FONDAZIONE
BRUNO KESSLER

Security Testing Laboratory

Emanuele Viglianisi

Security&Trust Research Unit

Fondazione Bruno Kessler

Trento, Italy

eviglianisi@fbk.eu

FONDAZIONE
BRUNO KESSLER

Agenda

- Solve Homework-5
- Call stack example
- Exercises
  - Buffer Overflow

FONDAZIONE
BRUNO KESSLER

Homework-5

Homework-5:

1. Download the code from https://github.com/emavgl/sectestlab_2019/tree/master/lab_5/homework-5

2. Exploit Integer Overflow vulnerability in order to buy the iPhone for free.

3. Try to fix or mitigate the vulnerability

FONDAZIONE
BRUNO KESSLER

Call Stack

When you call a function, the system sets aside space in memory for that function to do its necessary work.

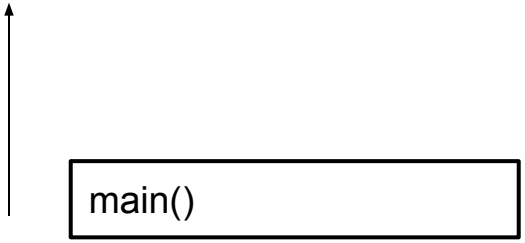
- We call such chunks of memory **stack frames**

FONDAZIONE
BRUNO KESSLER

Call Stack

When you call a function, the system sets aside space in memory for that function to do its necessary work.

- We call such chunks of memory **stack frames**

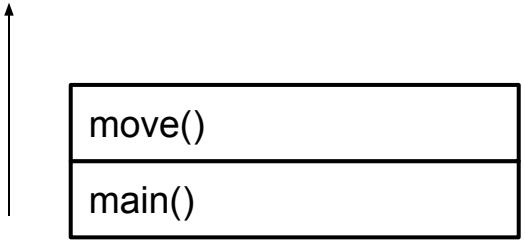


FONDAZIONE
BRUNO KESSLER

Call Stack

When you call a function, the system sets aside space in memory for that function to do its necessary work.

- We call such chunks of memory **stack frames**

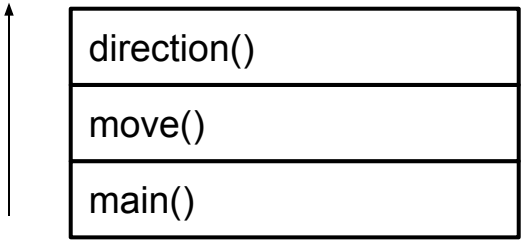


FONDAZIONE
BRUNO KESSLER

Call Stack

When you call a function, the system sets aside space in memory for that function to do its necessary work.

- We call such chunks of memory **stack frames**



FONDAZIONE BRUNO KESSLER

Call Stack

```c
#include <stdio.h>

int foo1(int a, int b) {
    int c = a + b;
    return c;
}

int main()
{
    foo1(1, 2);
    return 0;
}
```

Registries:

- **ESP**: points to the last thing pushed on the stack
- **EIP**: points to the next instruction to execute
- **EBP**: address of the frame's base

FONDAZIONE BRUNO KESSLER

Call Stack

```c
#include <stdio.h>

int foo1(int a, int b) {
    int c = a + b;
    return c;
}

int main()
{
    foo1(1, 2);
    return 0;
}
```

Registries:

- **ESP**: points to the last thing pushed on the stack
- **EIP**: points to the next instruction to execute
- **EBP**: address of the frame's base

