Cyber Attacks & Defense

Buffer Overflow

Dr. Yeongjin Jang

Oregon State University

Week1

- Due: 04/13 2:00pm
- Late submission due: 04/20 2:00pm (50% pts)
  - You will not get any point after this date

Week2 Objectives

- Understand how stack works in Linux x86/amd64 ABIs
- Understand why buffer overflow could be a security vulnerability
- Identify a buffer overflow vulnerability in the program
- Exploit a buffer overflow vulnerability to hijack the control of the program

X86 Stack (pointed by %esp/%rsp)

- Stores **local variables** (negative indexing over ebp)
  - `mov -0x8(%ebp), %eax` -- value at ebp-0x8
  - `lea -0x24(%ebp), %eax` -- address at ebp-0x24
- Stores **function arguments from caller** (positive indexing over ebp)
  - `mov 0x8(%ebp), %eax` -- 1st arg
  - `mov 0xc(%ebp), %eax` -- 2nd arg
- Pushes **function arguments to callee** (positive indexing over esp)
  - `mov %eax, 0(%esp)` -- 1st arg
  - `mov $0x4, 0x4(%esp)` -- 2nd arg

Calling Convention
Receiving Func Arguments from Caller

- x86 (32 bit) – pushes argument on the stack (for callee)
  - **0x4 * n** (%esp) indicates n-th argument
  - **0**(%esp) – accessed by callee via **0x8**(%ebp)
  - **0x4**(%esp) – accessed by callee via **0xc**(%ebp)
  - **0x8**(%esp) – accessed by callee via **0x10**(%ebp)
  - Focus on +0x8, +0xc, +0x10, etc., for the index on %ebp at the callee…

- amd64 (64 bit) – use registers to pass argument to callee
  - Register order (1st, 2nd, 3rd, 4th, 5th, 6th, etc.)
    - `%rdi, %rsi, %rdx, %rcx, %r8, %r9`, … (use stack for more arguments)
  - Callee also uses these registers to get arguments

Examples

printf(0x8048727, ebp_8, ebp_c)

- printf() in x86

```
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)          1st
mov     %ecx,0x4(%esp)       2nd
mov     %edx,0x8(%esp)       3rd
call    0x8048370 <printf@plt>
```

printf(0x400905, rbp_8, rbp_10)

- printf() in amd64

```
movabs $0x400905,%rdi        1st
mov    -0x8(%rbp),%rsi       2nd
mov    -0x10(%rbp),%rdx      3rd
callq  0x400500 <printf@plt>
```

In amd64

- Stores local variables (negative access over rbp)
  - `mov -0x8(%rbp), %rax`
  - `lea -0x24(%rbp), %rax`
- Function arguments to **callee** does not use stack
  - `mov %rax, %rdi`
  - `mov $0x4, %rsi`
  - `mov $0x40006, %rdx`
- Function arguments from **caller** does not use stack
  - `mov %rdi, %rax`
  - `mov %rsi, %rbx`

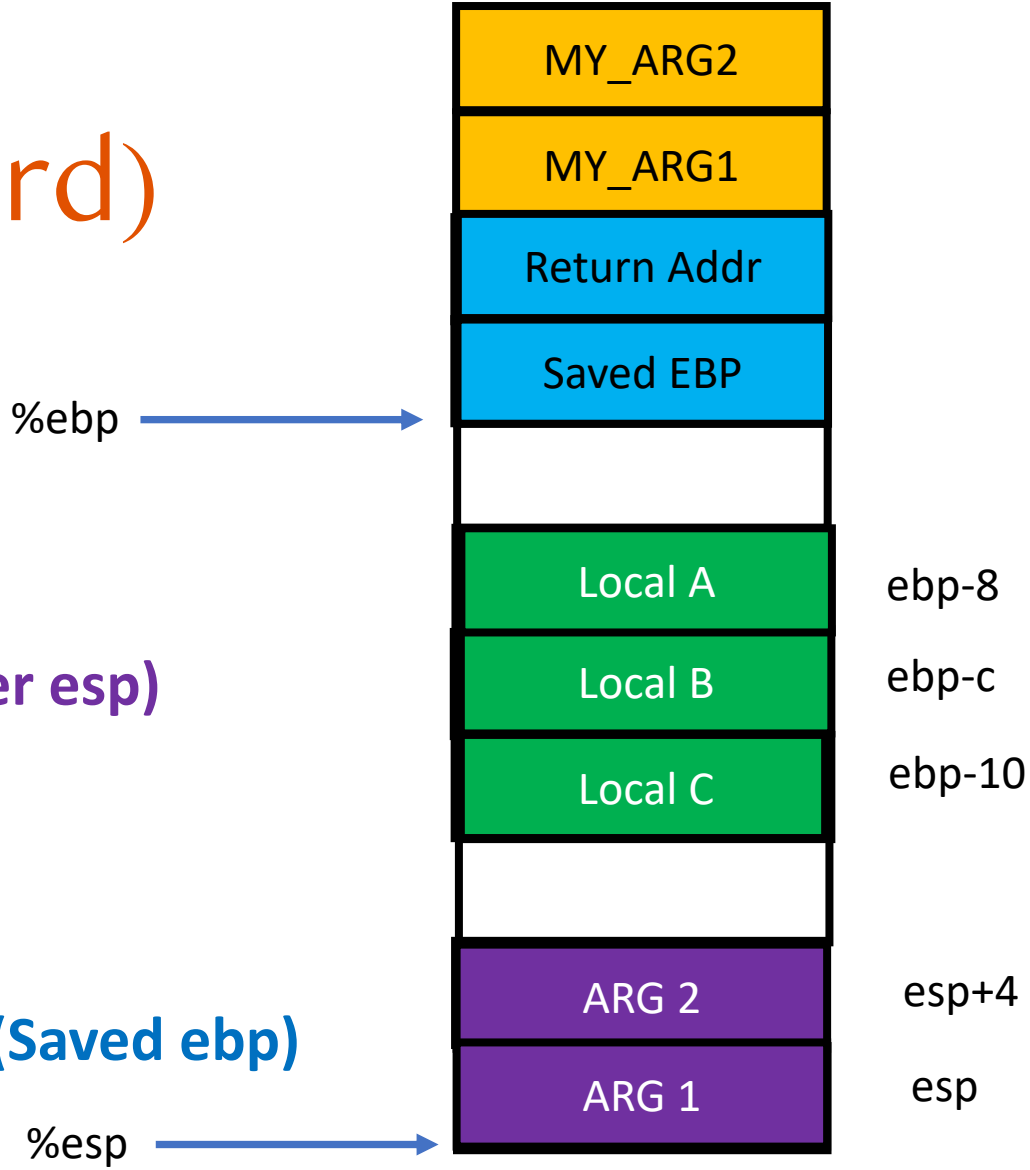
amd64 Calling Convention

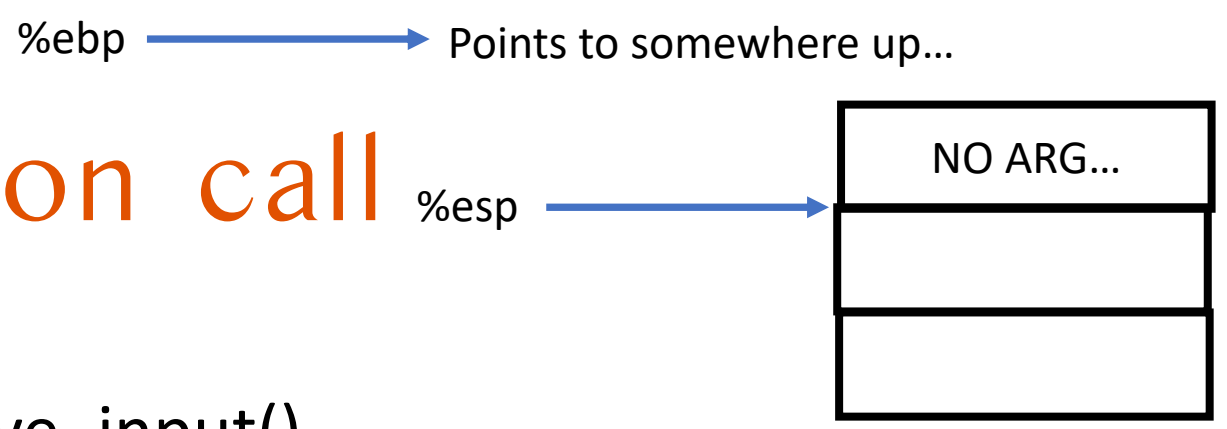
%rdi – 1st arg
%rsi – 2nd arg
%rdx – 3rd arg
%rcx – 4th arg
%r8 – 5th arg
%r9 – 6th arg

It has 16 registers,
rax, rbx, rcx, rdx, rsi, rdi, rbp, rsp
r8, r9, r10, r11, r12, r13, r14, r15

Stack (Grows Downward)

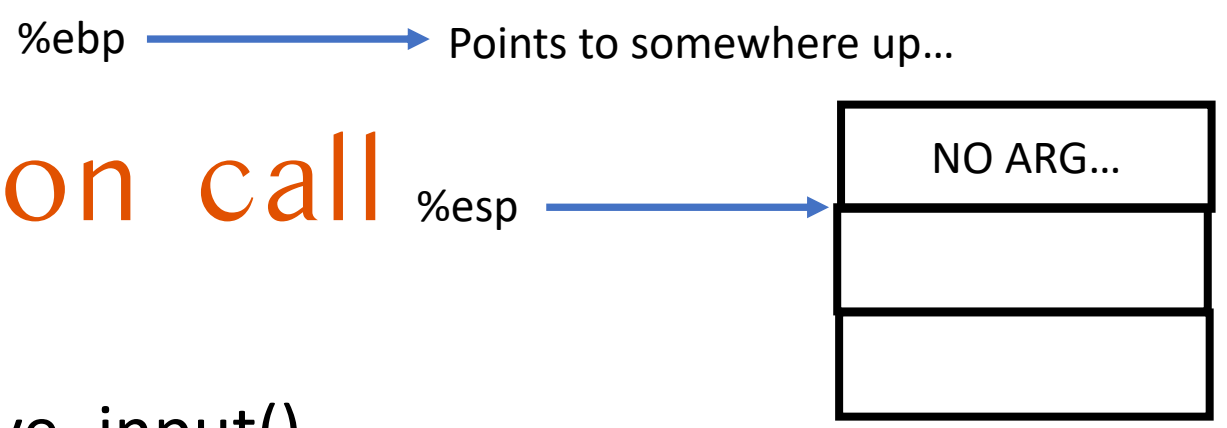
- Defines a variable scope of a function
  - **Local variables (negative index over ebp)**
  - **Arguments (positive index over ebp)**
  - **Function call arguments (positive index over esp)**
- Maintains nested function calls
  - **Return target (return address)**
  - **Local variables of the upper level function (Saved ebp)**
- Starts at `%ebp` (bottom), ends at `%esp` (top)





