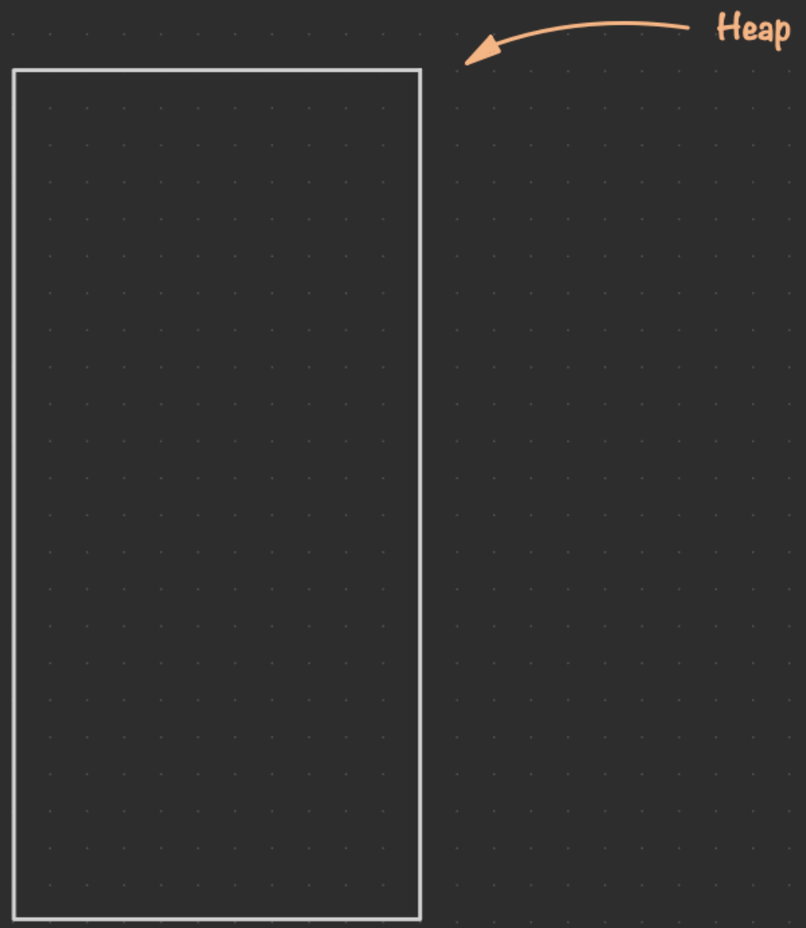
r8 = arg_5

r9 = arg_6

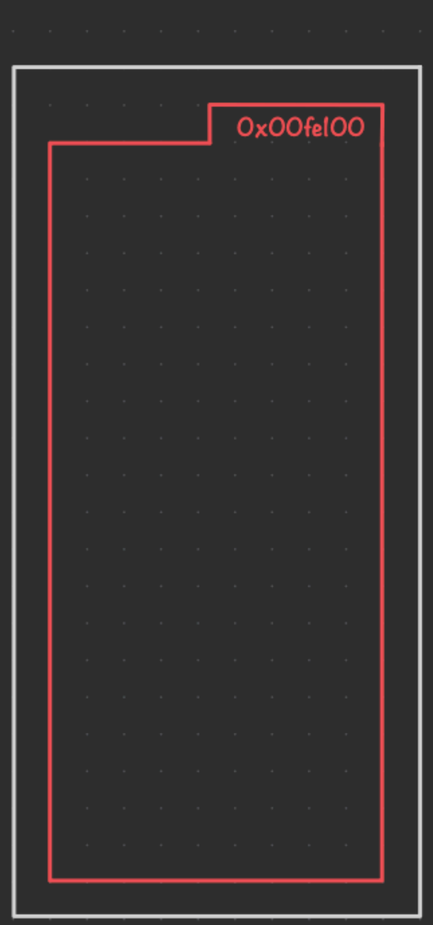
...