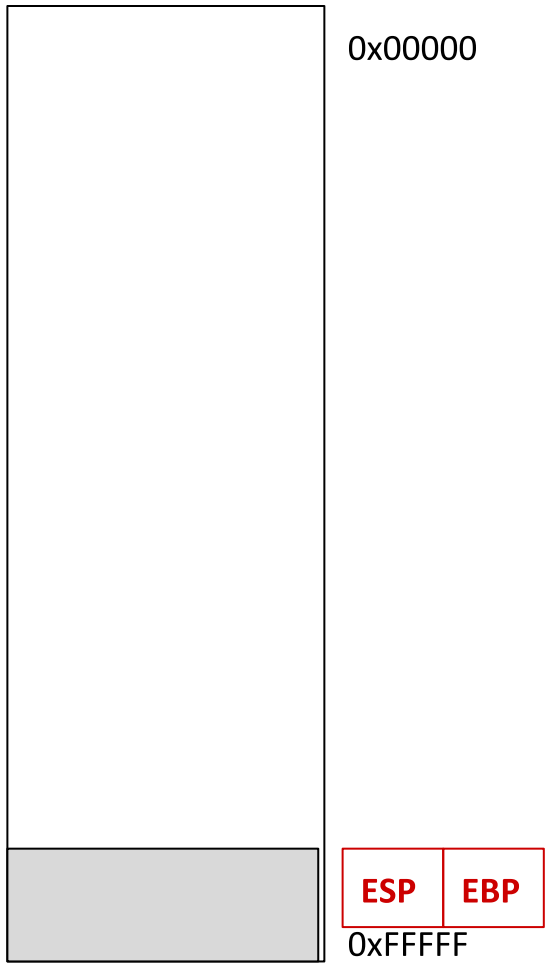
CALL <addr>
pushes the current value of **EIP** and changes EIP to *<addr>*

Arguments are pushed onto the stack before a function call

Call Stack

```
// Main's code
// ...
0x00000531 <+13>:    push   $0x2
0x00000533 <+15>:    push   $0x1
0x00000535 <+17>:    call   0x4ed <foo1>
0x0000053a <+22>:    add    $0x8,%esp
```

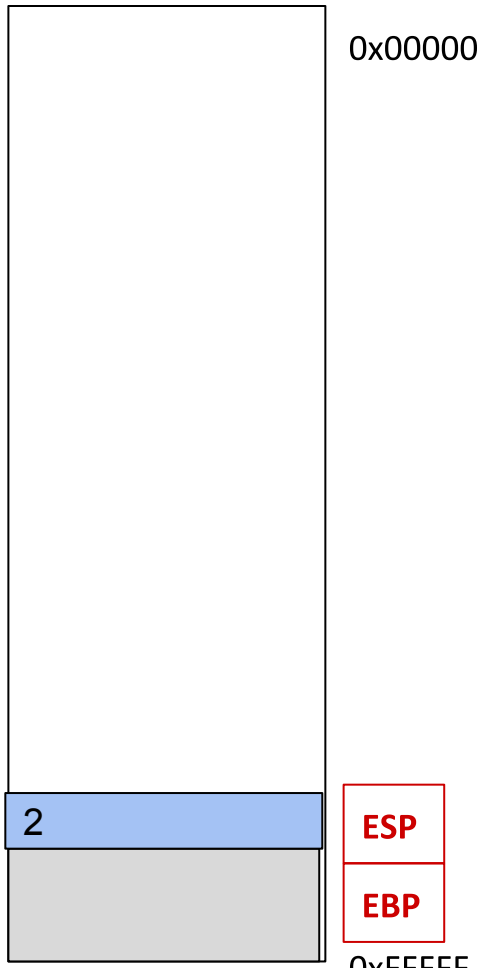
```
int main()
{
    foo1(1, 2);
    return 0;
}
```



Calling foo1

```
// Main's code
// ...
0x00000531 <+13>:    push   $0x2
0x00000533 <+15>:    push   $0x1 EIP
0x00000535 <+17>:    call   0x4ed <foo1>
0x0000053a <+22>:    add    $0x8,%esp
```

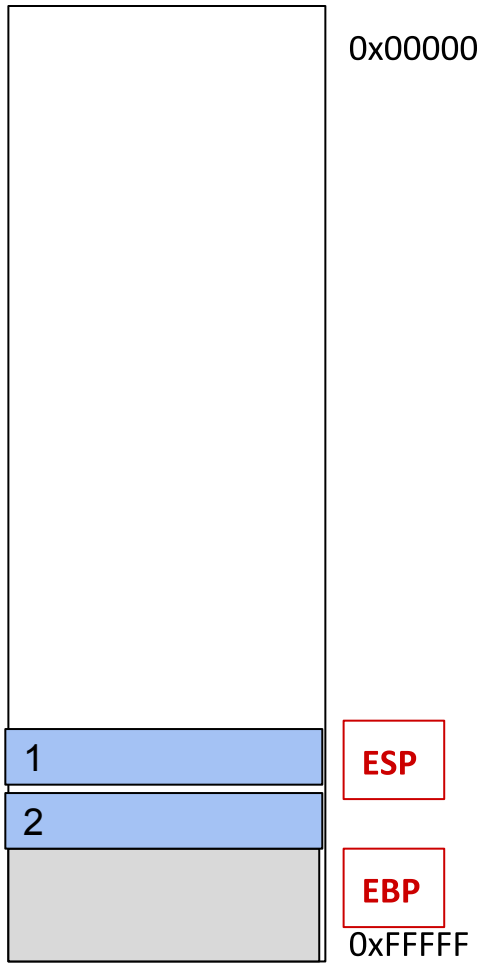
```
int main()
{
    foo1(1, 2);
    return 0;
}
```