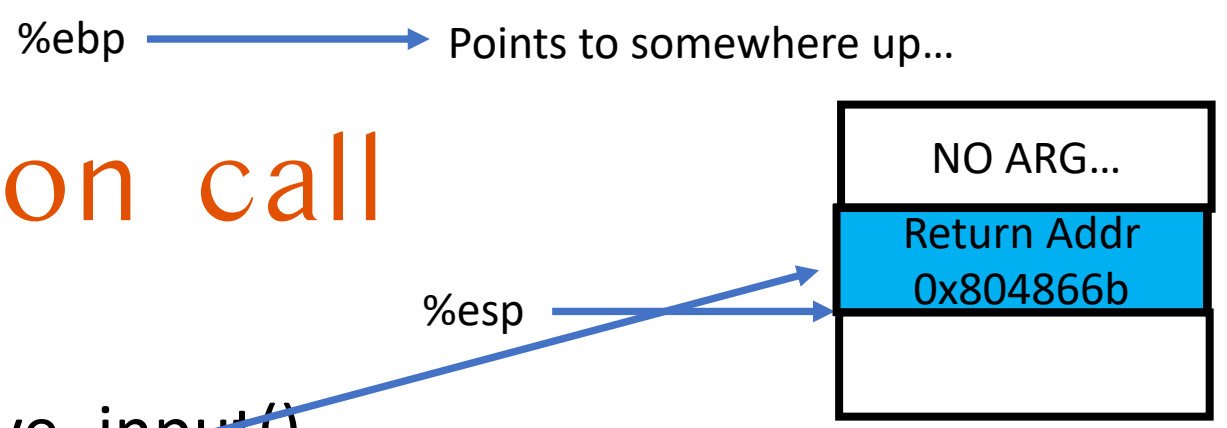
Example – Function call

- In bof-level0, main() calls receive_input()
  - `call   0x8048570 <receive_input>`
  - `0x0804866b <+11>:    xor    %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push   %ebp`
  - `0x08048571 <+1>: mov    %esp,%ebp`
  - `0x08048573 <+3>: push   %esi`
  - `0x08048574 <+4>: sub    $0x54,%esp`



Example – Function call

- In bof-level0, main() calls receive_input()
  - **`call 0x8048570 <receive_input>`**
  - `0x0804866b <+11>: xor %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push %ebp`
  - `0x08048571 <+1>: mov %esp,%ebp`
  - `0x08048573 <+3>: push %esi`
  - `0x08048574 <+4>: sub $0x54,%esp`

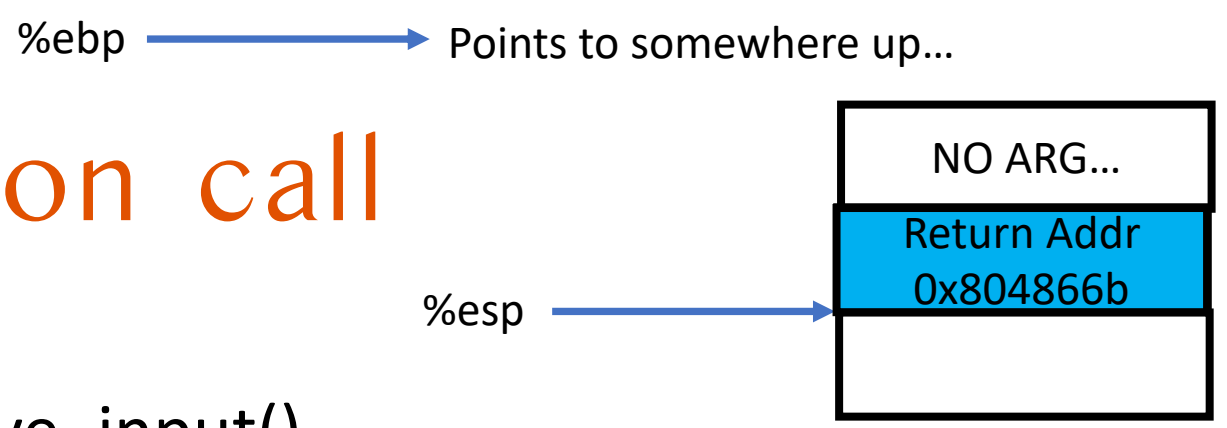


Example – Function call

- In bof-level0, main() calls receive_input()
  - **`call 0x8048570 <receive_input>`**
  - `0x0804866b <+11>: xor %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push %ebp`
  - `0x08048571 <+1>: mov %esp,%ebp`
  - `0x08048573 <+3>: push %esi`
  - `0x08048574 <+4>: sub $0x54,%esp`

Call: push the address to return to the stack, then jump!

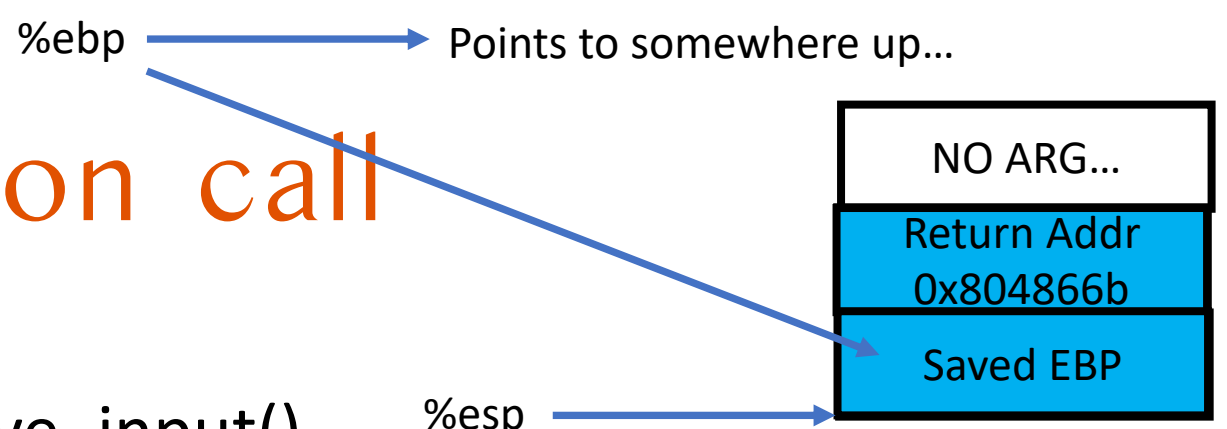
```
push %eip   -- points the next instruction
jmp 0x8048570 <receive_input>
```



Example – Function call

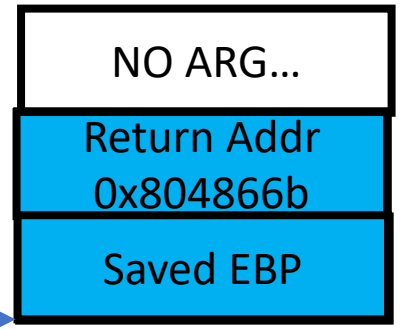
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - **`0x08048570 <+0>: push  %ebp`**
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`

Example – Function call



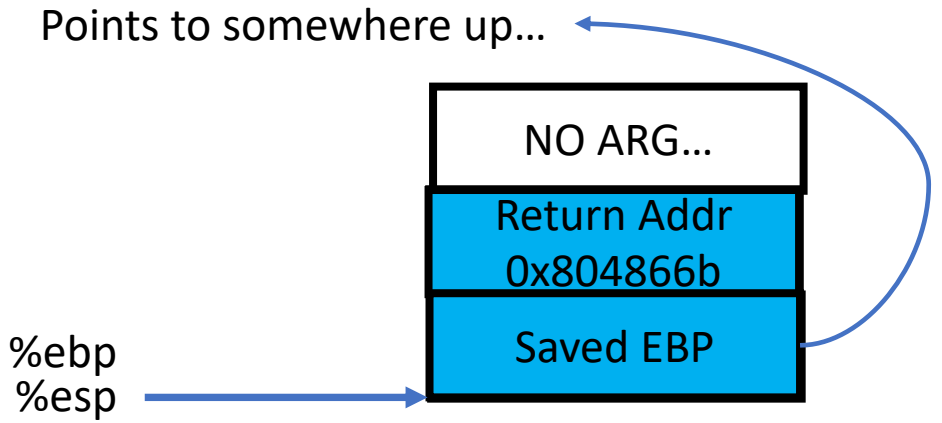
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - **`0x08048570 <+0>: push  %ebp`**
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`

Example – Function call



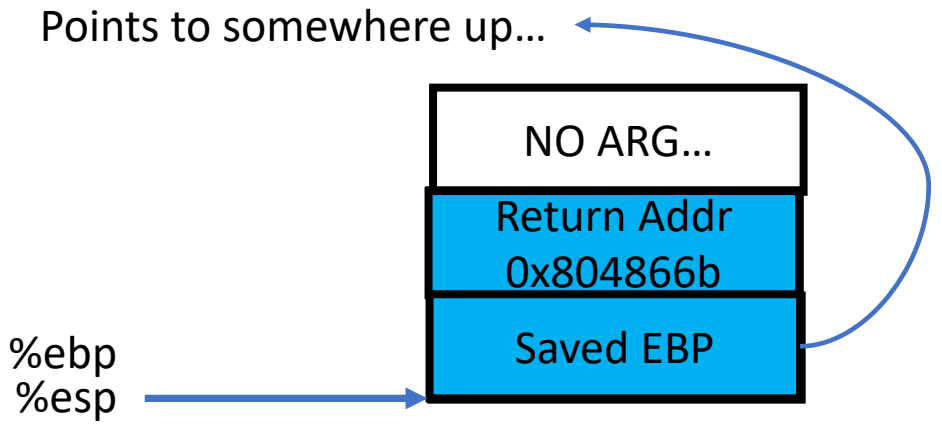
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - **`0x08048571 <+1>: mov  %esp,%ebp`**
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`

Example – Function call



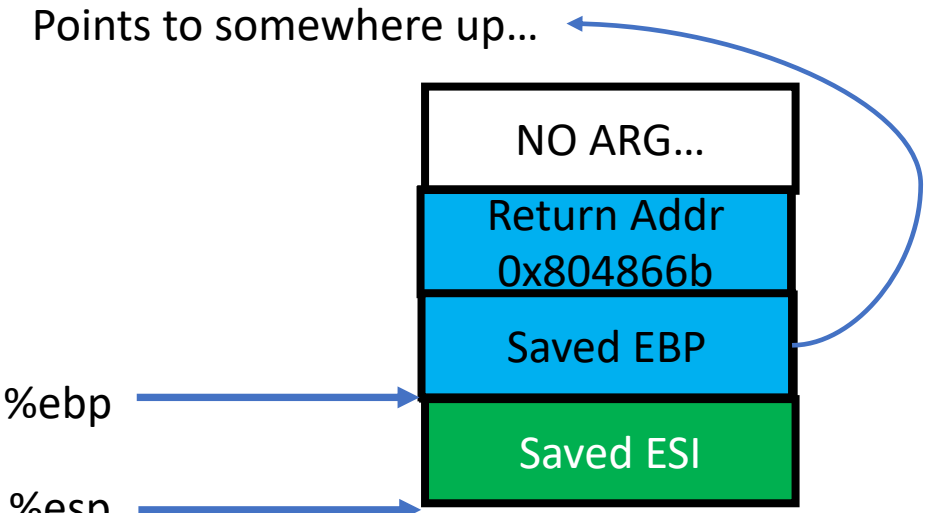
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - **`0x08048571 <+1>: mov  %esp,%ebp`**
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`