Heap Vulnerabilities

Heap overflow

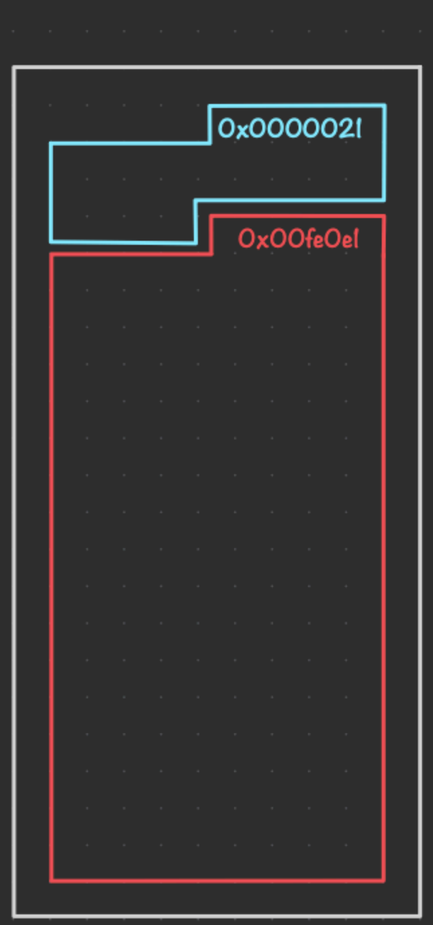


Malloc internals



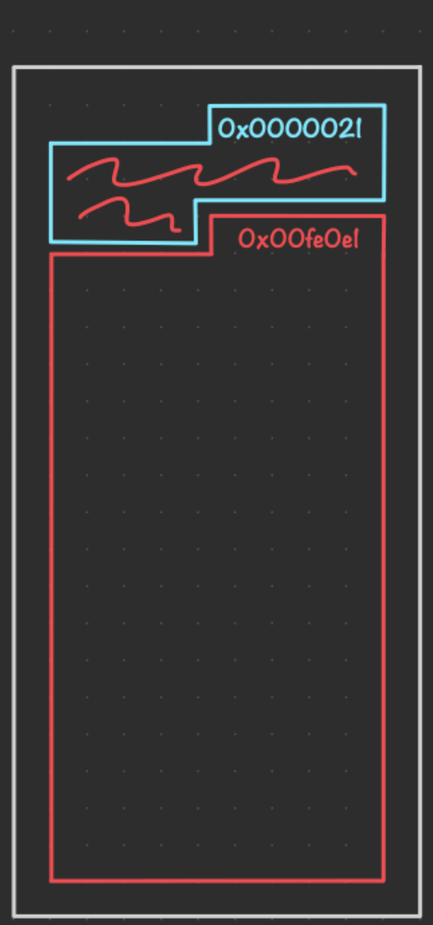

```
char buffer[10] = malloc(10);
puts("What's you name?");
scanf("%s", buffer);
free(buffer);
```

Malloc internals



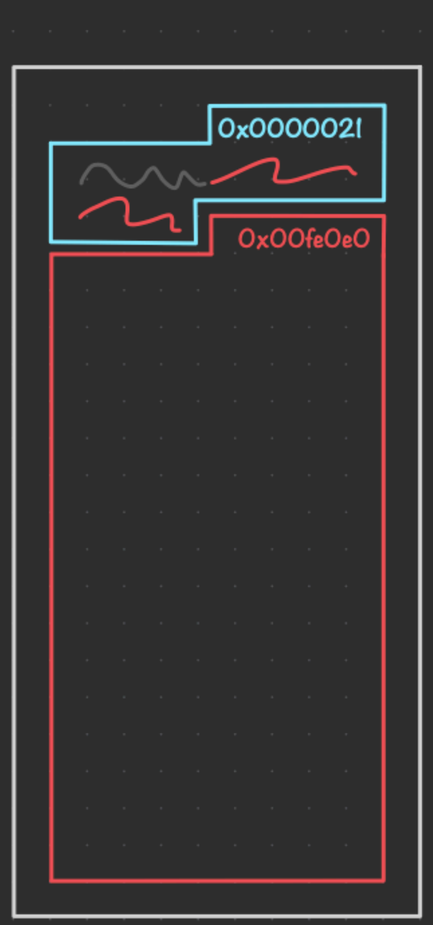

```
char buffer[10] = malloc(10);
puts("What's you name?");
scanf("%s", buffer);
free(buffer);
```