Calling foo1

```
// Main's code
// ...
0x00000531 <+13>:    push   $0x2
0x00000533 <+15>:    push   $0x1
0x00000535 <+17>:    call   0x4ed <foo1> EIP
0x0000053a <+22>:    add    $0x8,%esp
```

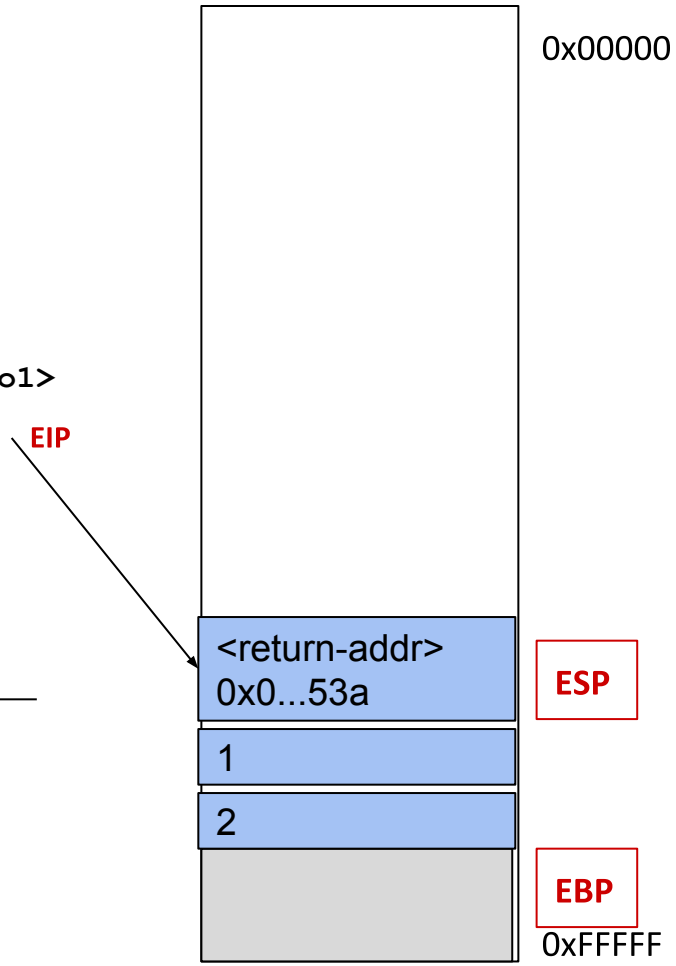
```
int main()
{
    foo1(1, 2);
    return 0;
}
```



Calling foo1

```
// Main's code
// ...
0x00000531 <+13>:    push   $0x2
0x00000533 <+15>:    push   $0x1
0x00000535 <+17>:    call   0x4ed <foo1>
0x0000053a <+22>:    ...
```

```
int main()
{
    foo1(1, 2);
    return 0;
}
```

