Once upon a segfault

By Yusuf Hegazy

https://hegz.me

Agenda

- History
- x86 Crash Course
- Buffer Overflows Explained
- Demo
- Preventing Buffer Overflows

History: Beginnings

1. 1988: Morris Worm

History: Beginnings

1. 1988: Morris Worm
2. 1996: Smashing the Stack for Fun and Profit - Aleph One

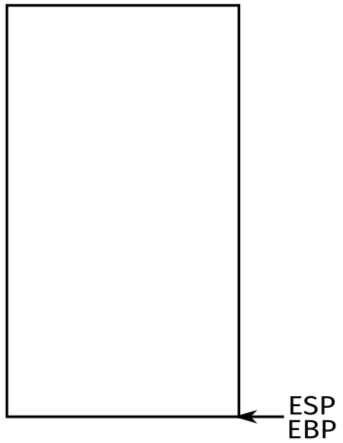
x86 Crash Course

Remarks

1. Assume 32 bits
    1. Register Size
    2. Memory Address Size
2. Hex is 4 bits a number

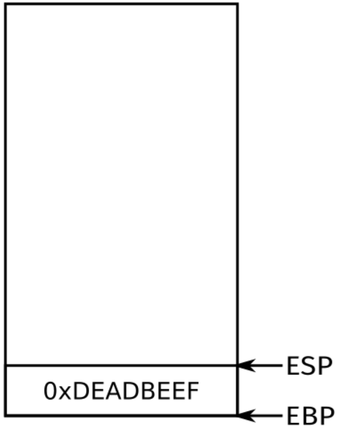
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



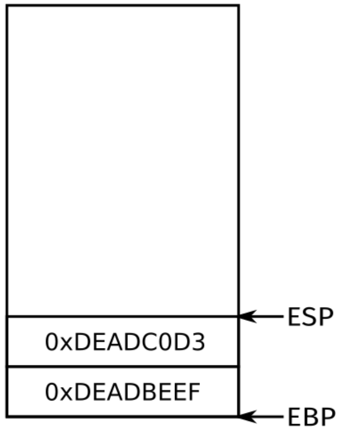
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



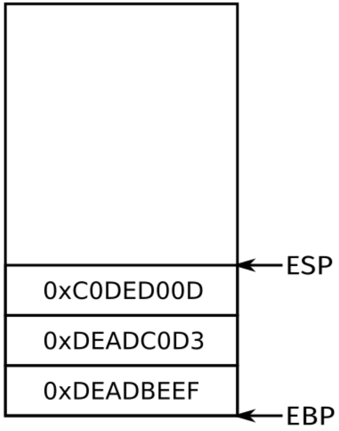
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



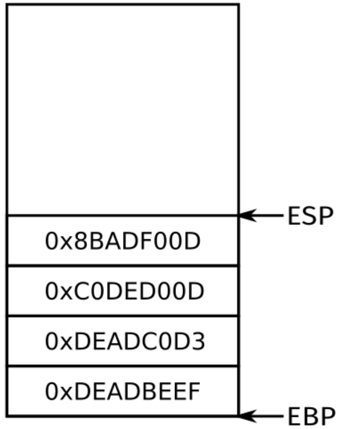
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



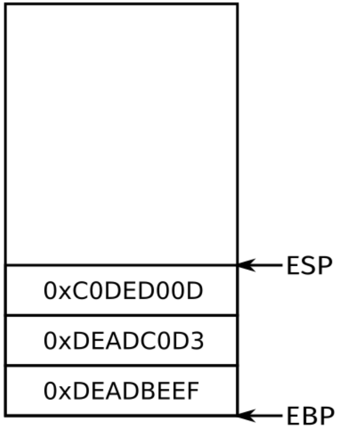
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



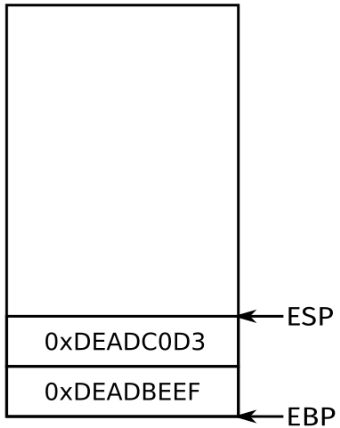
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