Example – Function call



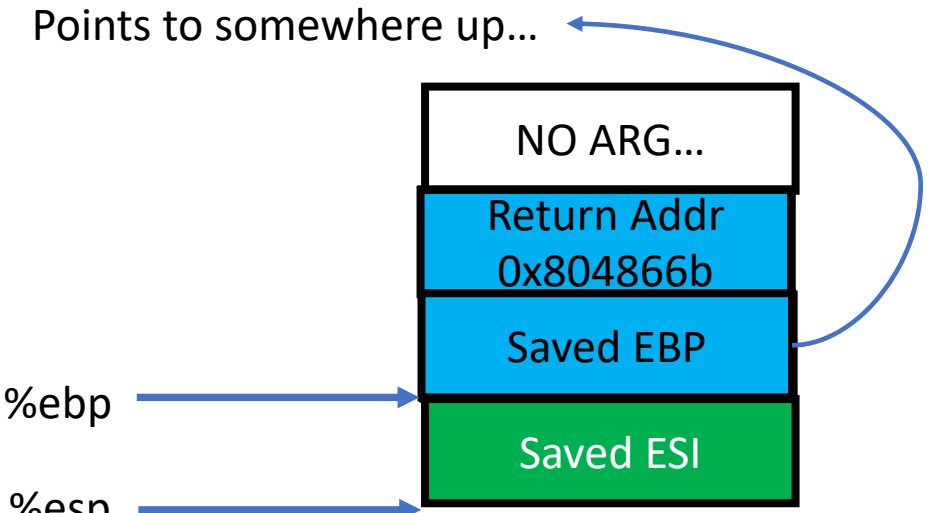
- In bof-level0, main() calls receive_input()
  - `call 0x8048570 <receive_input>`
  - `0x0804866b <+11>: xor %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push %ebp`
  - `0x08048571 <+1>: mov %esp,%ebp`
  - **`0x08048573 <+3>: push %esi`**
  - `0x08048574 <+4>: sub $0x54,%esp`

Example – Function call



- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - **`0x08048573 <+3>: push  %esi`**
  - `0x08048574 <+4>: sub  $0x54,%esp`

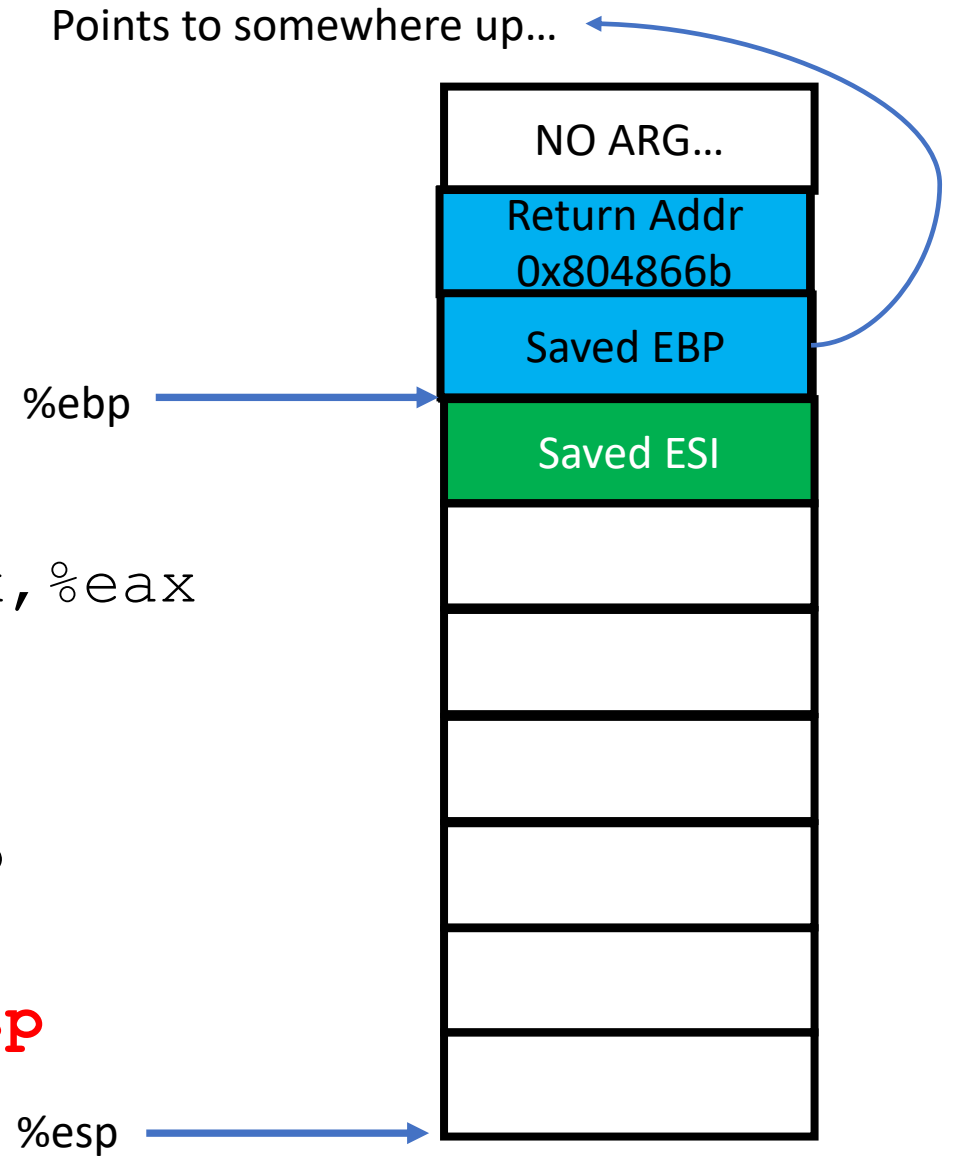
Example – Function call



- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - `0x08048573 <+3>: push  %esi`
  - **`0x08048574 <+4>: sub  $0x54,%esp`**

Example – Function call

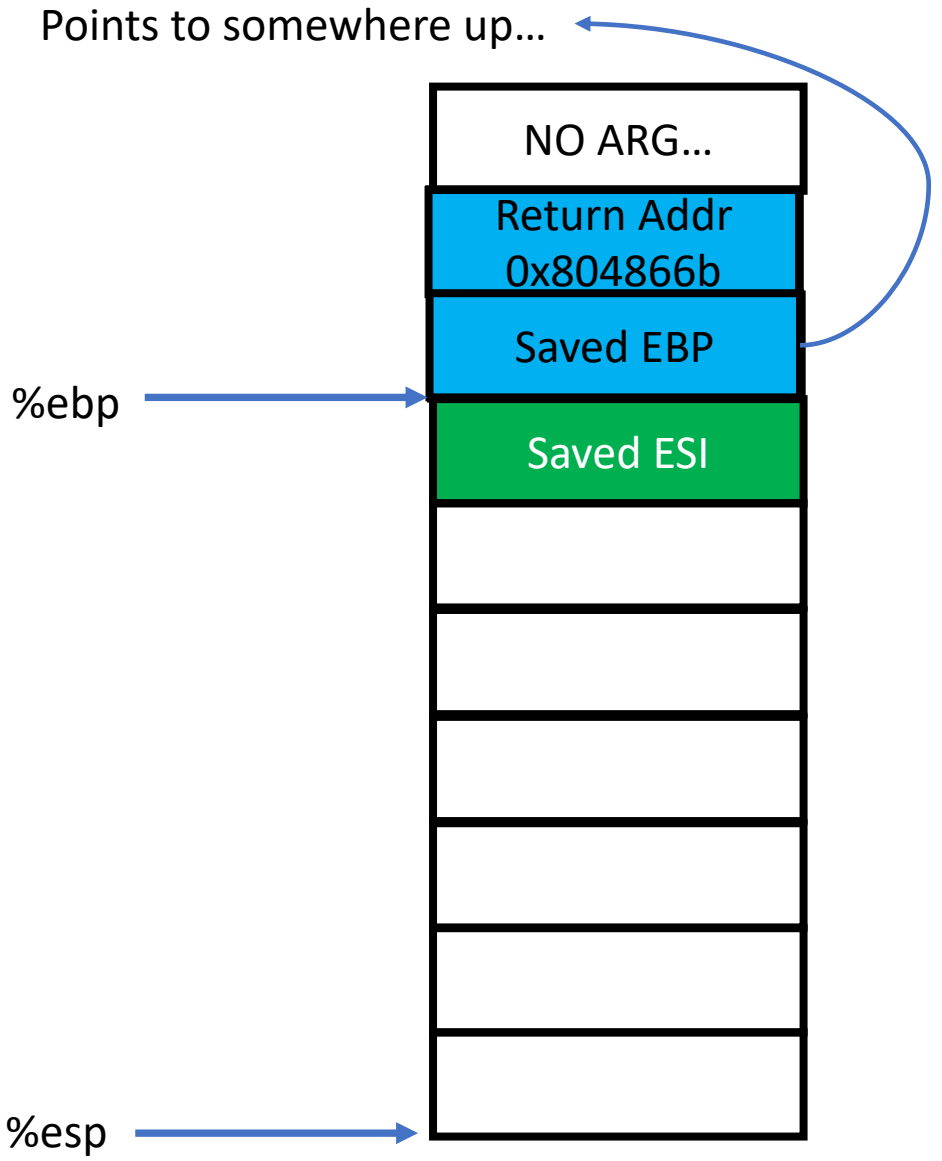
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - `0x08048573 <+3>: push  %esi`
  - **`0x08048574 <+4>: sub  $0x54,%esp`**



Example – Function call

- Call printf?