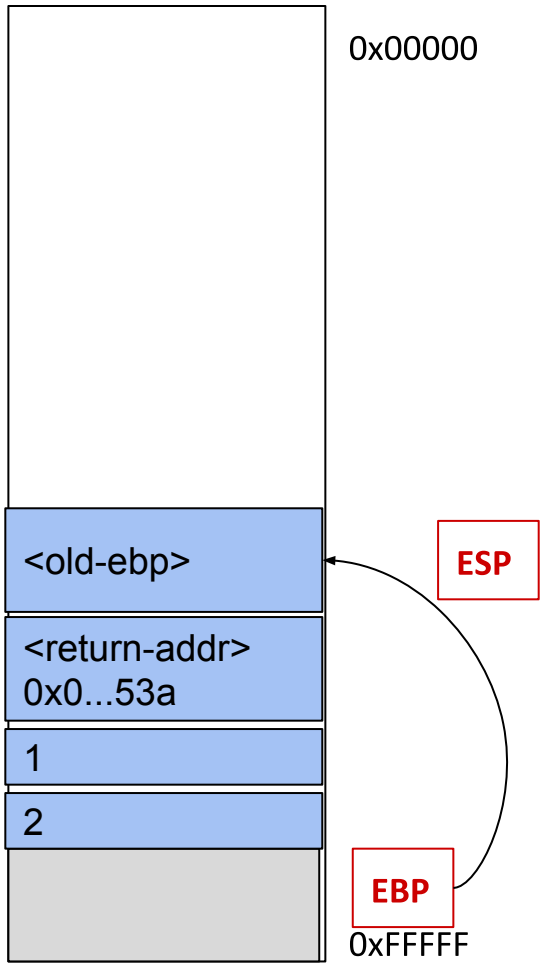


Calling foo1

```
// Foo1's code
// ...
0x00000504 <+0>:     push   %ebp
0x00000505 <+1>:     mov    %esp,%ebp
0x00000514 <+16>:    mov    0x8(%ebp),%edx
0x00000517 <+19>:    mov    0xc(%ebp),%eax
0x0000051a <+22>:    add    %edx,%eax
0x0000051c <+24>:    mov    %eax,-0x4(%ebp)
0x0000051f <+27>:    mov    -0x4(%ebp),%eax
0x00000522 <+30>:    leave
0x00000523 <+31>:    ret
```

EIP

```
int foo1(int a, int b) {
    int c = a + b;
    return c;
}
```

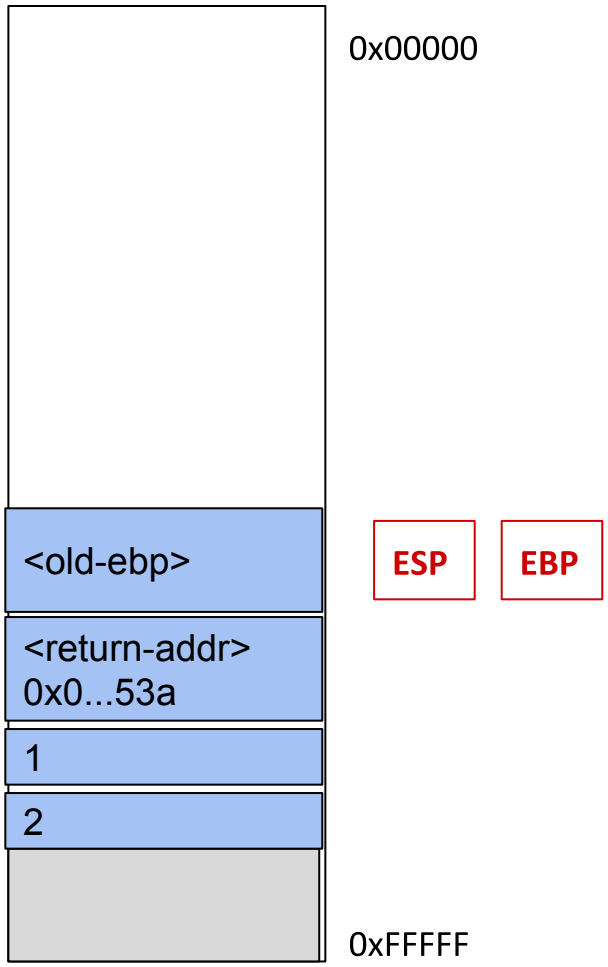


Calling foo1

```
// Foo1's code
// ...
0x00000504 <+0>:     push   %ebp
0x00000505 <+1>:     mov    %esp,%ebp
0x00000514 <+16>:    mov    0x8(%ebp),%edx
0x00000517 <+19>:    mov    0xc(%ebp),%eax
0x0000051a <+22>:    add    %edx,%eax
0x0000051c <+24>:    mov    %eax,-0x4(%ebp)
0x0000051f <+27>:    mov    -0x4(%ebp),%eax
0x00000522 <+30>:    leave
0x00000523 <+31>:    ret
```