Malloc internals



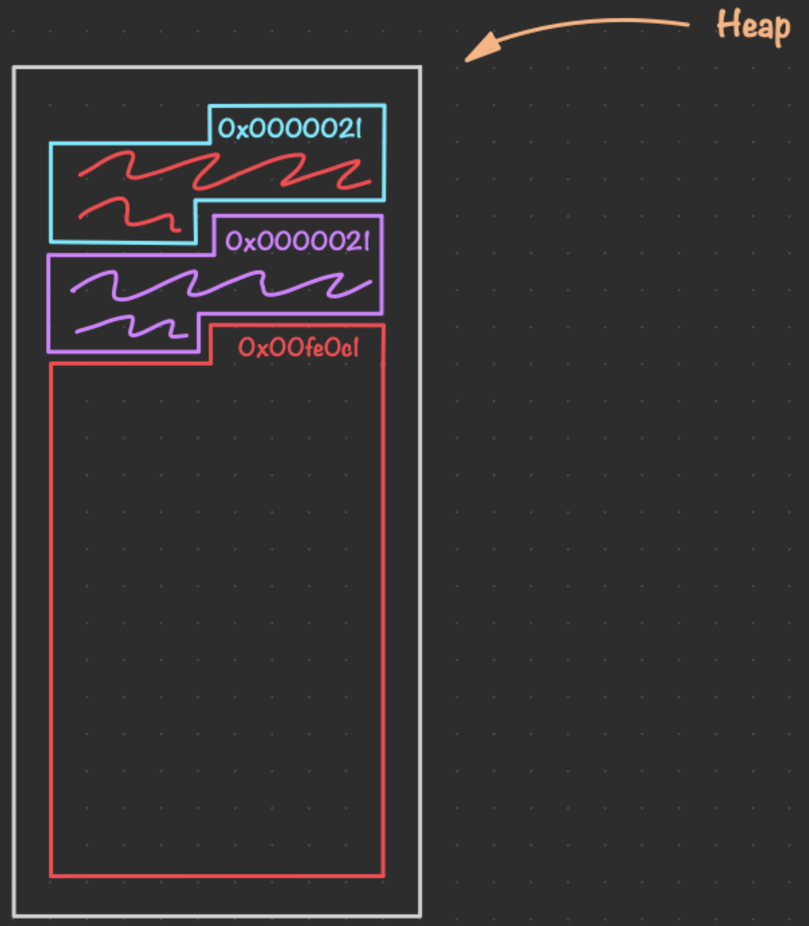

```
char buffer[10] = malloc(10);
puts("What's you name?");
scanf("%s", buffer);
free(buffer);
```

Malloc internals

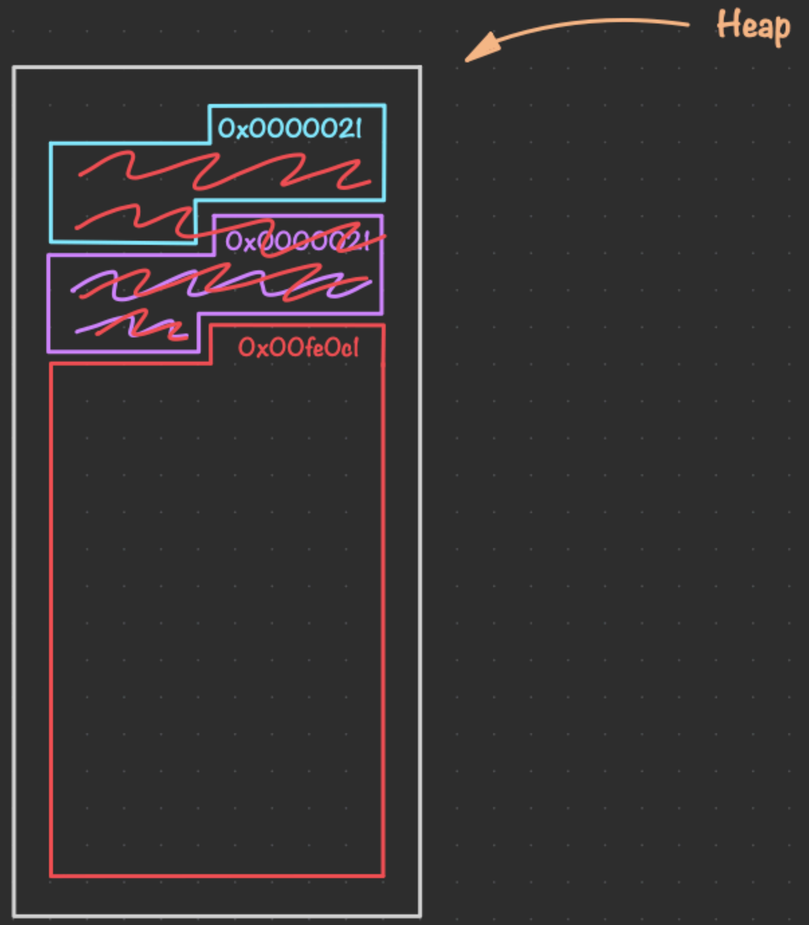



```
char buffer[10] = malloc(10);
puts("What's you name?");
scanf("%s", buffer);
free(buffer);
```