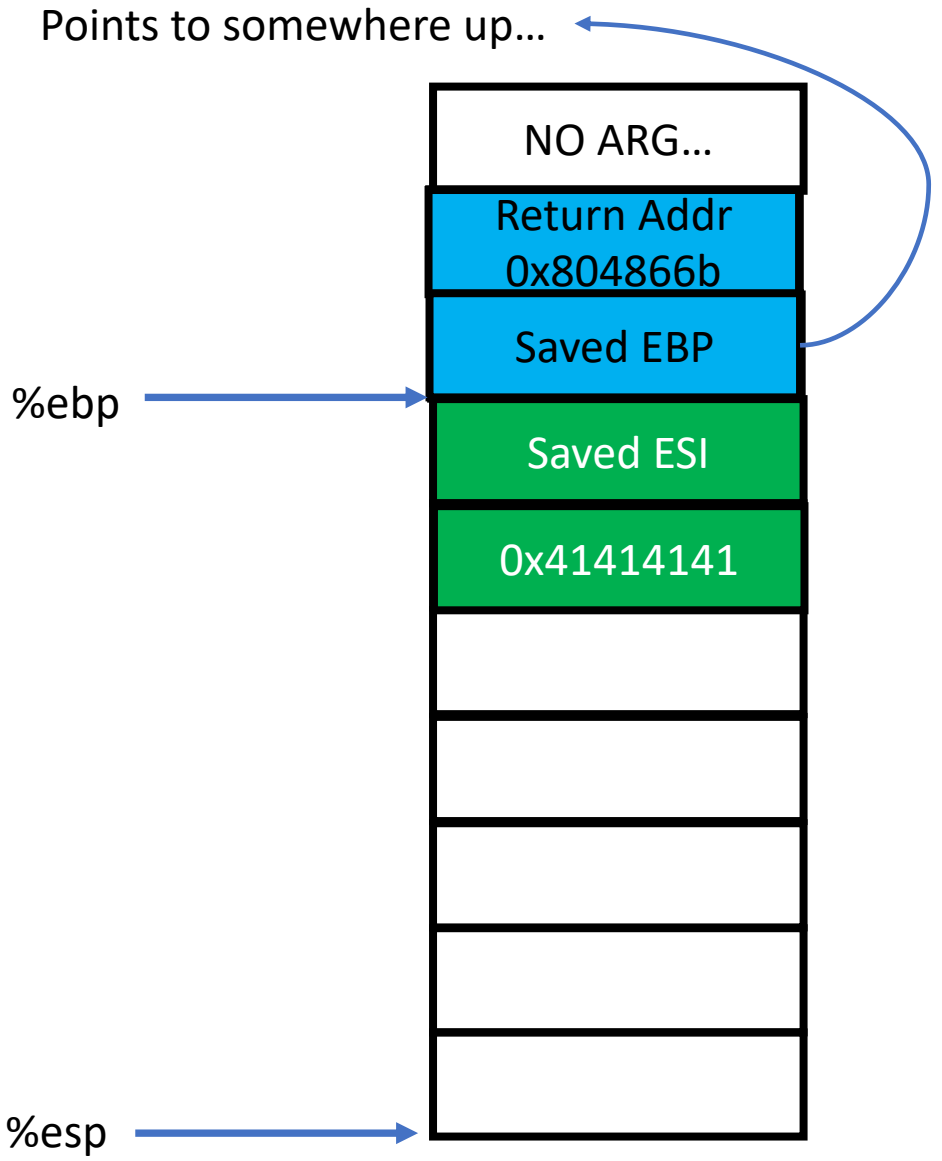
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

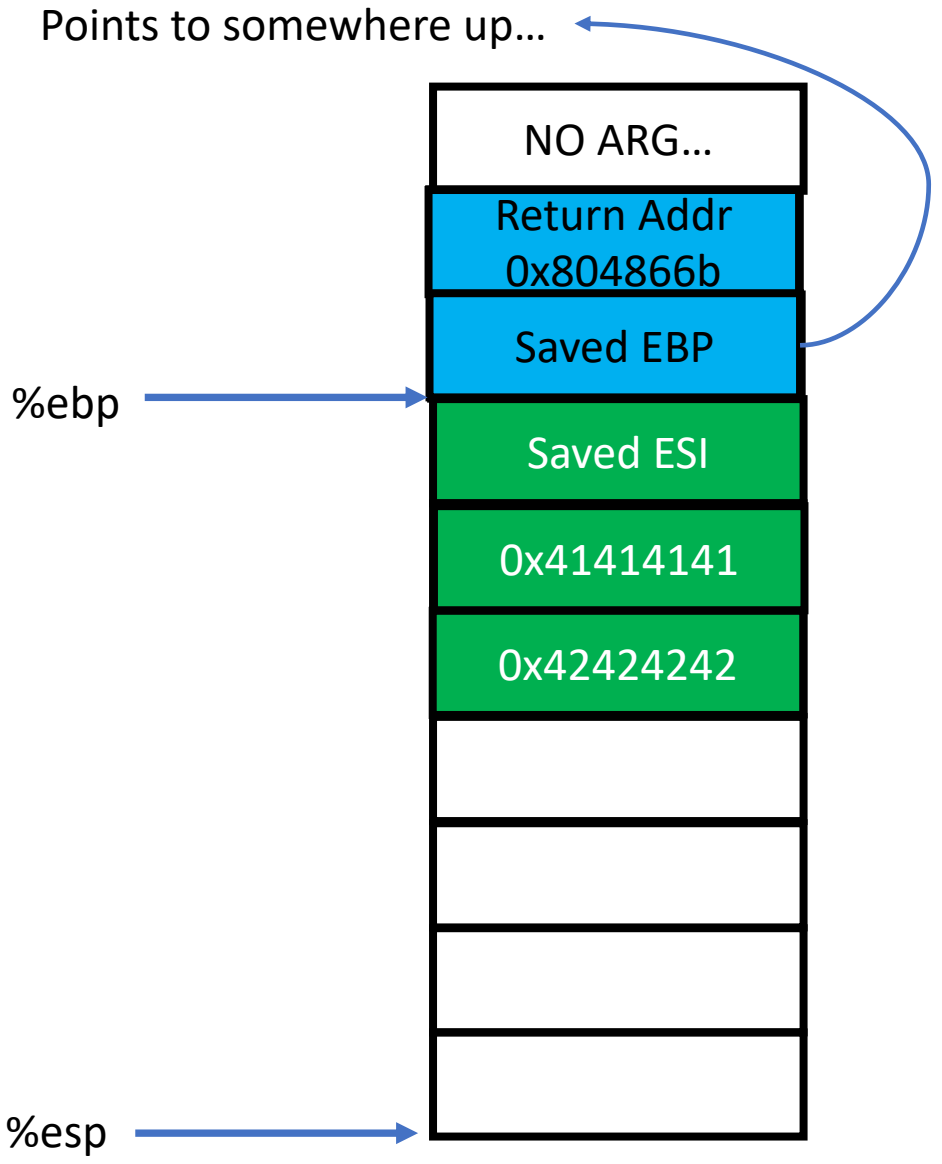
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

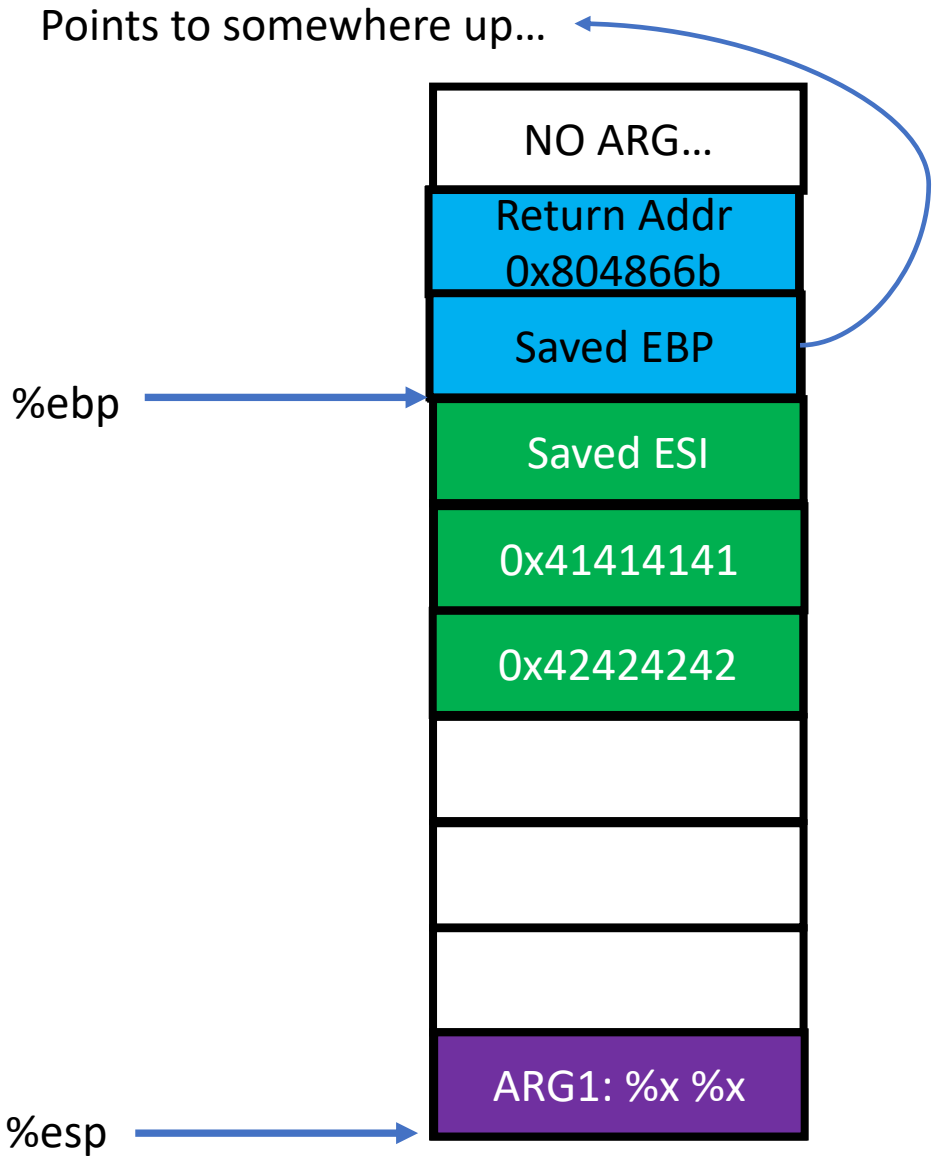
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

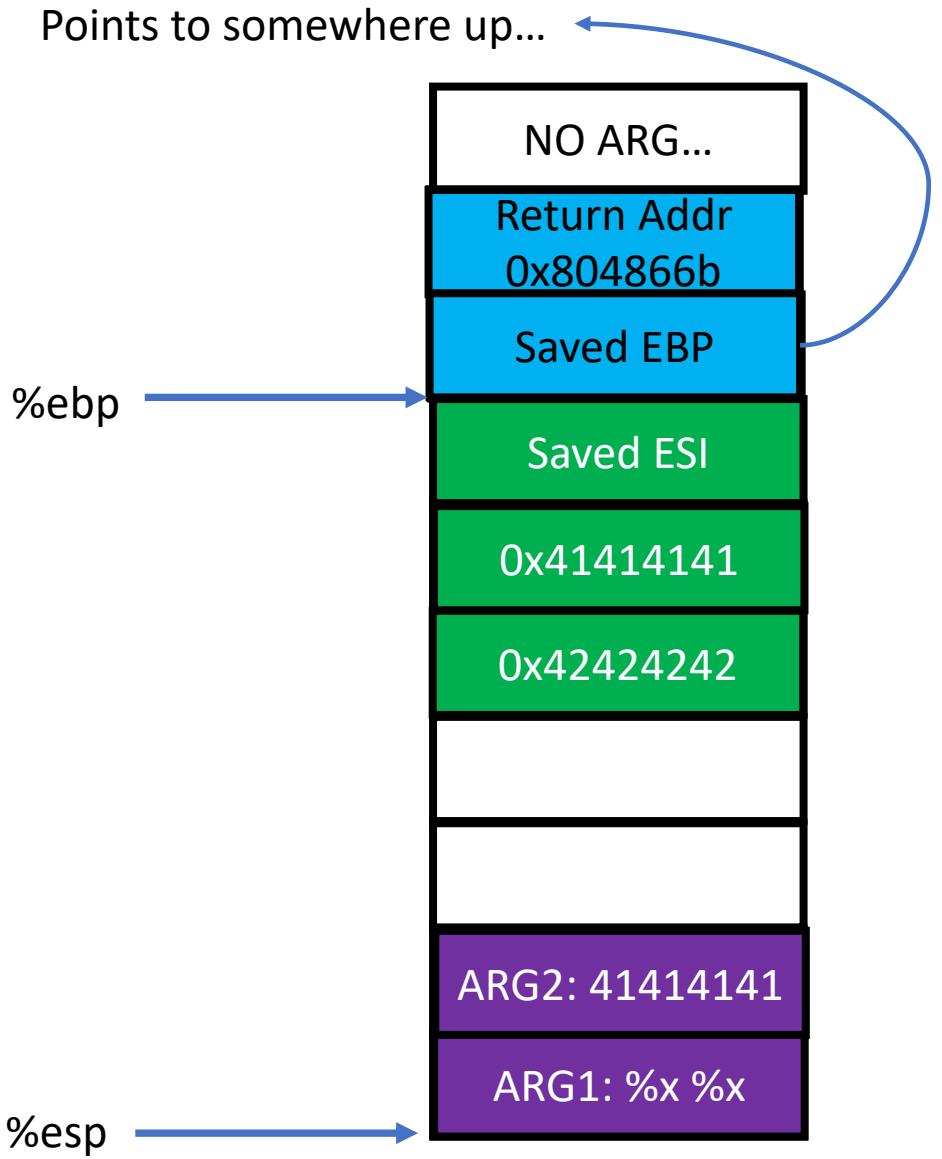
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