EIP

```
int foo1(int a, int b) {
    int c = a + b;
    return c;
}
```

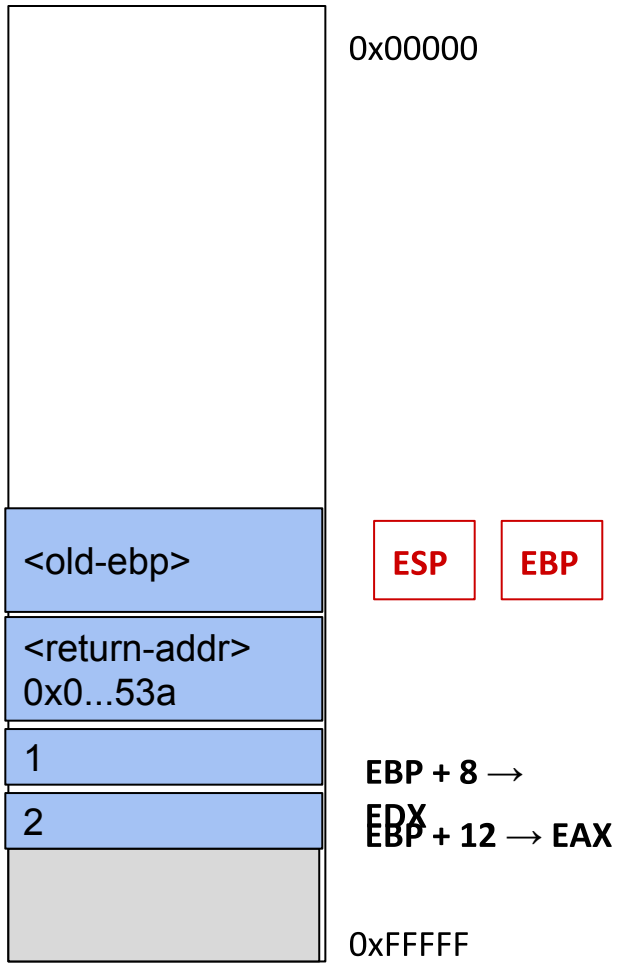


Calling foo1

```
// Foo1's code
// ...
0x00000504 <+0>:     push   %ebp
0x00000505 <+1>:     mov    %esp,%ebp
0x00000514 <+16>:    mov    0x8(%ebp),%edx
0x00000517 <+19>:    mov    0xc(%ebp),%eax
0x0000051a <+22>:    add    %edx,%eax
0x0000051c <+24>:    mov    %eax,-0x4(%ebp)
0x0000051f <+27>:    mov    -0x4(%ebp),%eax
0x00000522 <+30>:    leave
0x00000523 <+31>:    ret
```

