

U23 - Binary Exploitation

Stratum Auhuur

robbje@aachen.ccc.de

November 21, 2016

Context

- ▶ OS: Linux

Context

- ▶ OS: Linux
- ▶ CPU: x86 (32 bit)

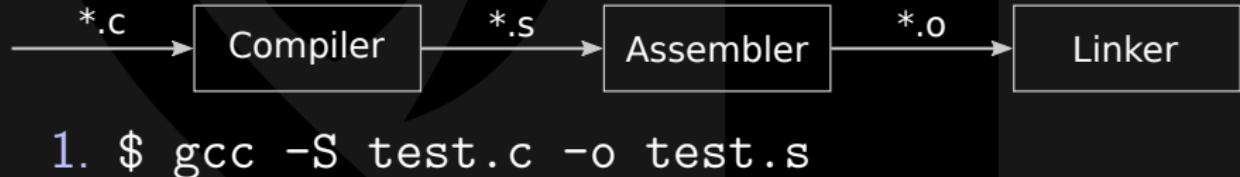
Context

- ▶ OS: Linux
- ▶ CPU: x86 (32 bit)
- ▶ Address Space Layout Randomization: off
- ▶ No eXecution: off
- ▶ Stack cookies: off

From source code to binary

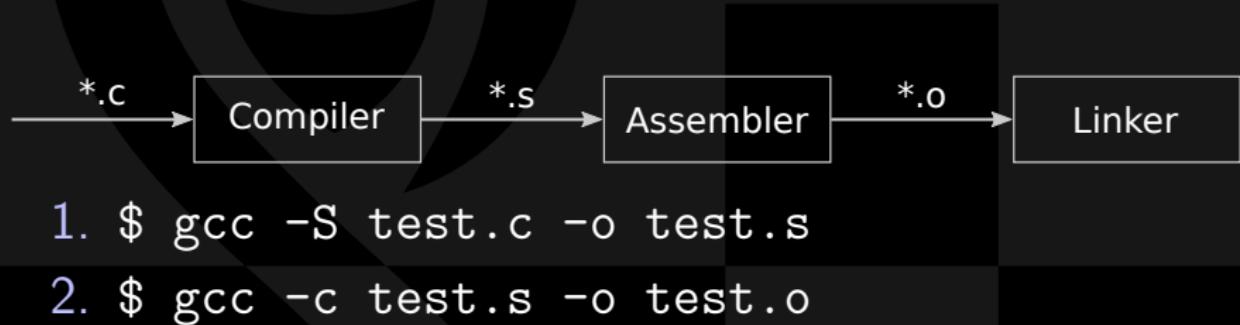


From source code to binary



```
1. $ gcc -S test.c -o test.s
```

From source code to binary



From source code to binary



1. \$ gcc -S test.c -o test.s
2. \$ gcc -c test.s -o test.o
3. \$ gcc test.o -o test

From source code to binary



1. \$ gcc -S test.c -o test.s
2. \$ gcc -c test.s -o test.o
3. \$ gcc test.o -o test

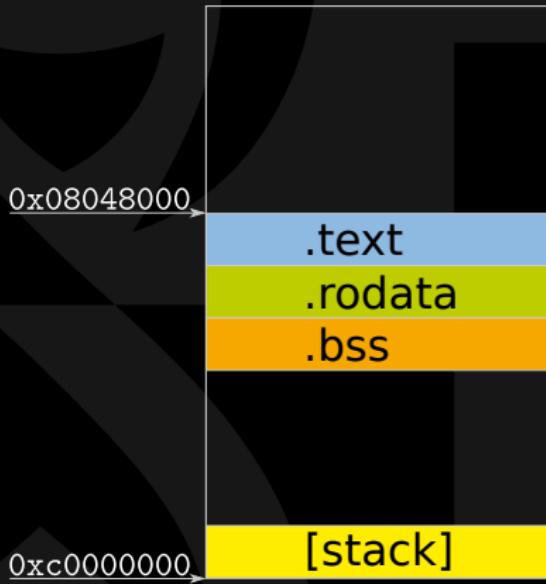
(There's also a preprocessor stage...)