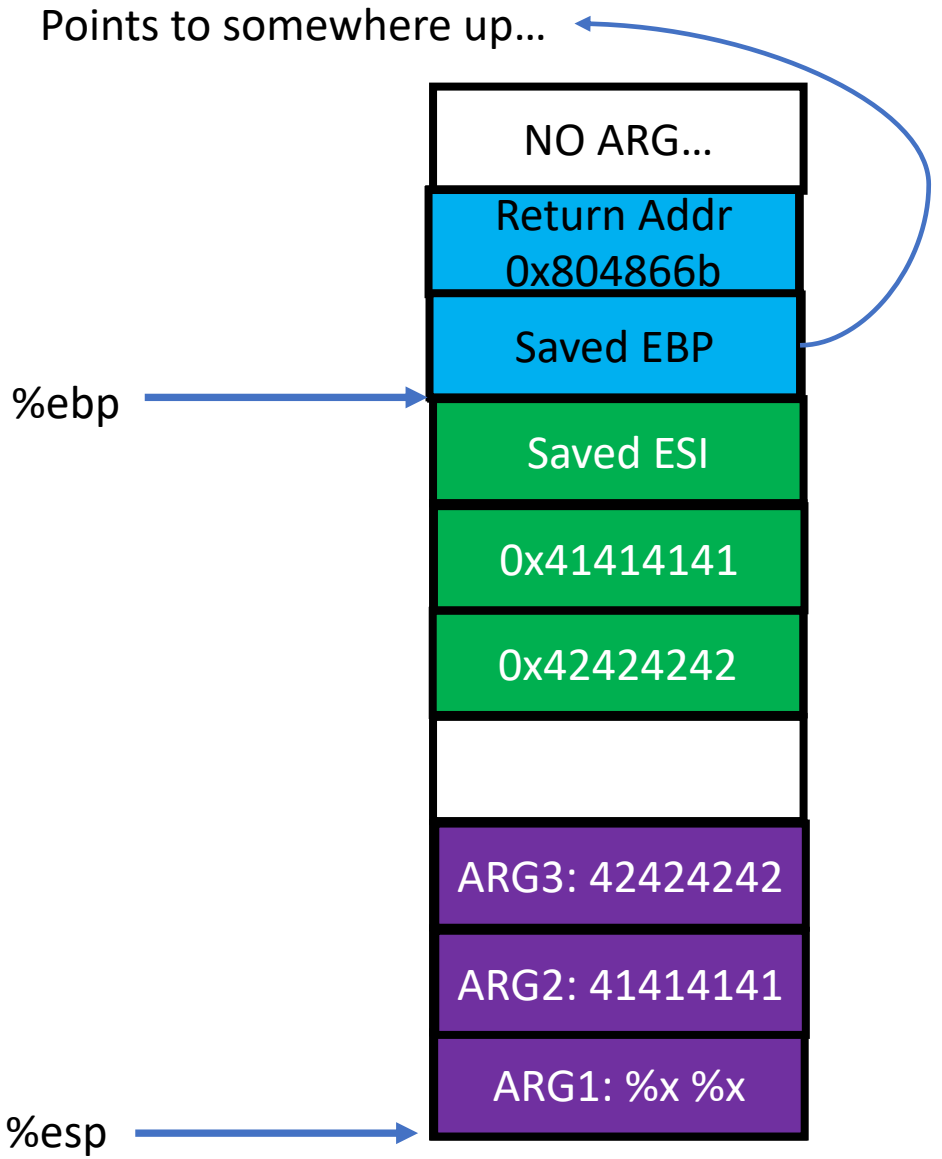
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

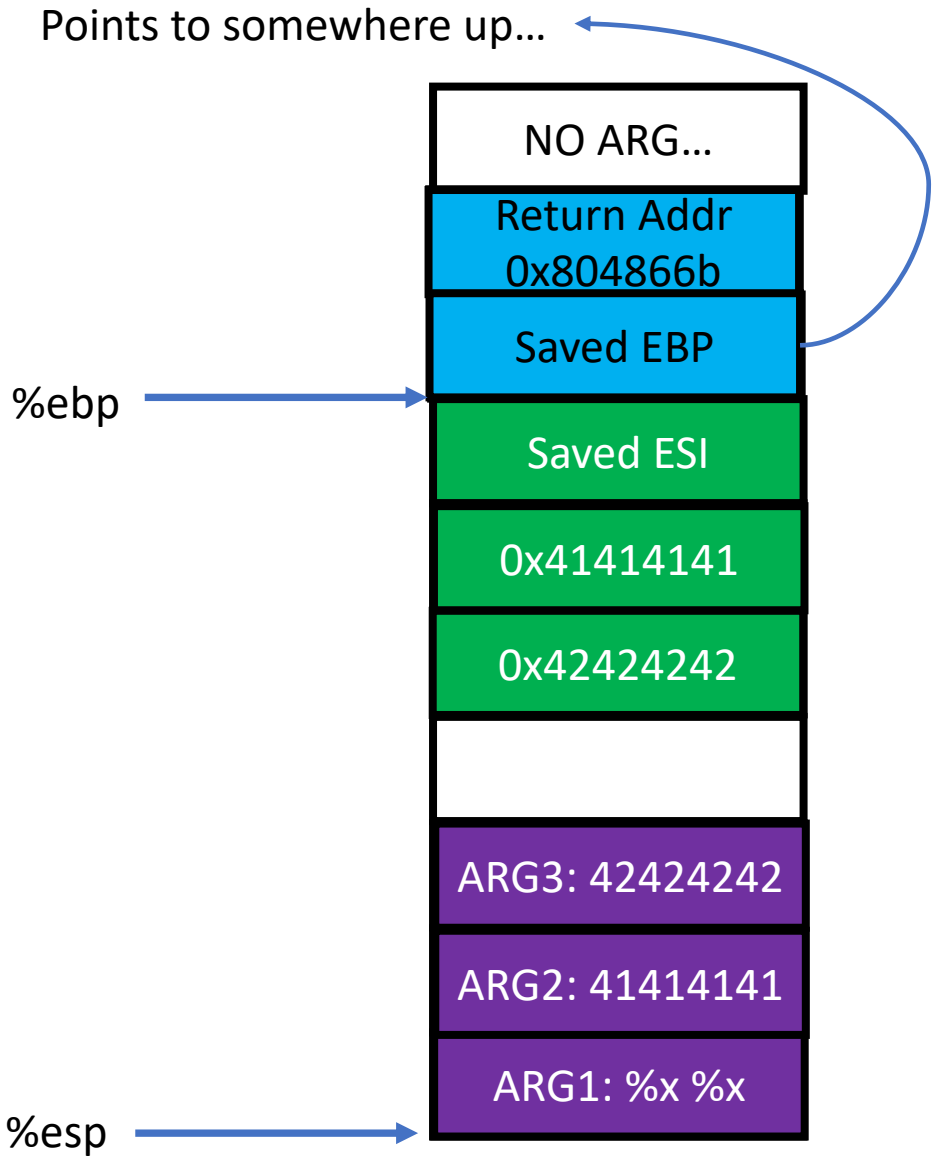
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

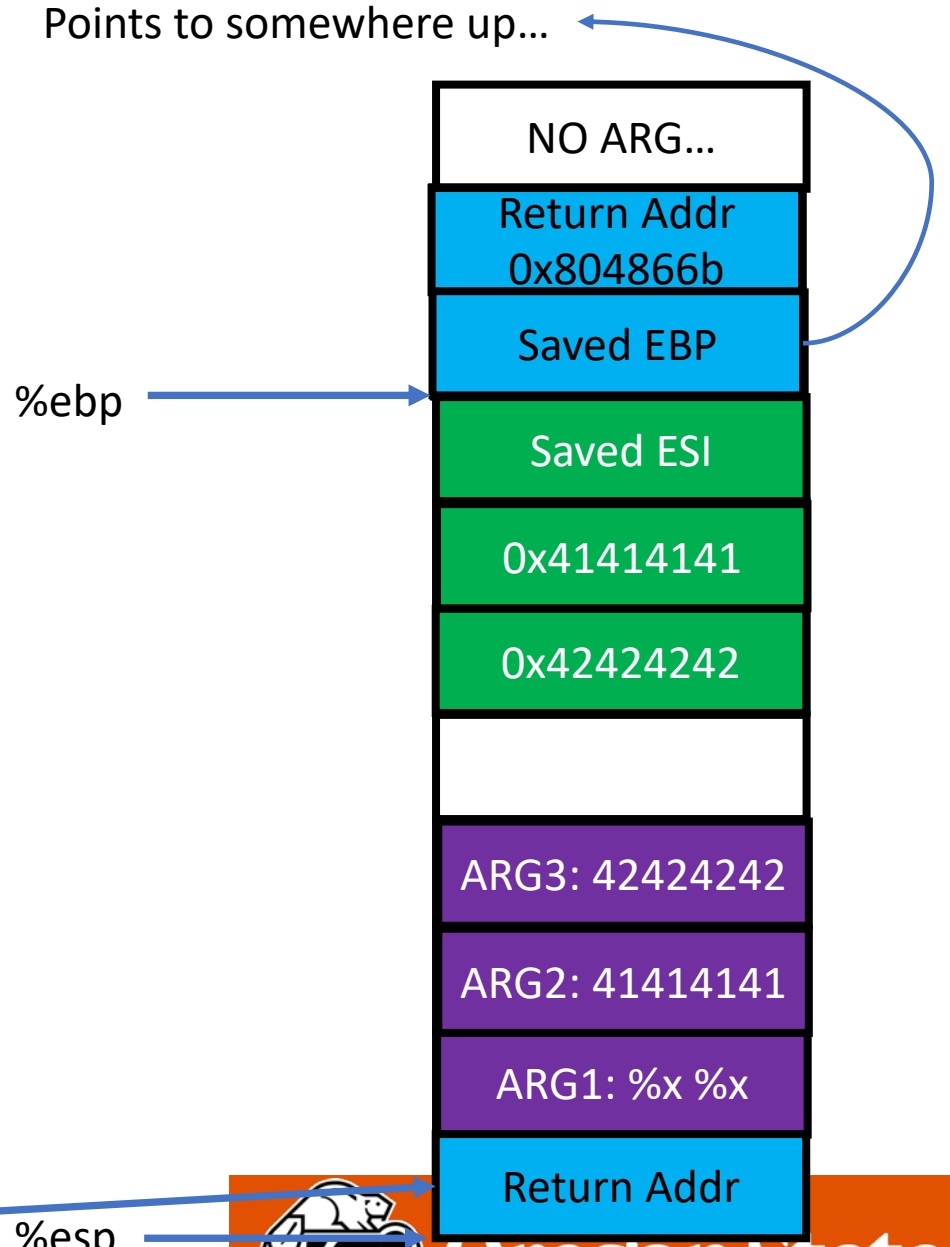
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
lea     0x8048765,%ecx
```



Example – Function call

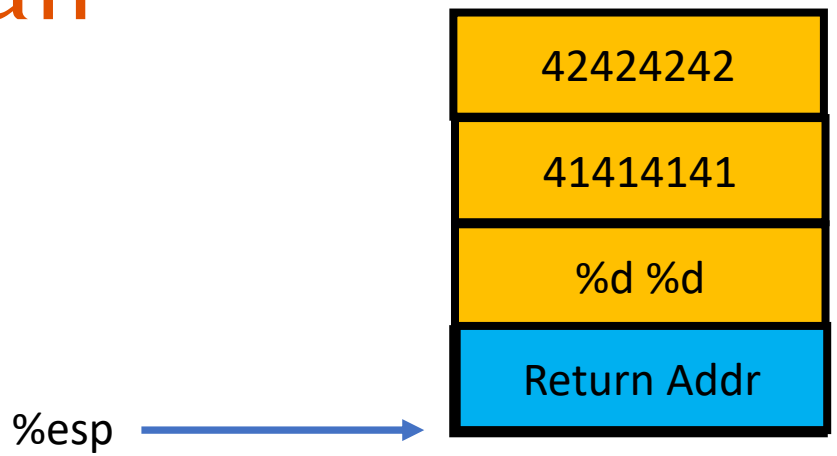
- Call printf?