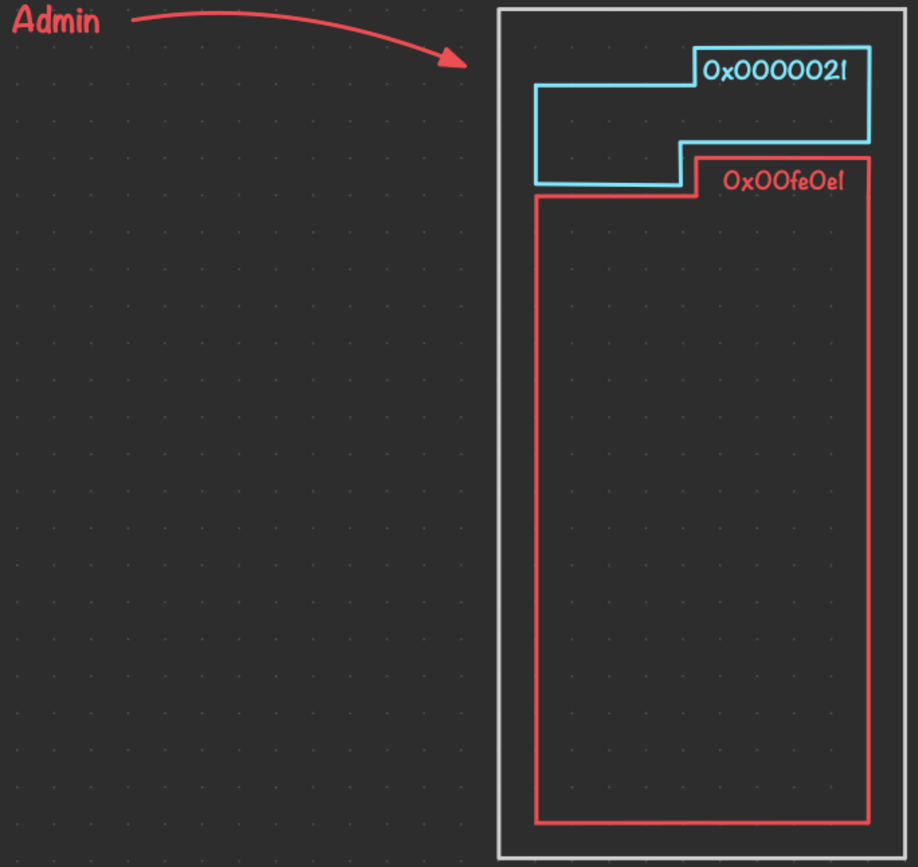
Heap overflow



Heap overflow

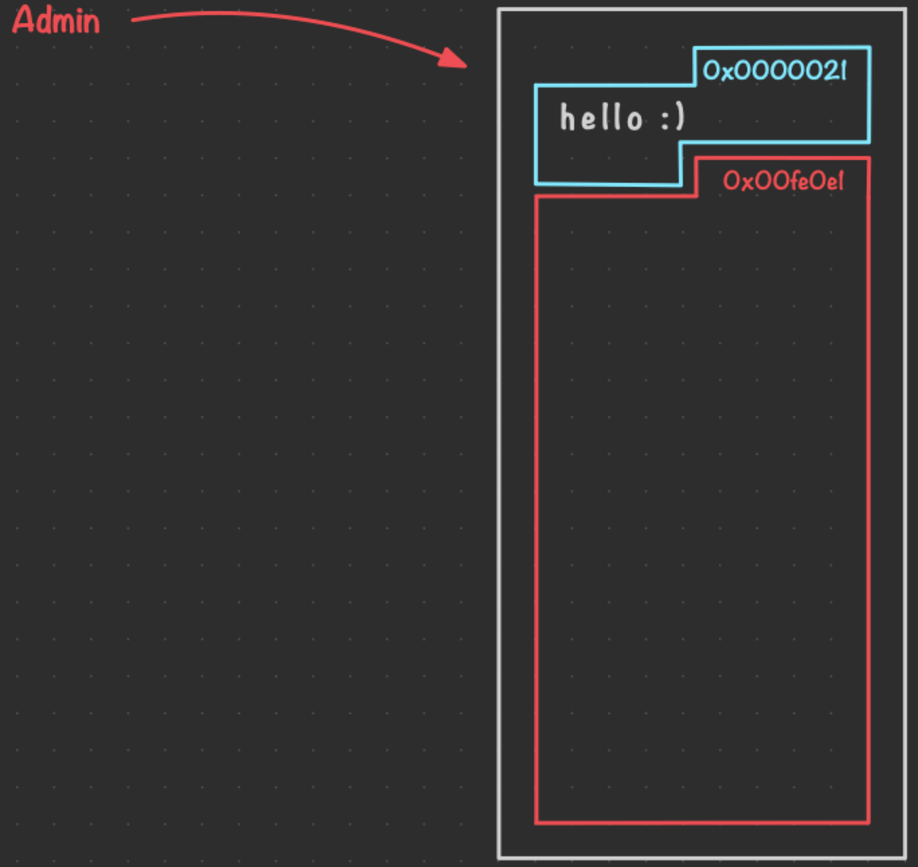


Use After Free



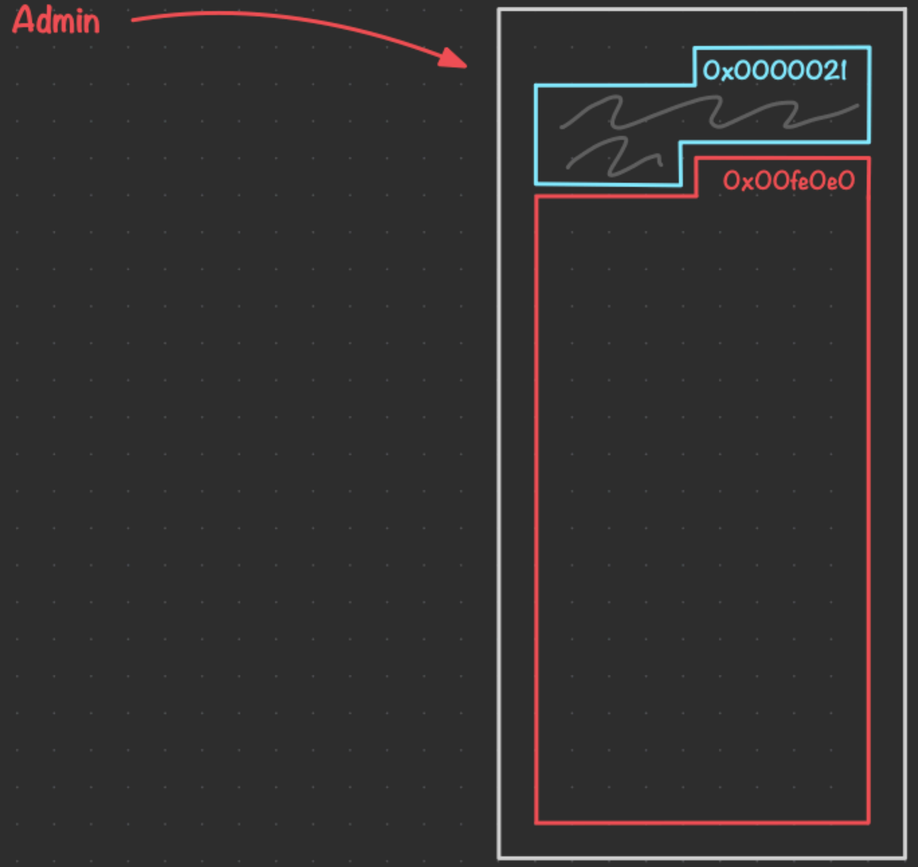

```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
  ...
free(admin);
  ...
char user[10] = malloc(10);
scanf("%9s", user);
  ...
printf(admin);
```