Memory layout: Flat memory model

- ▶ Userspace (3GB):
0x00000000 \Rightarrow 0xbfffffff
- ▶ Kernelspace (1GB):
0xc0000000 \Rightarrow 0xffffffff
- ▶ Every program has this address space available

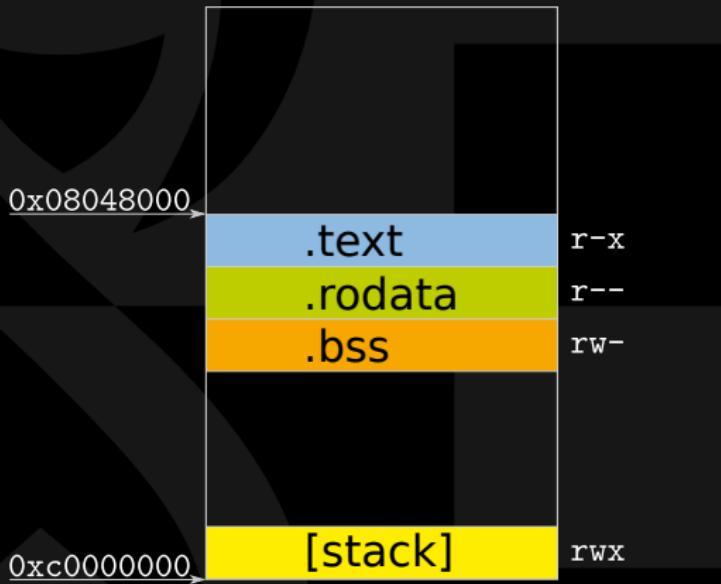


A binary is loaded into memory...



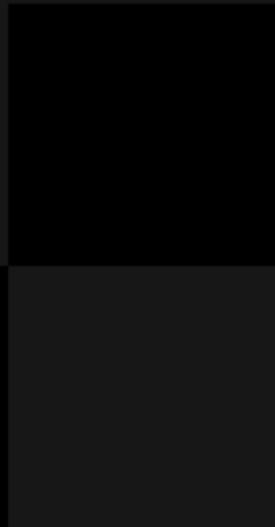
- ▶ Stack grows to low addresses!

A binary is loaded into memory...

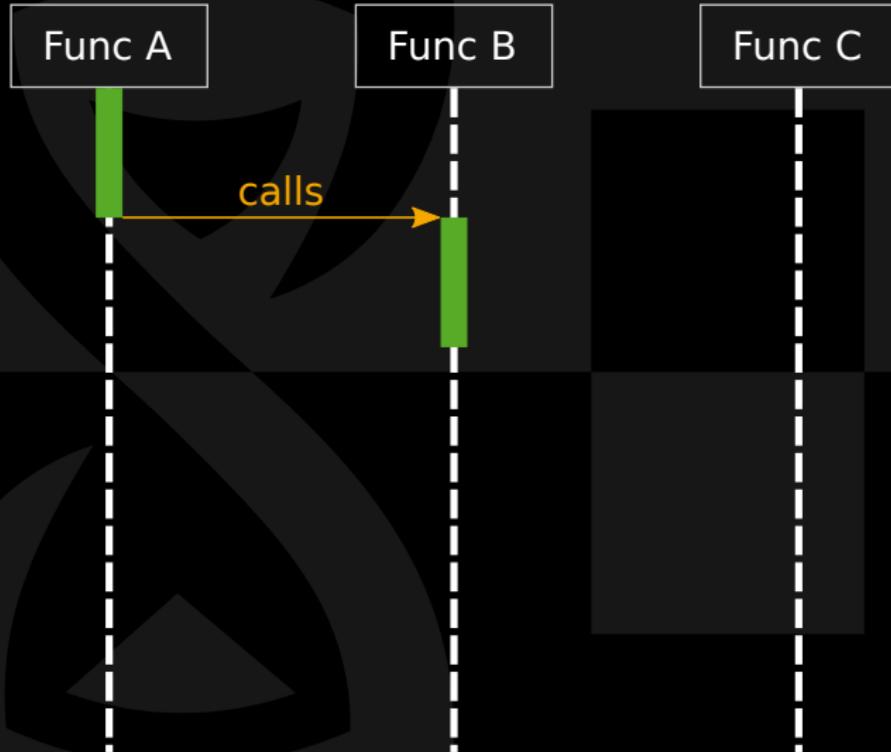


- ▶ Stack grows to low addresses!