EIP

```
int foo1(int a, int b) {
    int c = a + b;
    return c;
}
```

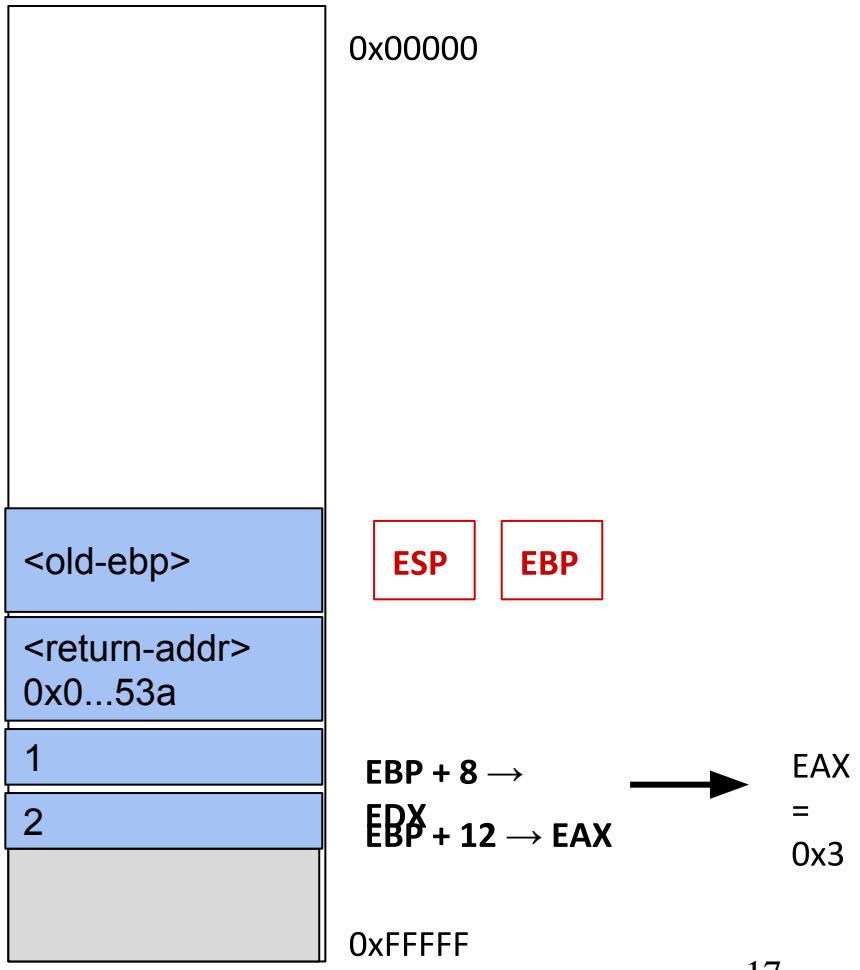


Calling foo1

```
// Foo1's code
// ...
0x00000504 <+0>:     push   %ebp
0x00000505 <+1>:     mov    %esp,%ebp
0x00000514 <+16>:    mov    0x8(%ebp),%edx
0x00000517 <+19>:    mov    0xc(%ebp),%eax
0x0000051a <+22>:    add    %edx,%eax
0x0000051c <+24>:    mov    %eax,-0x4(%ebp)
0x0000051f <+27>:    mov    -0x4(%ebp),%eax
0x00000522 <+30>:    leave
0x00000523 <+31>:    ret
```

EIP

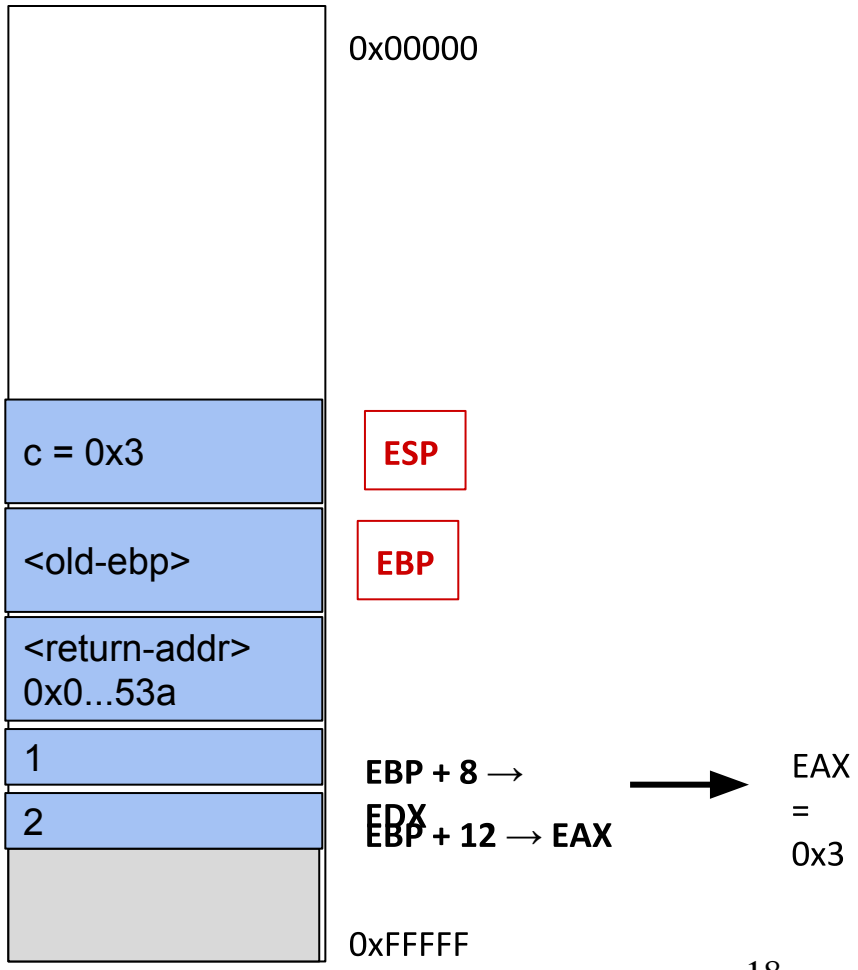
```
int foo1(int a, int b) {
    int c = a + b;
    return c;
}
```