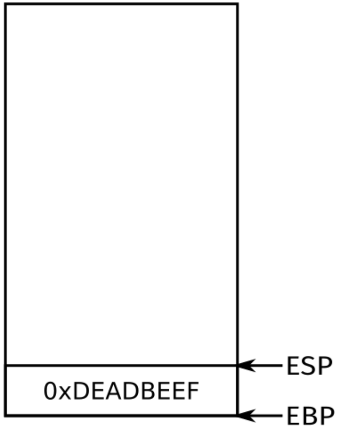
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



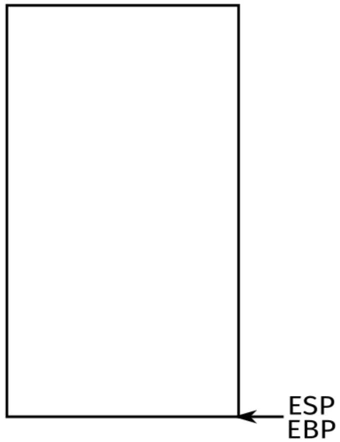
The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom



The Stack

- LIFO
- Grows towards lower addresses
- ESP –> Stack Top
- EBP –> Stack Bottom

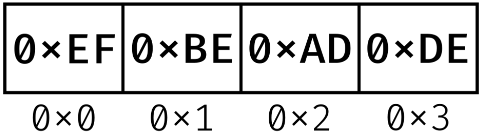


Little Endian Systems

LSB goes into the lowest memory addresses

Little Endian Systems

0×DEADBEEF



System V ABI

- Function Parameters are passed on the stack in reverse.

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction
    1. Address of next instruction is pushed on the stack

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction
    1. Address of next instruction is pushed on the stack
    2. Jump to the operand

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction
  1. Address of next instruction is pushed on the stack
  2. Jump to the operand
- Functions return to the caller using the ret instruction

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction
    1. Address of next instruction is pushed on the stack
    2. Jump to the operand
- Functions return to the caller using the ret instruction
    1. Pops value from the stack

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction
  1. Address of next instruction is pushed on the stack
  2. Jump to the operand
- Functions return to the caller using the ret instruction
  1. Pops value from the stack
  2. Jumps to that value

System V ABI

- Function Parameters are passed on the stack in reverse.
- Functions are called using the call instruction
    1. Address of next instruction is pushed on the stack
    2. Jump to the operand
- Functions return to the caller using the ret instruction
    1. Pops value from the stack
    2. Jumps to that value
- Stack is 16-bit aligned before call instruction is called.

System V ABI

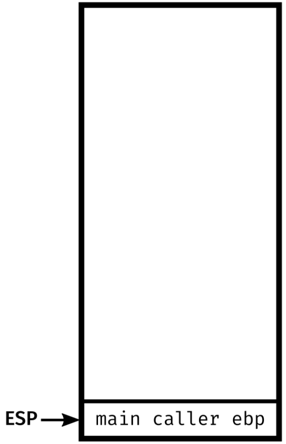
- EBP register
    - Saved in the function prologue
    - Restored in the function epilogue
    - Fixed in the stack frame
    - Local vars. are accessed using EBP-OFFSET

System V ABI: Demo

```
int add(int x, int y){
        return x + y;
}
void main(){
        int a, b, z;

        a = 2;
        b = 3;
        z = add(a, b) + 10;
}
```