```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
lea     0x8048765,%ecx
```



Example – Function call

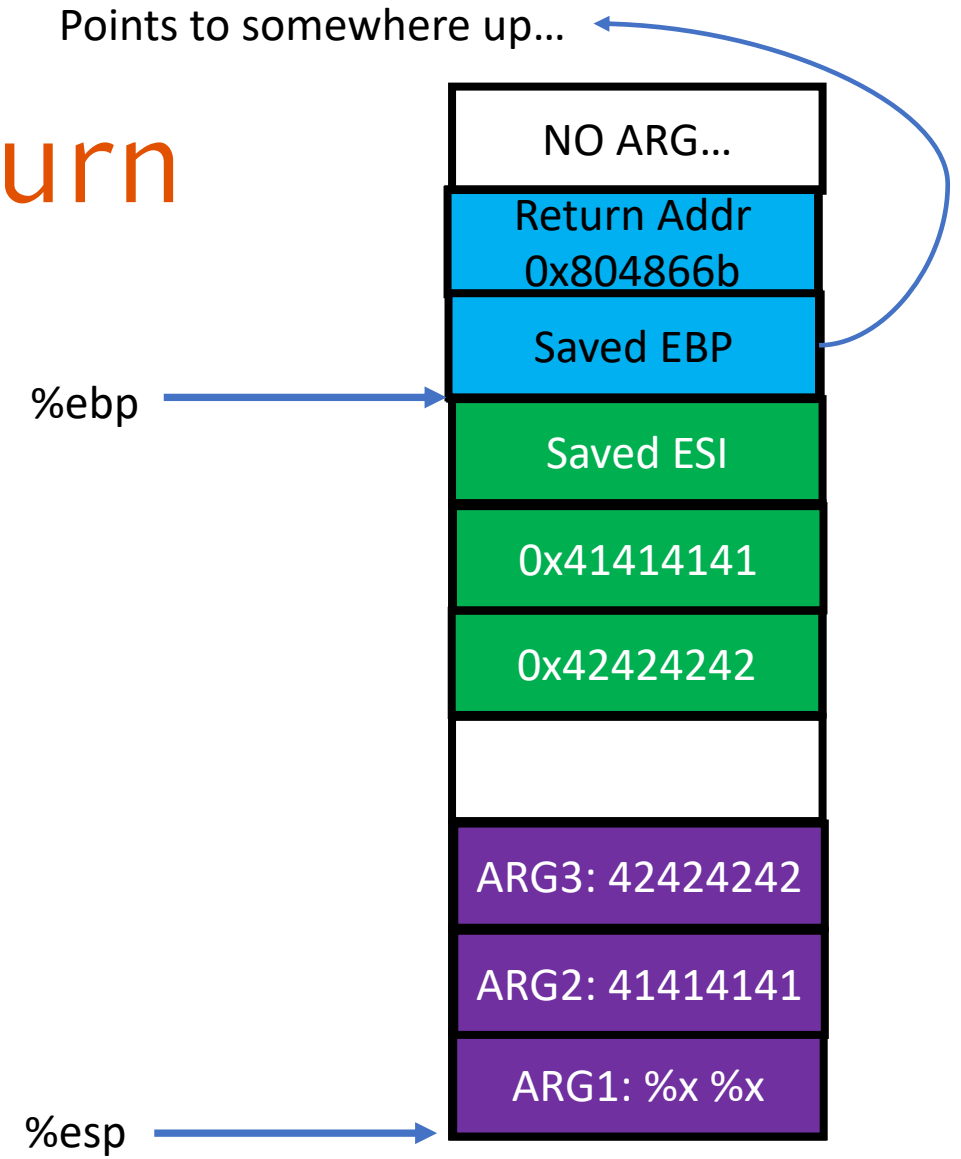
- What printf will see?



Example – Function Return

- Function return of receive_input

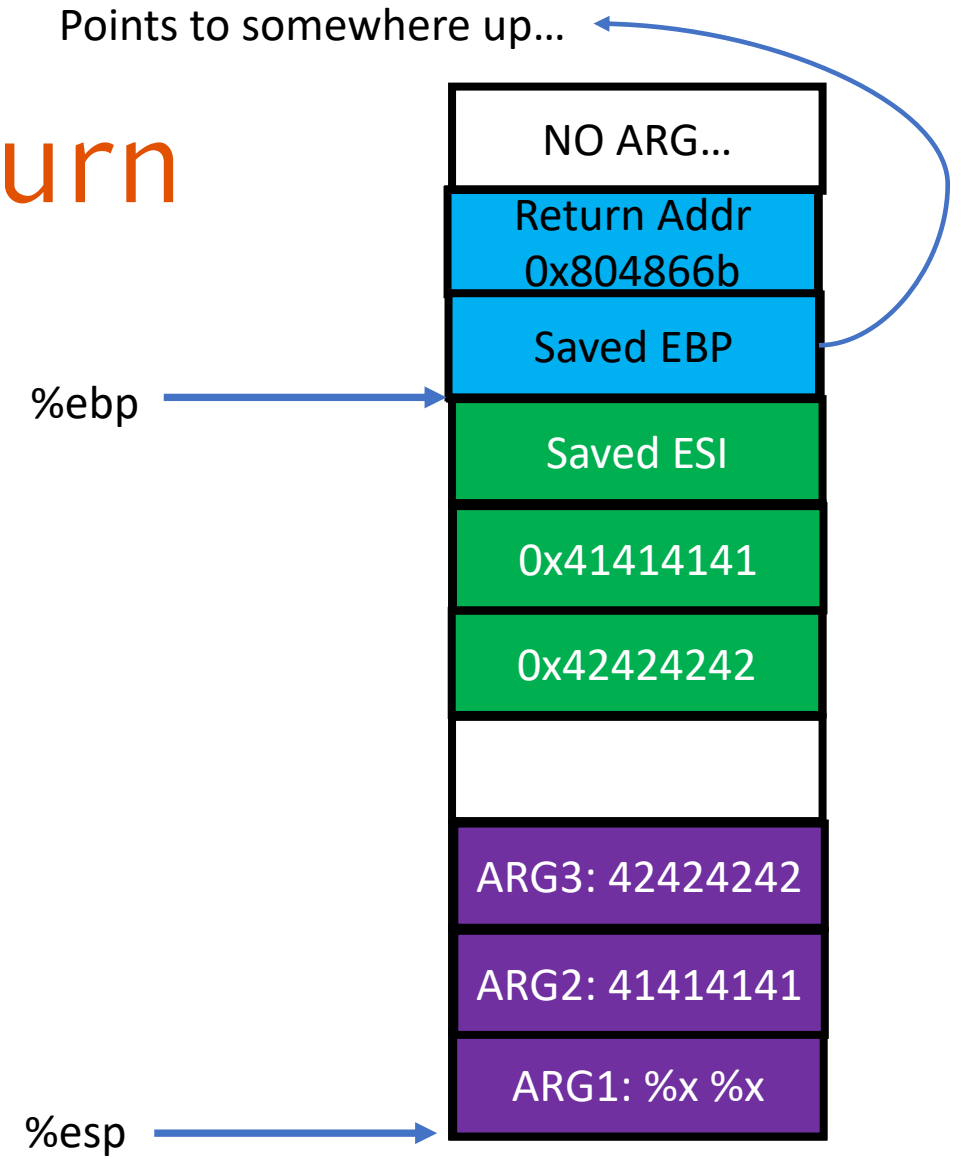
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

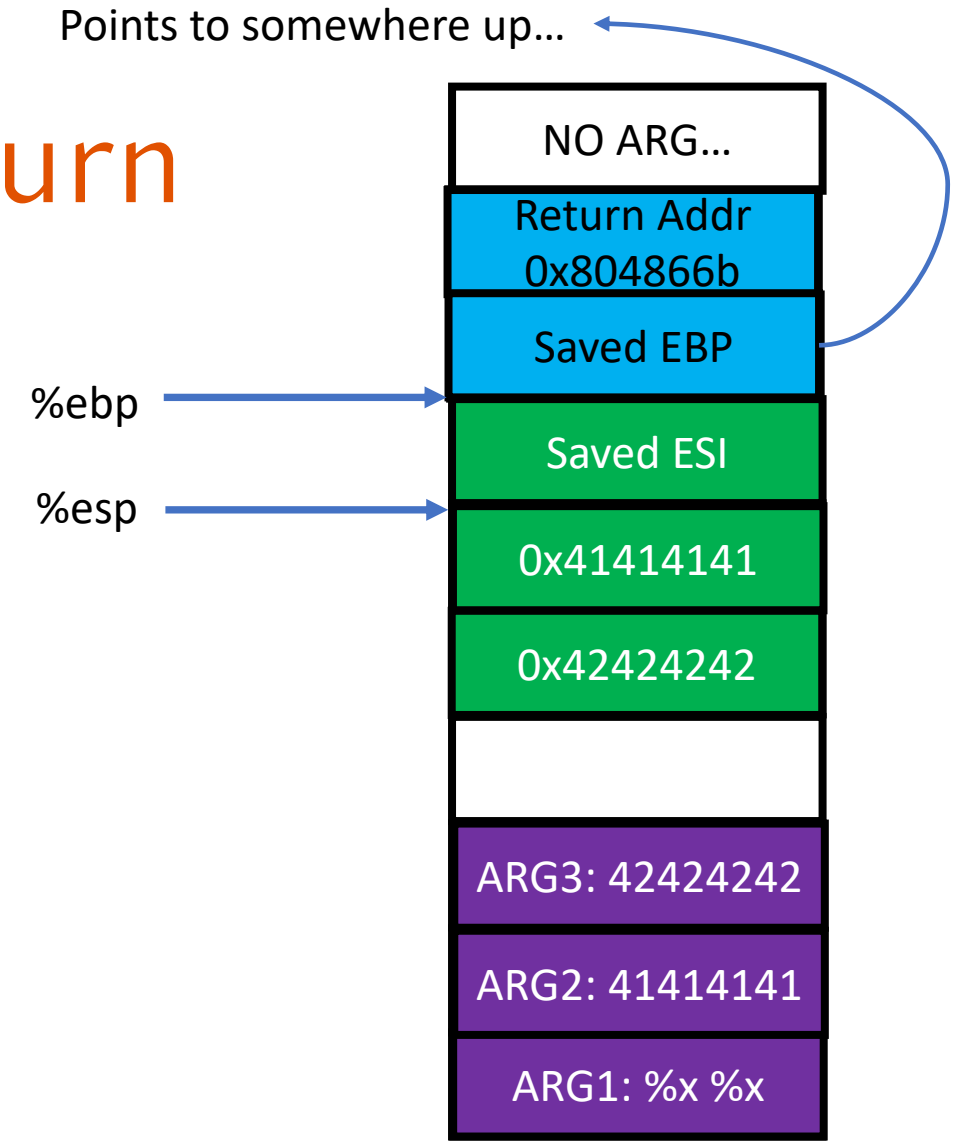
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

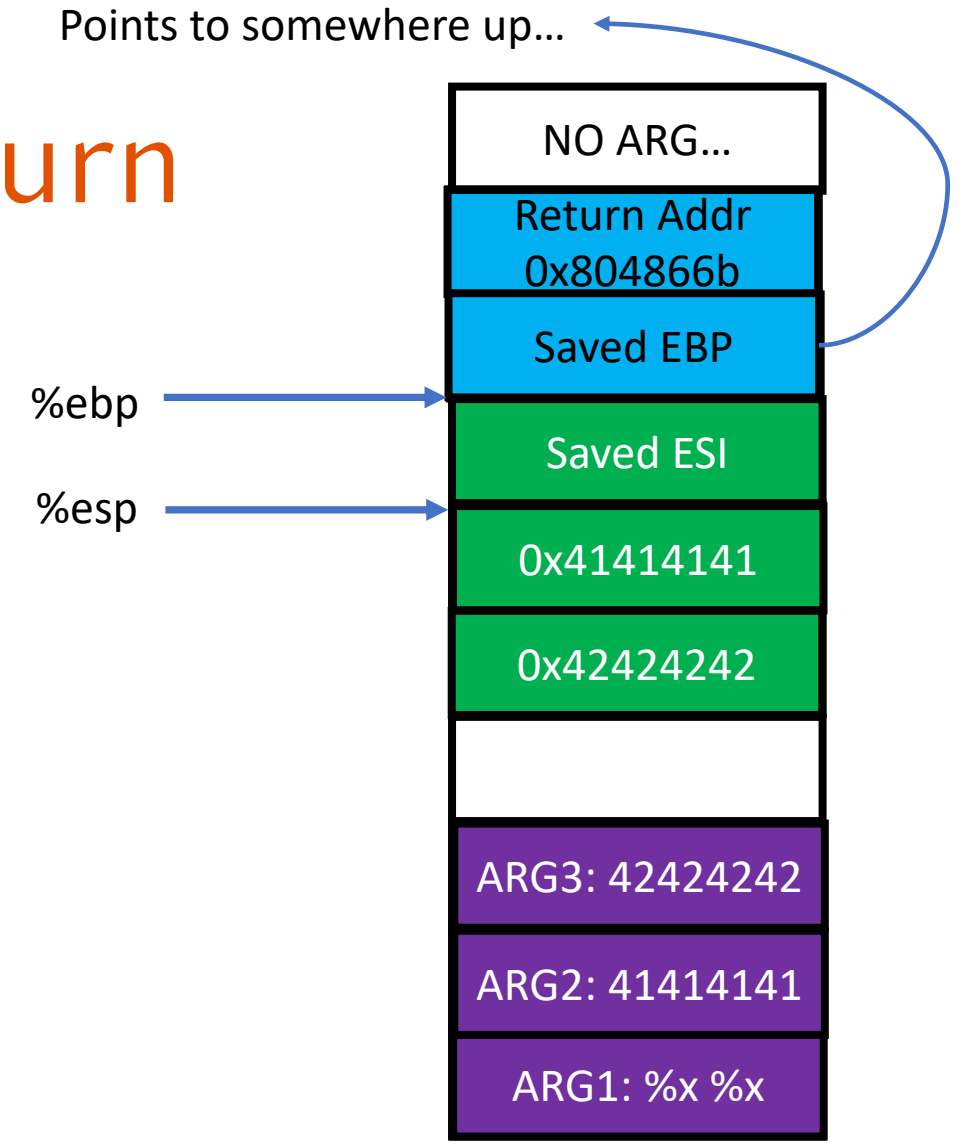