Demo of readelf

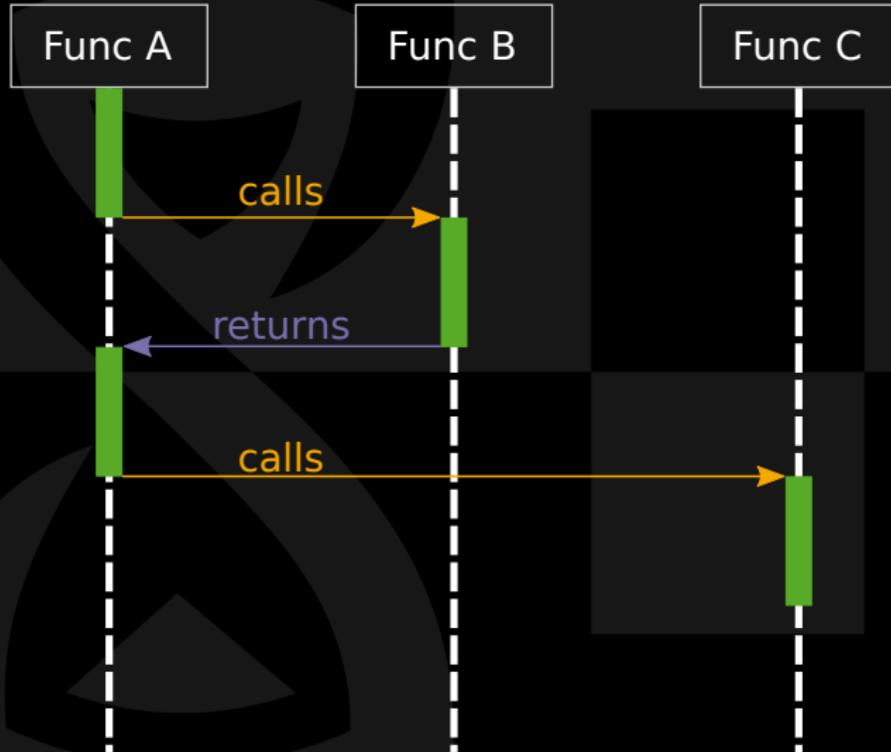


Functions



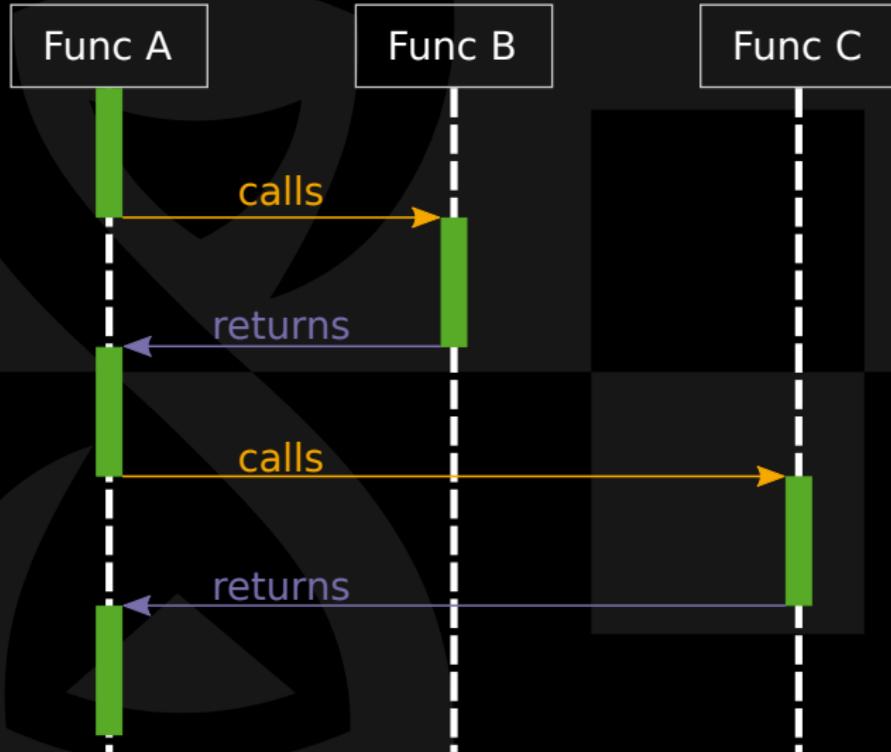
- ▶ We can call functions and they return