System V ABI: Demo

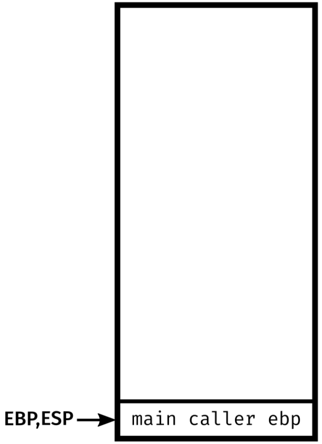
```
push ebp
```



@hegzploit

System V ABI: Demo

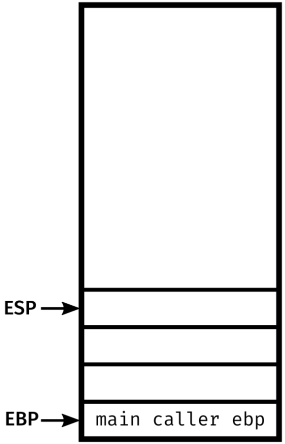
```
push ebp
mov ebp, esp
```



@hegzploit

System V ABI: Demo

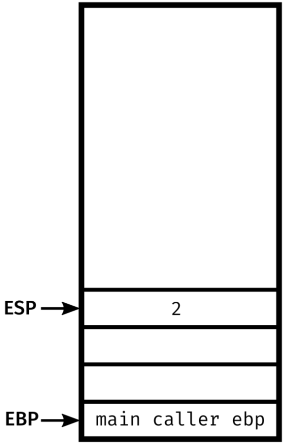
```
push ebp
mov ebp, esp
sub esp, 16
```



@hegzploit

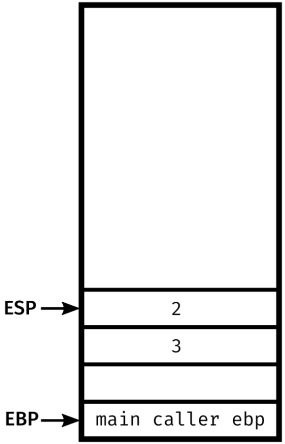
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
```



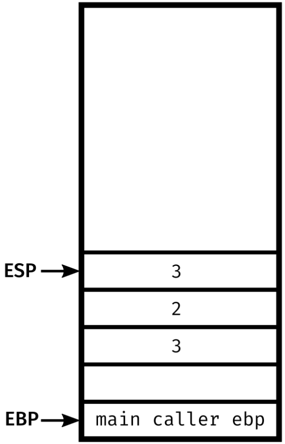
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
```



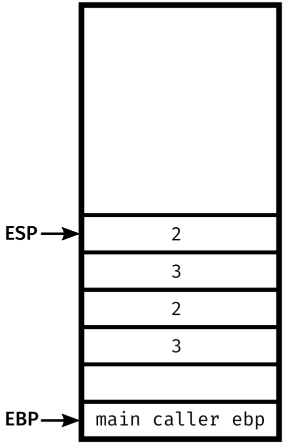
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
```



System V ABI: Demo

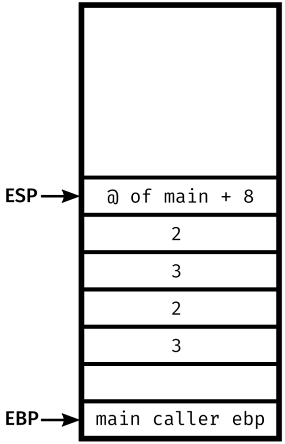
```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
```



@hegzploit

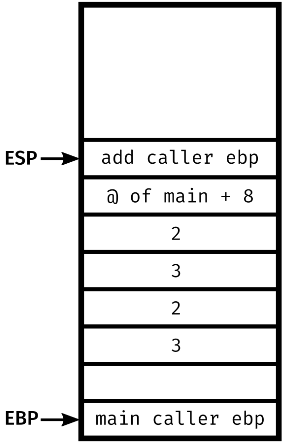
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
```



System V ABI: Demo

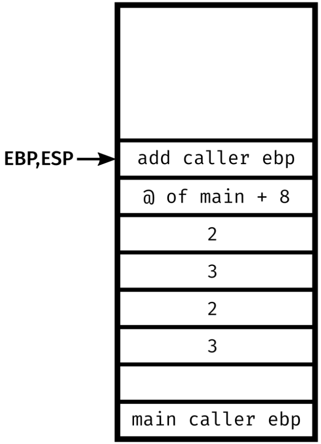
```
add:
push ebp
```



@hegzploit

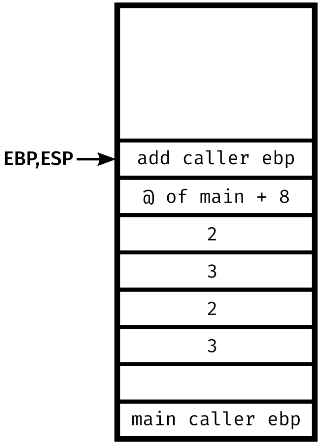
System V ABI: Demo

```
add:
push ebp
mov ebp, esp
```