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

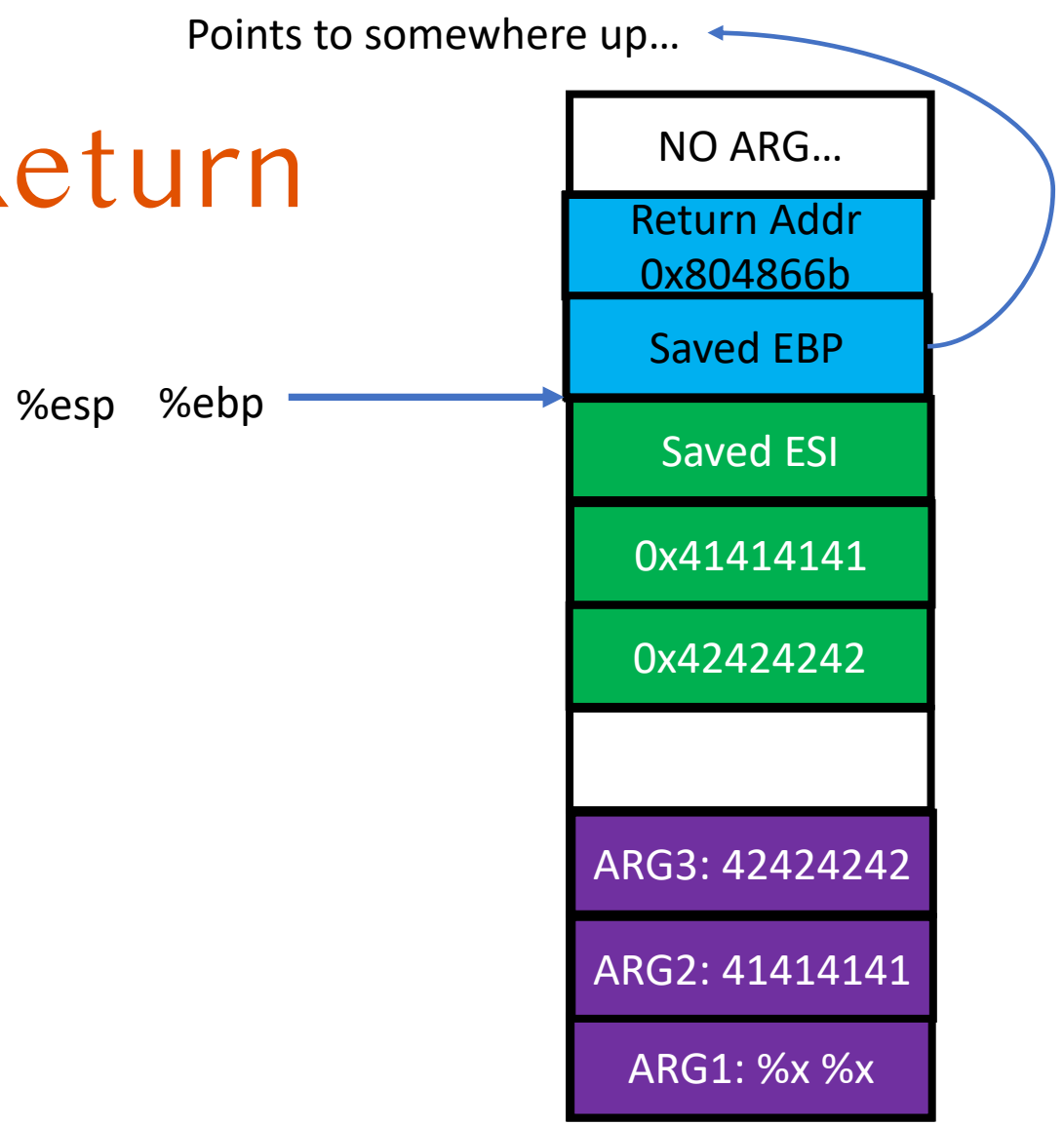
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

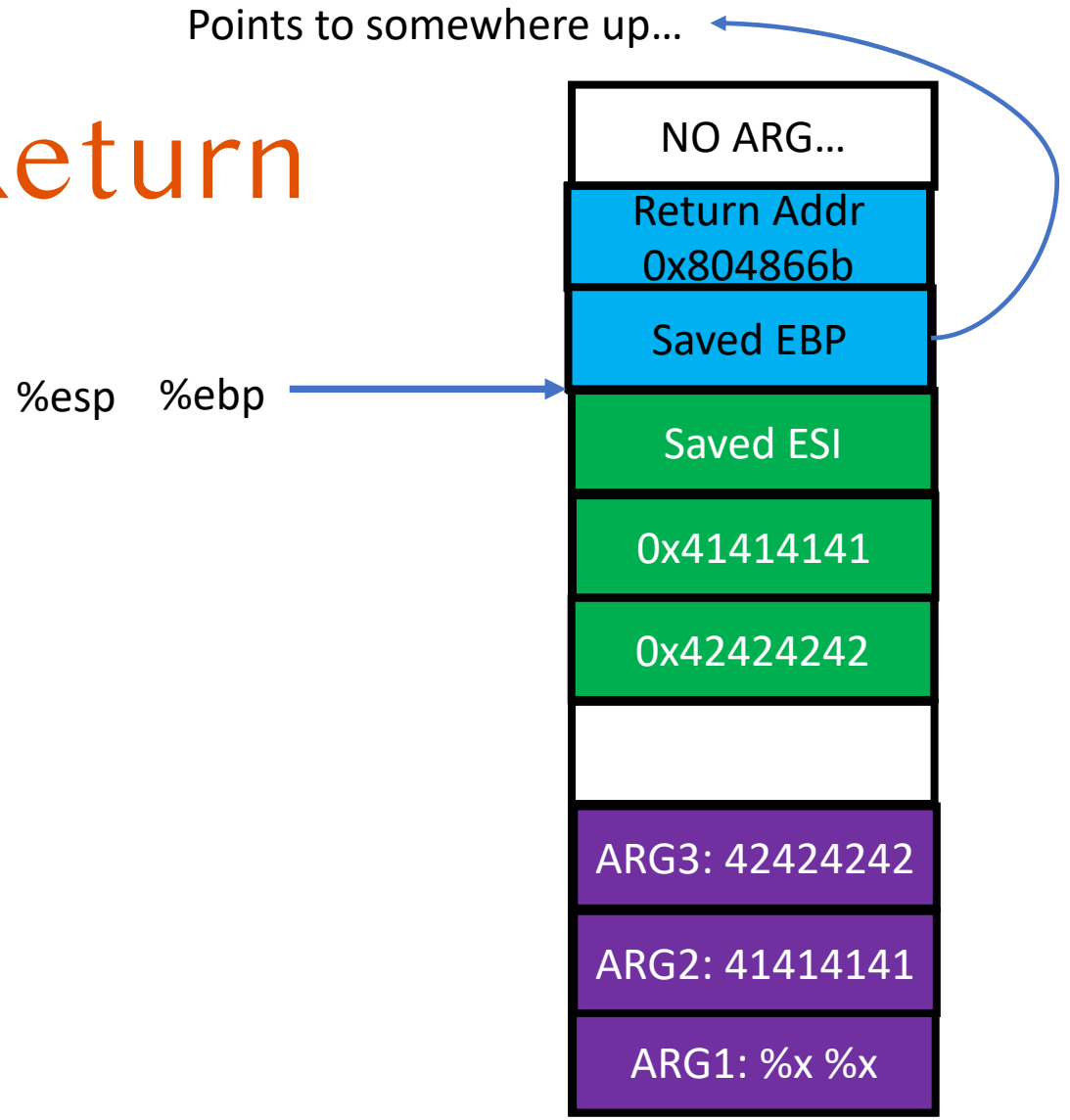
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

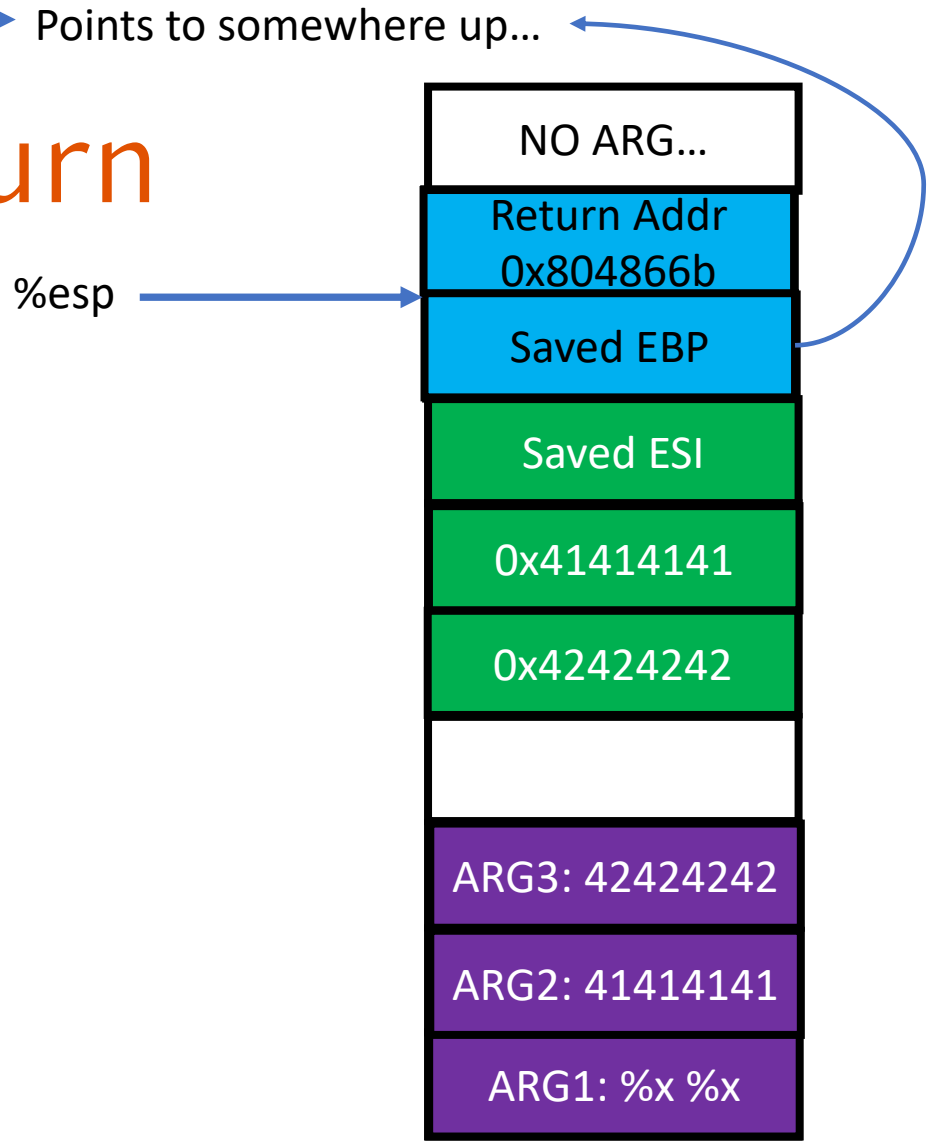
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



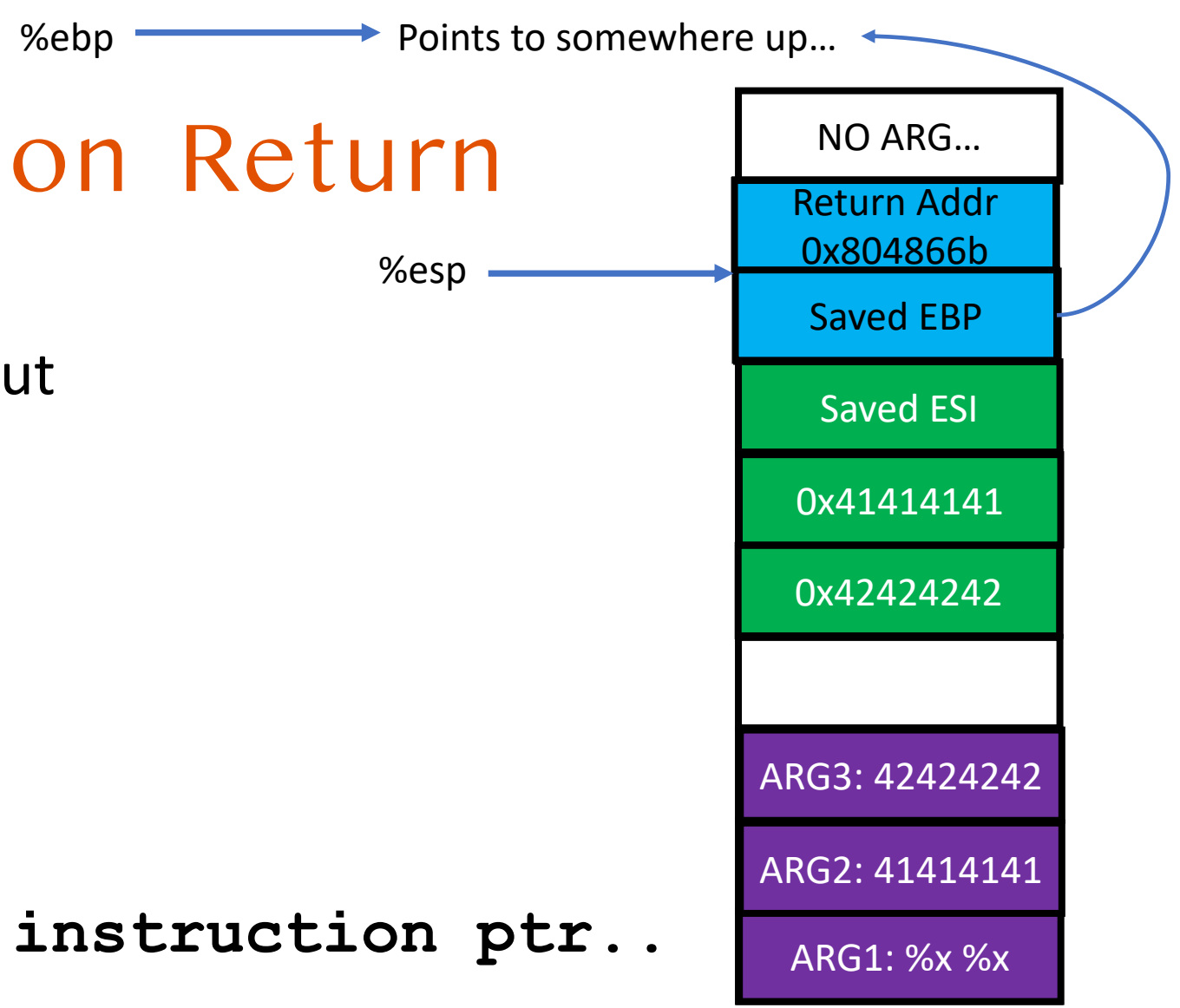
Example – Function Return

- Function return of receive_input