Functions



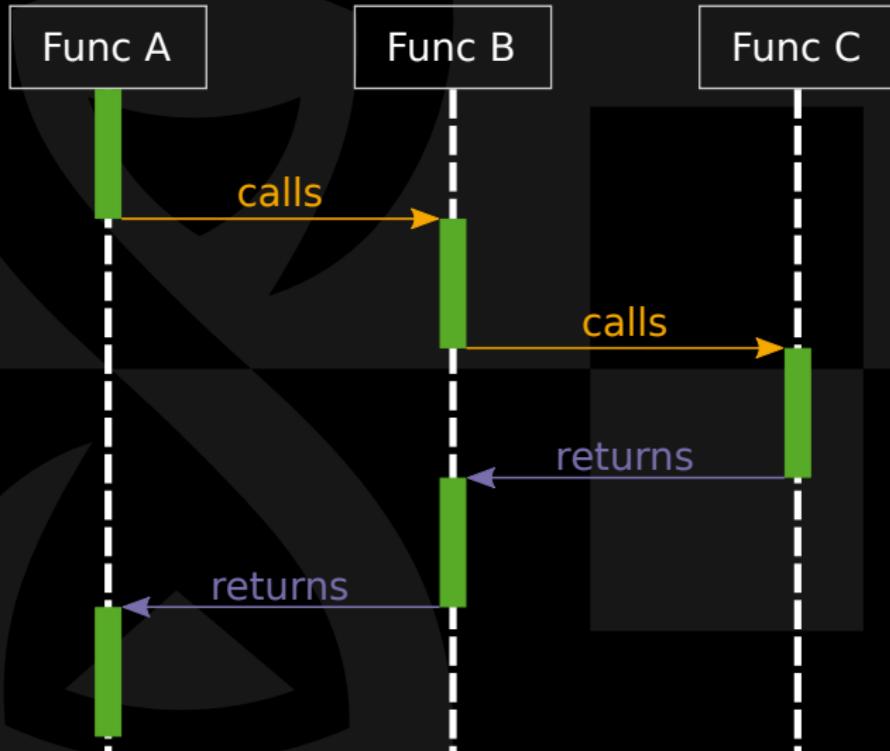
- ▶ We can call functions and they return