Use After Free



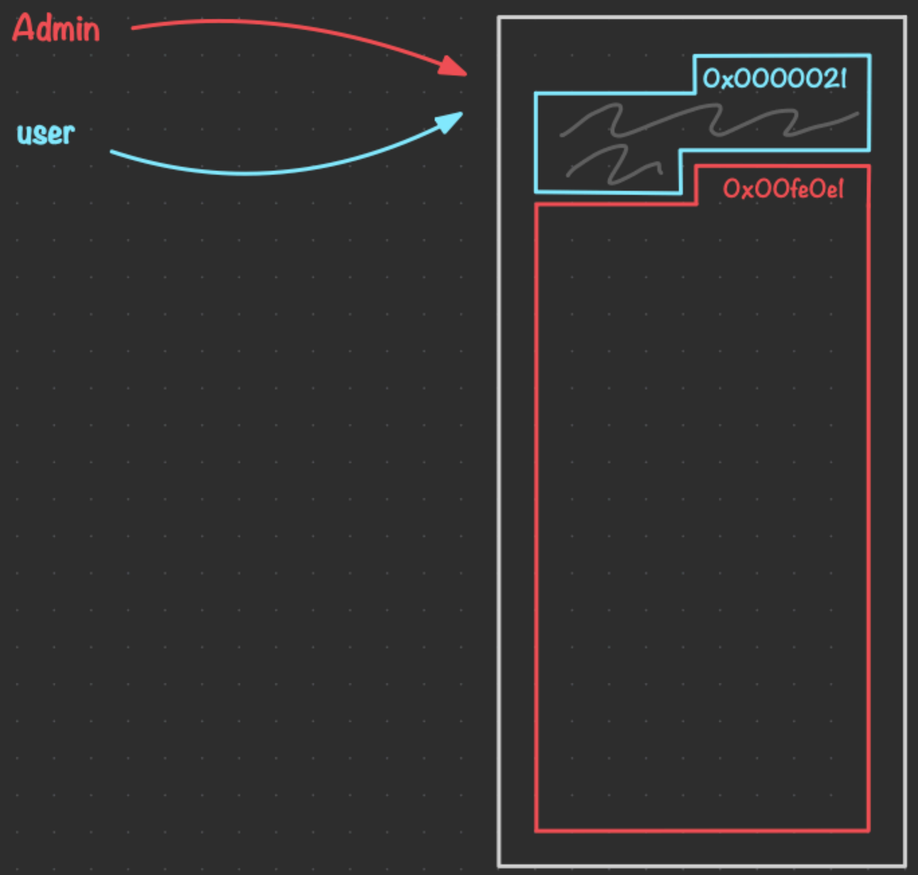

```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
  ...
free(admin);
  ...
char user[10] = malloc(10);
scanf("%9s", user);
  ...
printf(admin);
```