Calling foo1

```
// Foo1's code
// ...
0x00000504 <+0>:     push   %ebp
0x00000505 <+1>:     mov    %esp,%ebp
0x00000514 <+16>:    mov    0x8(%ebp),%edx
0x00000517 <+19>:    mov    0xc(%ebp),%eax
0x0000051a <+22>:    add    %edx,%eax
0x0000051c <+24>:    mov    %eax,-0x4(%ebp)
0x0000051f <+27>:    mov    -0x4(%ebp),%eax
0x00000522 <+30>:    leave
0x00000523 <+31>:    ret
```

```
int foo1(int a, int b) {
    int c = a + b;
    return c;
}
```

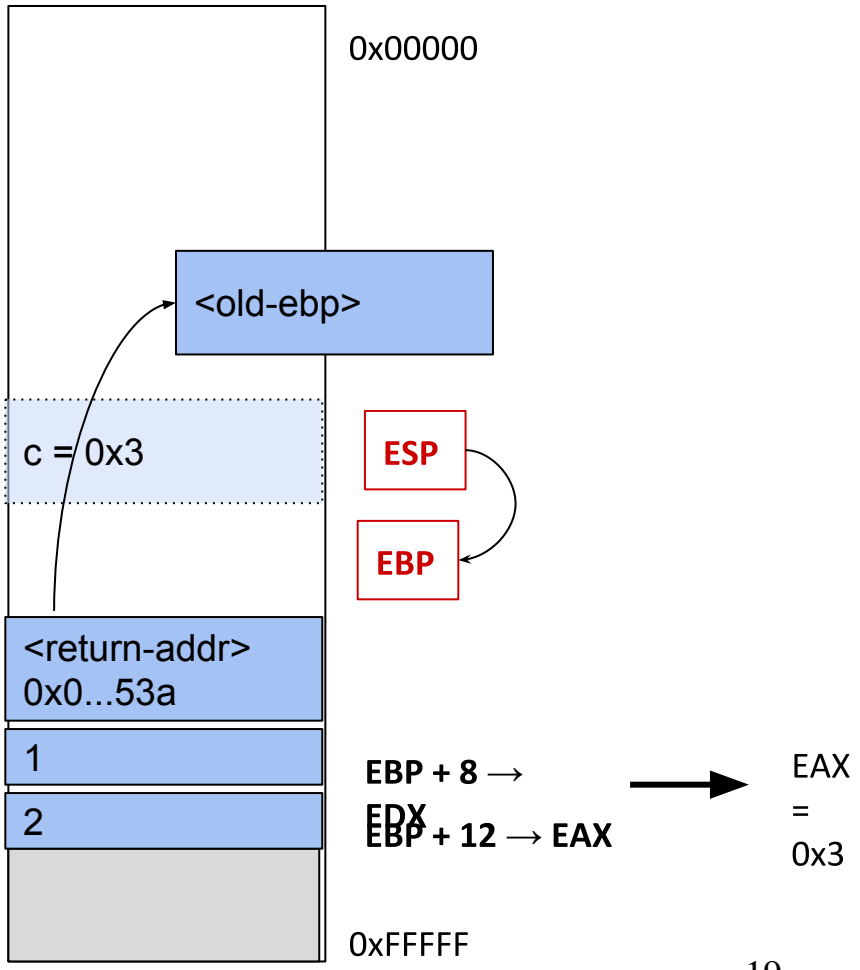


Calling foo1

```
// Foo1's code
// ...
0x00000504 <+0>:     push   %ebp
0x00000505 <+1>:     mov    %esp,%ebp
0x00000514 <+16>:    mov    0x8(%ebp),%edx
0x00000517 <+19>:    mov    0xc(%ebp),%eax
0x0000051a <+22>:    add    %edx,%eax
0x0000051c <+24>:    mov    %eax,-0x4(%ebp)
0x0000051f <+27>:    mov    -0x4(%ebp),%eax
0x00000522 <+30>:    leave
0x00000523 <+31>:    ret
```

EIP

```
int foo1(int a, int b) {
    int c = a + b;
    return c;
}
```



Buffer Overflow

Buffer Overflow (BOF) consists on reading/writing more than the allocated buffer amount

```
int foo() {
    char a;
    char buf[3];
    char password[] = "ciao";
    strcpy(buf, password);
    return 0;
}

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

Buffer of 3 chars

We are trying to copy 4 chars to the buffer

Buffer Overflow

Buffer Overflow (BOF) consists on reading/writing more than the allocated buffer amount

```
int foo() {
    char a;
    char buf[3];
    char password[] = "ciao";
    strcpy(buf, password);
    return 0;
}

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