Functions

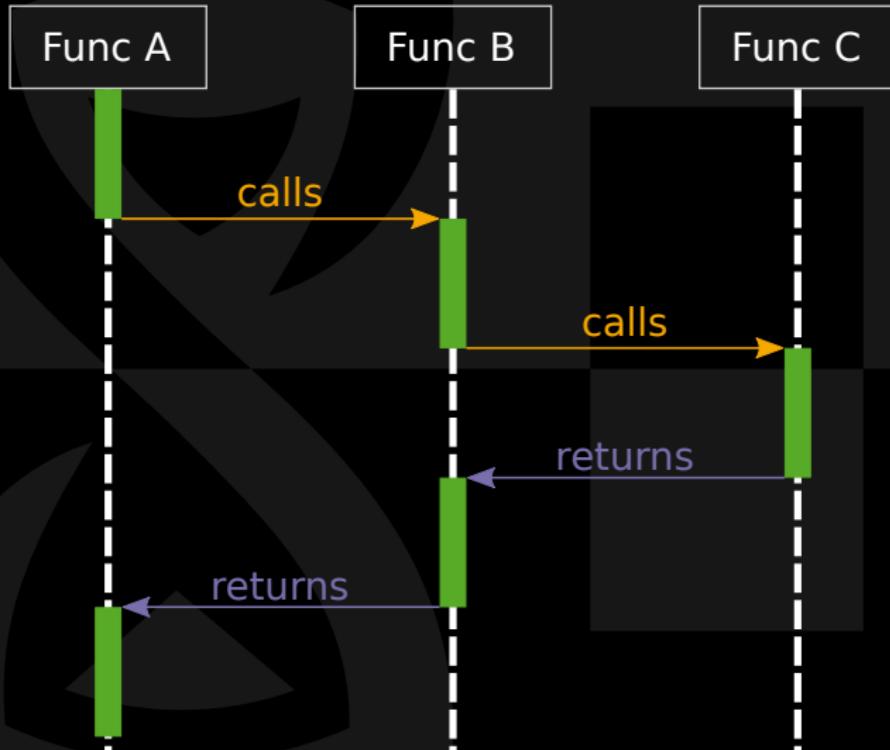


- ▶ We can call functions and they return

Functions 2

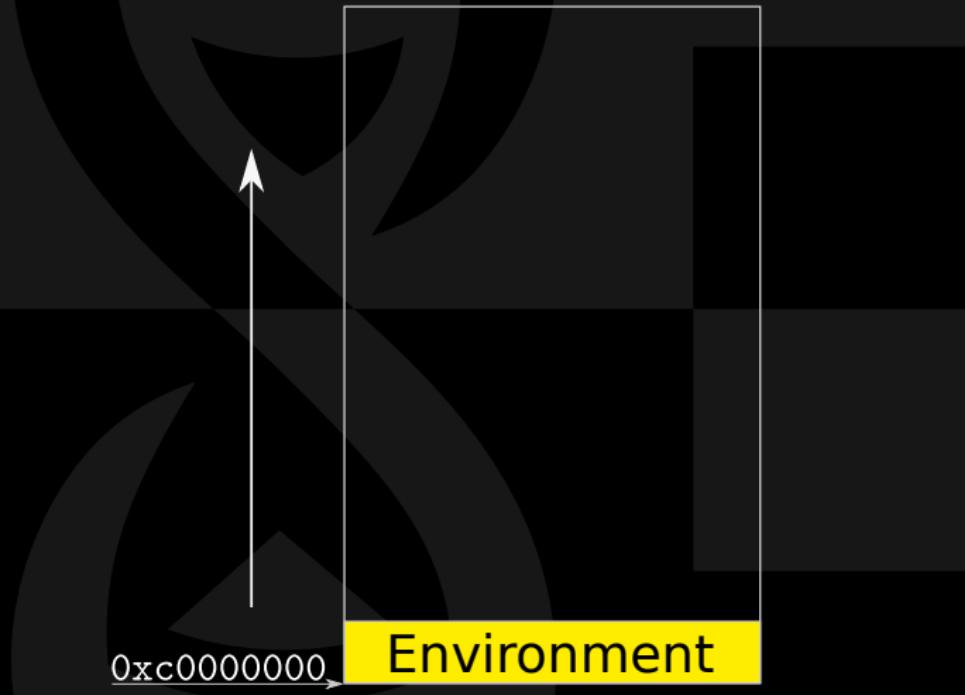


Functions 2



- ▶ How does the CPU know where to return to?

The Stack layout



The Stack layout

A diagram illustrating the Stack layout. At the bottom left, the memory address `0xc0000000` is shown with an arrow pointing to the right. Above this, a vertical white line with an upward-pointing arrow indicates the direction of stack growth. To the right of the address, a yellow box contains two labels: `C cmdline args` at the top and `Environment` below it. The background features a dark gray circular pattern.