Use After Free



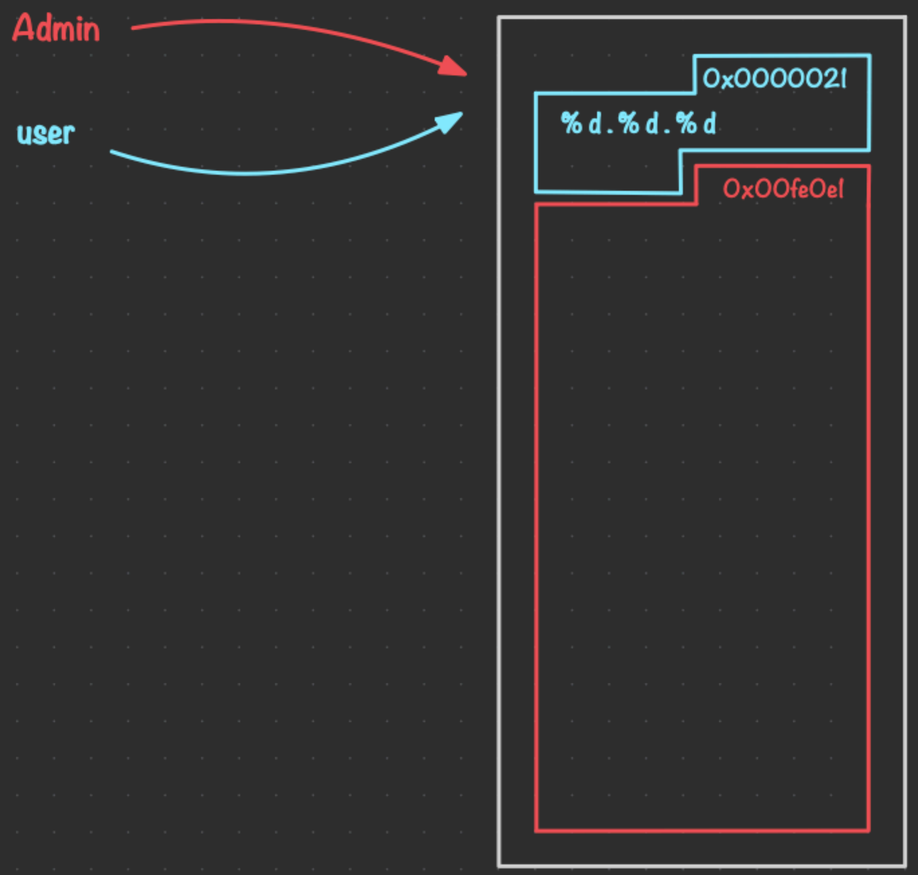

```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
...
free(admin);
...
char user[10] = malloc(10);
scanf("%9s", user);
...
printf(admin);
```

Use After Free



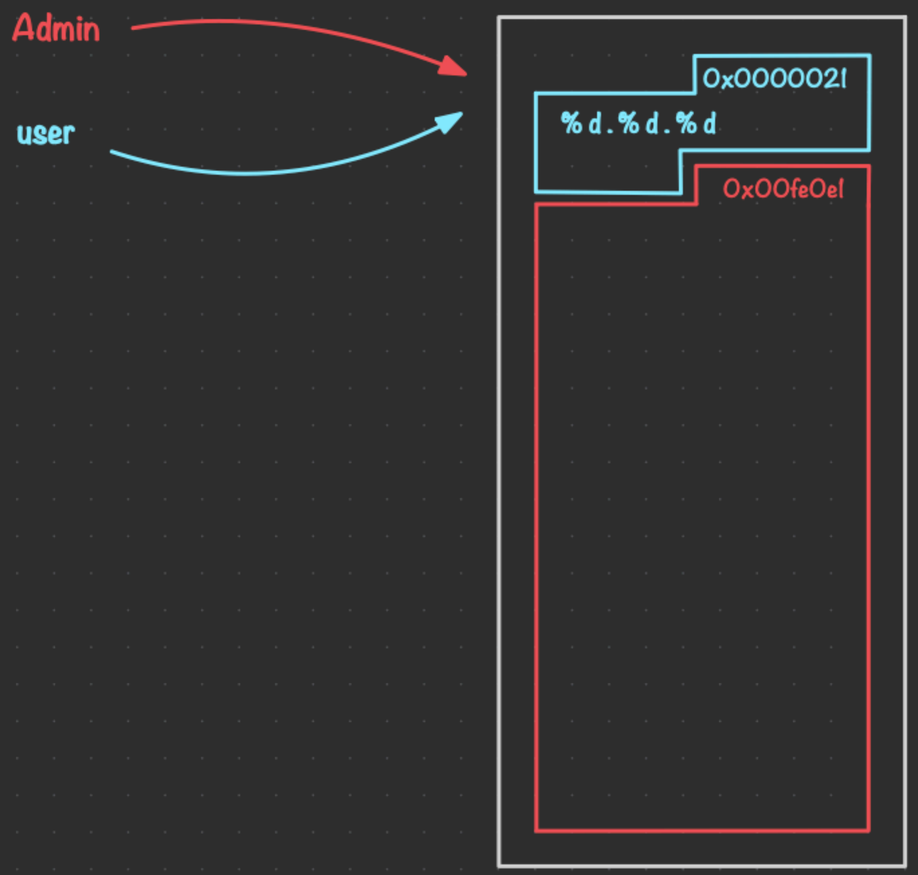

```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
  ...
free(admin);
  ...
char user[10] = malloc(10);
scanf("%9s", user);
  ...
printf(admin);
```