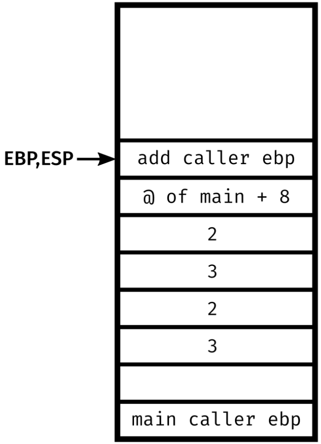
System V ABI: Demo

```
add:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
```



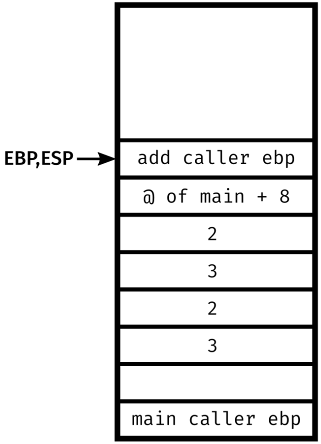
System V ABI: Demo

```
add:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
```



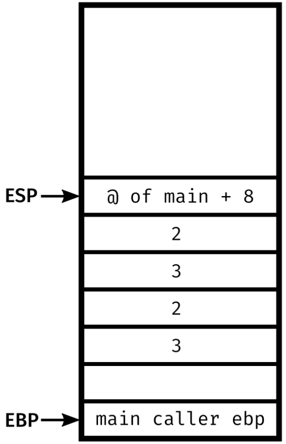
System V ABI: Demo

```
add:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add eax, edx
```



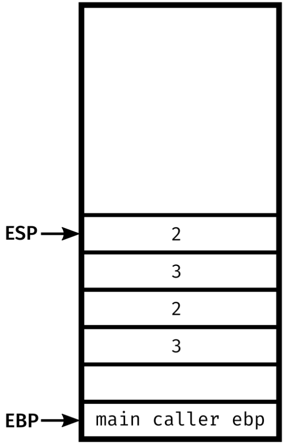
System V ABI: Demo

```
add:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add eax, edx
pop ebp
```



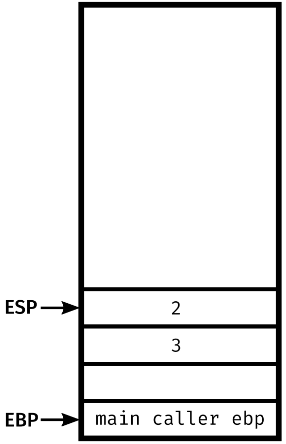
System V ABI: Demo

```
add:
push ebp
mov ebp, esp
mov edx, DWORD PTR [ebp+8]
mov eax, DWORD PTR [ebp+12]
add eax, edx
pop ebp
ret
```



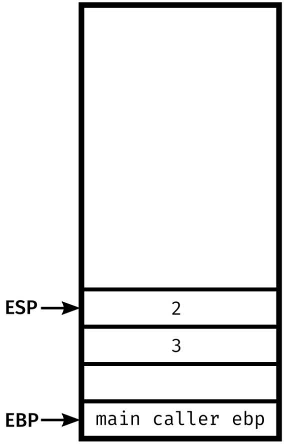
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
add esp, 8
```



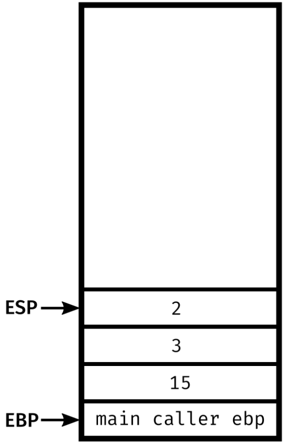
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
add esp, 8
add eax, 10
```