0xc0000000

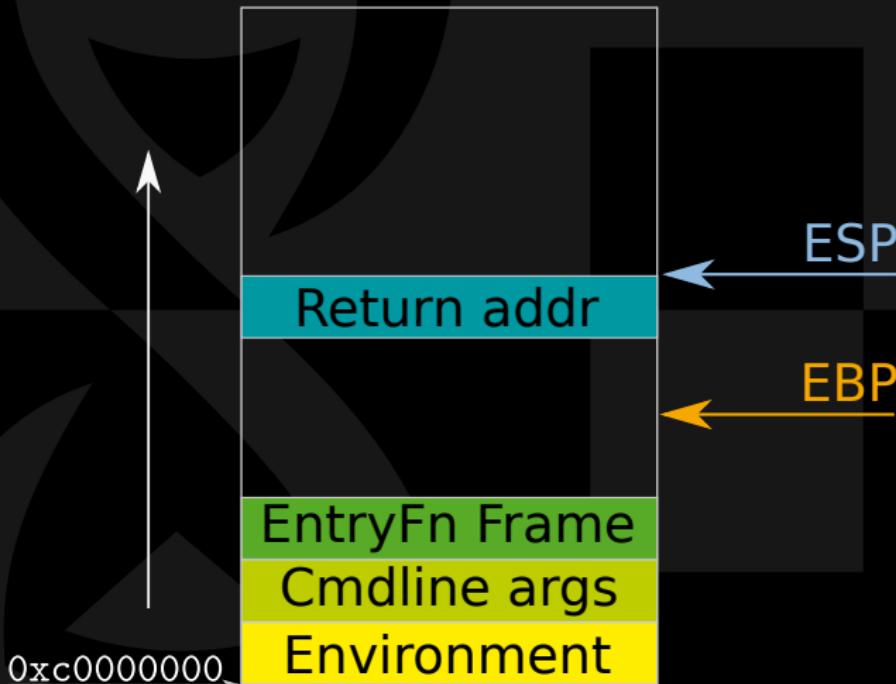
C cmdline args

Environment

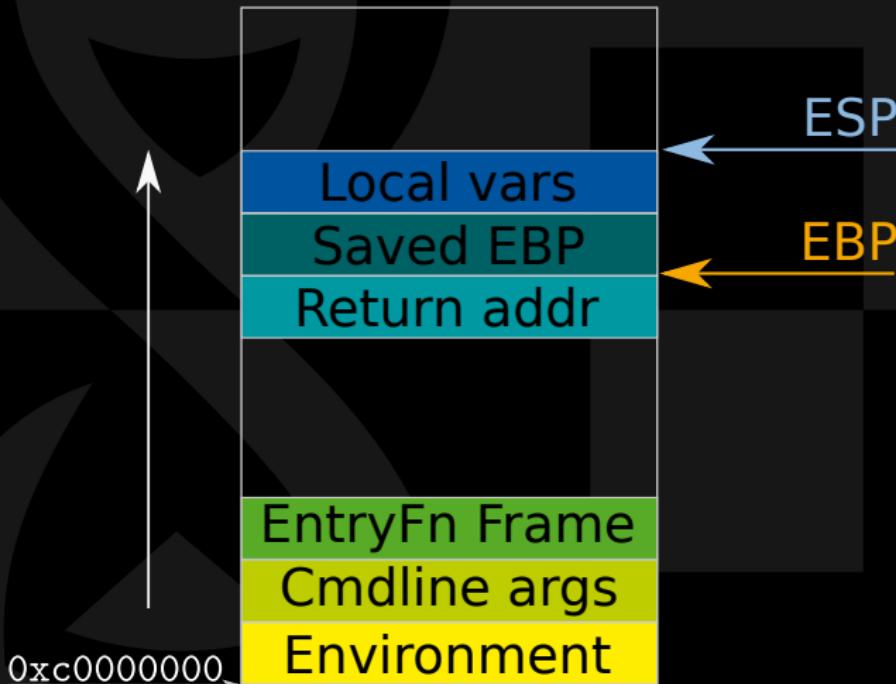
The Stack layout



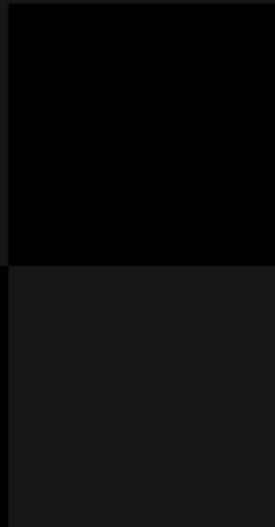
The Stack layout



The Stack layout



Demo of gdb



CPU Registers: dumbed down

- ▶ Stack related: ESP, EBP

CPU Registers: dumbed down

- ▶ Stack related: ESP, EBP
- ▶ Instruction Pointer: EIP

CPU Registers: dumbed down

- ▶ Stack related: ESP, EBP
- ▶ Instruction Pointer: EIP
- ▶ More or less multi-purpose: EAX, EBX, ECX, EDX, ESI, EDI

CPU Registers: dumbed down

- ▶ Stack related: ESP, EBP
- ▶ Instruction Pointer: EIP
- ▶ More or less multi-purpose: EAX, EBX, ECX, EDX, ESI, EDI

EAX

CPU Registers: dumbed down

- ▶ Stack related: ESP, EBP
- ▶ Instruction Pointer: EIP
- ▶ More or less multi-purpose: EAX, EBX, ECX, EDX, ESI, EDI