Use After Free




```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
  ...
free(admin);
  ...
char user[10] = malloc(10);
scanf("%9s", user);
  ...
printf(admin);
```

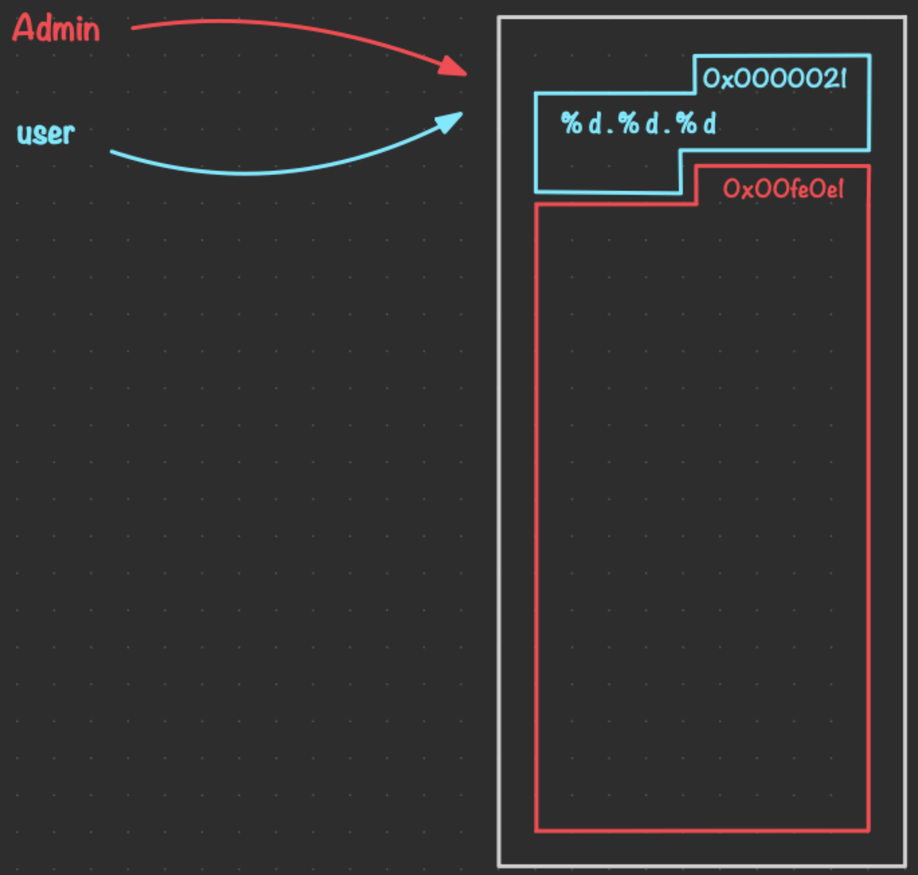
Use After Free




```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
  ...
free(admin);
  ...
char user[10] = malloc(10);
scanf("%9s", user);
  ...
printf(admin);
```

hello :)

Use After Free




```
char admin[10] = malloc(10);
strcpy(admin, "hello :)");
  ...
free(admin);
  ...
char user[10] = malloc(10);
scanf("%9s", user);
  ...
printf(admin);
```

leak !!