System V ABI: Demo

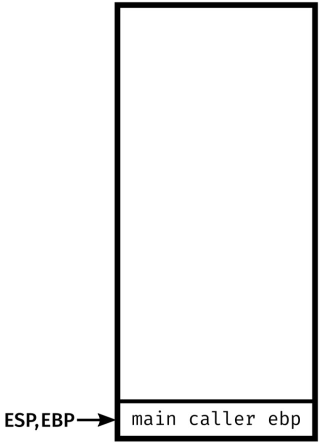
```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
add esp, 8
add eax, 10
mov DWORD PTR [ebp-4], eax
```



@hegzploit

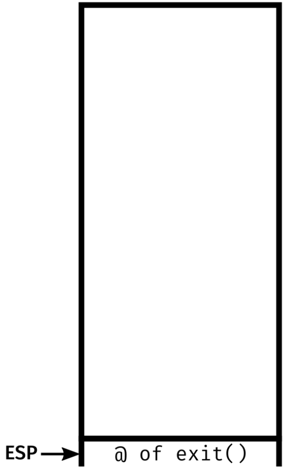
System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
add esp, 8
add eax, 10
mov DWORD PTR [ebp-4], eax
mov esp, ebp
```



System V ABI: Demo

```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
add esp, 8
add eax, 10
mov DWORD PTR [ebp-4], eax
mov esp, ebp
pop ebp
```



System V ABI: Demo

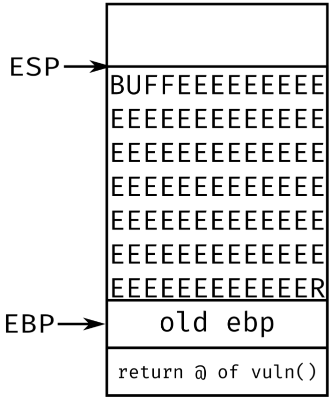
```
push ebp
mov ebp, esp
sub esp, 16
mov DWORD PTR [ebp-12], 2
mov DWORD PTR [ebp-8], 3
push 3
push 2
call add
add esp, 8
add eax, 10
mov DWORD PTR [ebp-4], eax
mov esp, ebp
pop ebp
ret
```

Buffer Overflows

```
void deadcode(){
        puts("Uhhh, I'm useless!");
}

void vuln(){
        char buffer[64];
        printf("What's your name? ");
        scanf("%s", &buffer);
        printf("Good Night Mr.%s\n", buffer);
}

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

Buffer Overflows

```
void deadcode(){
        puts("Uhhh, I'm useless!");
}

void vuln(){
        char buffer[64];
        printf("What's your name? ");
        scanf("%s", &buffer);
        printf("Good Night Mr.%s\n", buffer);
}

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



@hegzploit

Demo

More Exploitation Techniques

Return Oriented Programming

We gather pieces of instructions and rearrange them to get undefined behaviour!

- 1997: Getting around non-executable stack (and fix) - Solar Designer
- 2007: The geometry of innocent flesh on the bone (ROP) - Hovav Shacham

Return Oriented Programming/Return to LIBC

```c
char name[32];

int main() {
    printf("What's your name? ");
    read(0, name, 32);

    printf("Hi %s\n", name);

    printf("The time is currently ");
    system("/bin/date");

    char echo[100];
    printf("What do you want me to echo back? ");
    read(0, echo, 1000);
    puts(echo);

    return 0;
}
```

@hegzploit

Ret2Shellcode

- Attacker places his own shellcode on the stack
- Overwrites the return address with a pointer to the shellcode

Format String Attacks

- 2000: Format string vulnerability - Pascal Bouchareine
- Exploits missing format string in printf() family of functions.
- arbitrary read from memory
- arbitrary write to a pointer

Format String Attacks

```
int main(){
        char buffer[24];
        scanf("%s", buffer);
        printf(buffer);
}
```

Security Mitigations

- NX Stack (W^X Security)

Security Mitigations

- NX Stack (W^X Security)
- Stack Canary

Security Mitigations

- NX Stack (W^X Security)
- Stack Canary
- PIE

Security Mitigations

- NX Stack (W^X Security)
- Stack Canary
- PIE
- ASLR

Security Mitigations

- NX Stack (W^X Security)
- Stack Canary
- PIE
- ASLR
- RELRO