CPU Registers: dumbed down

- ▶ Stack related: ESP, EBP
- ▶ Instruction Pointer: EIP
- ▶ More or less multi-purpose: EAX, EBX, ECX, EDX, ESI, EDI

EAX	
	AX
AH	AL

String and memory copy operations

- ▶ `void *memcpy(void *dest, const void *src, size_t n);`

String and memory copy operations

- ▶ `void *memcpy(void *dest, const void *src, size_t n);`
- ▶ `char *strcpy(char *dest, const char *src);`

String and memory copy operations

- ▶ `void *memcpy(void *dest, const void *src, size_t n);`
- ▶ `char *strcpy(char *dest, const char *src);`
- ▶ `char *strncpy(char *dest, const char *src, size_t n);`

String and memory copy operations

- ▶ `void *memcpy(void *dest, const void *src, size_t n);`
- ▶ `char *strcpy(char *dest, const char *src);`
- ▶ `char *strncpy(char *dest, const char *src, size_t n);`
- ▶ `memmove, bcopy, memccpy, etc...`

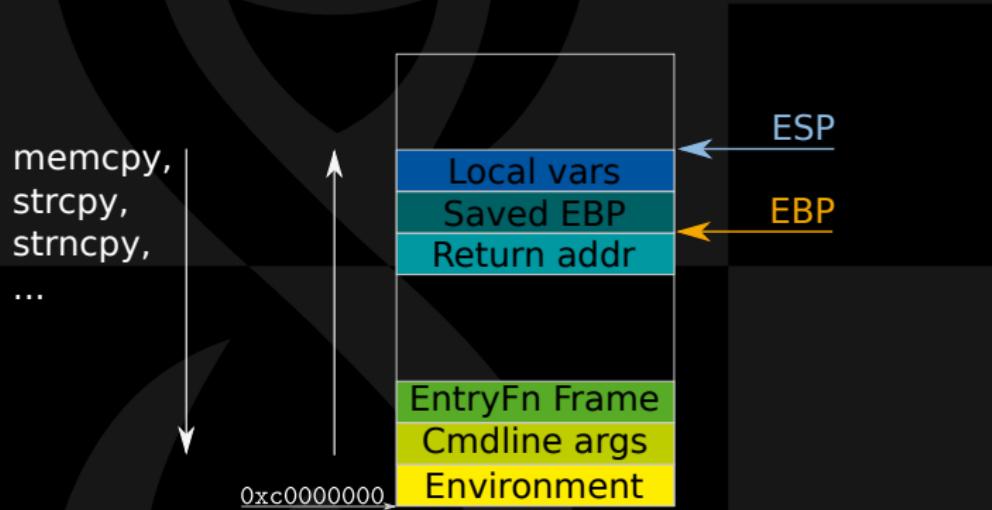
String and memory copy operations

- ▶ `void *memcpy(void *dest, const void *src, size_t n);`
- ▶ `char *strcpy(char *dest, const char *src);`
- ▶ `char *strncpy(char *dest, const char *src, size_t n);`
- ▶ `memmove, bcopy, memccpy, etc...`
- ▶ `gets, fgets, read, fread (files, network)`

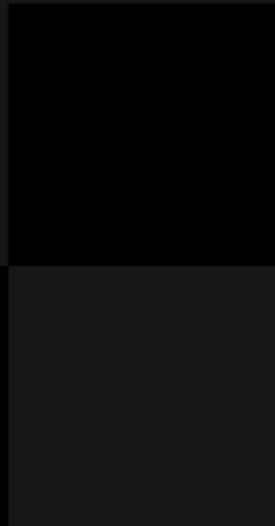
String and memory copy operations

- ▶ `void *memcpy(void *dest, const void *src, size_t n);`
- ▶ `char *strcpy(char *dest, const char *src);`
- ▶ `char *strncpy(char *dest, const char *src, size_t n);`
- ▶ `memmove, bcopy, memccpy, etc...`
- ▶ `gets, fgets, read, fread (files, network)`
- ▶ `rep movsb, ... (from [%esi] to [%edi], %ecx bytes)`

Stack growth vs copy operations



pwn demo



Pwning

- ▶ Check /u23/pwn for sources and binaries
- ▶ Use gdb and reproduce my exploit
- ▶ Use the shellcode provided and try to execute it instead
- ▶ Pwn sob2 (advanced)