```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

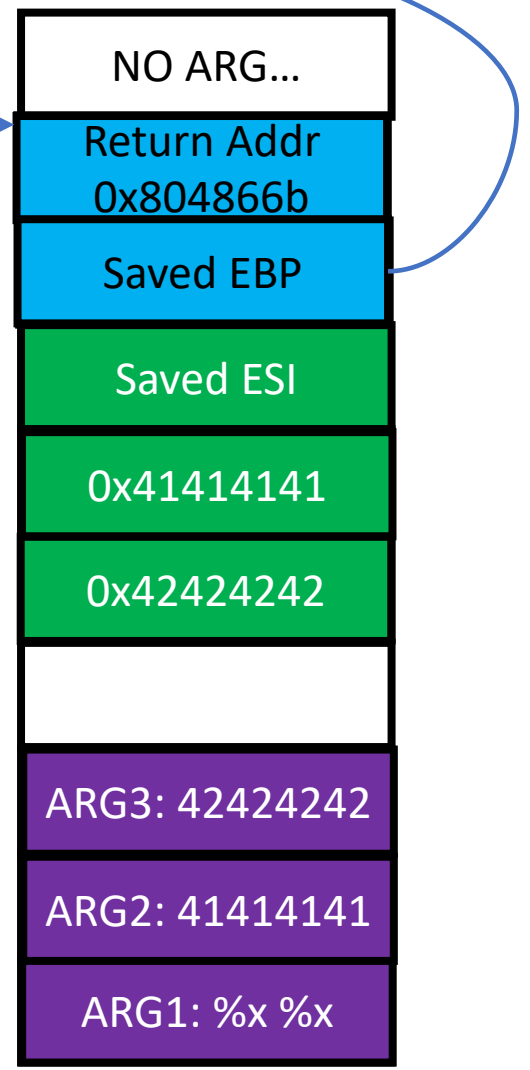


- Function return of receive_input

```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```

ret: pop %eip, change instruction ptr..

Example – Function Return



- Function return of receive_input

```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```

```
call   0x8048570 <receive_input>
0x0804866b <+11>:       xor    %eax,%eax
```

Buffer Overflow

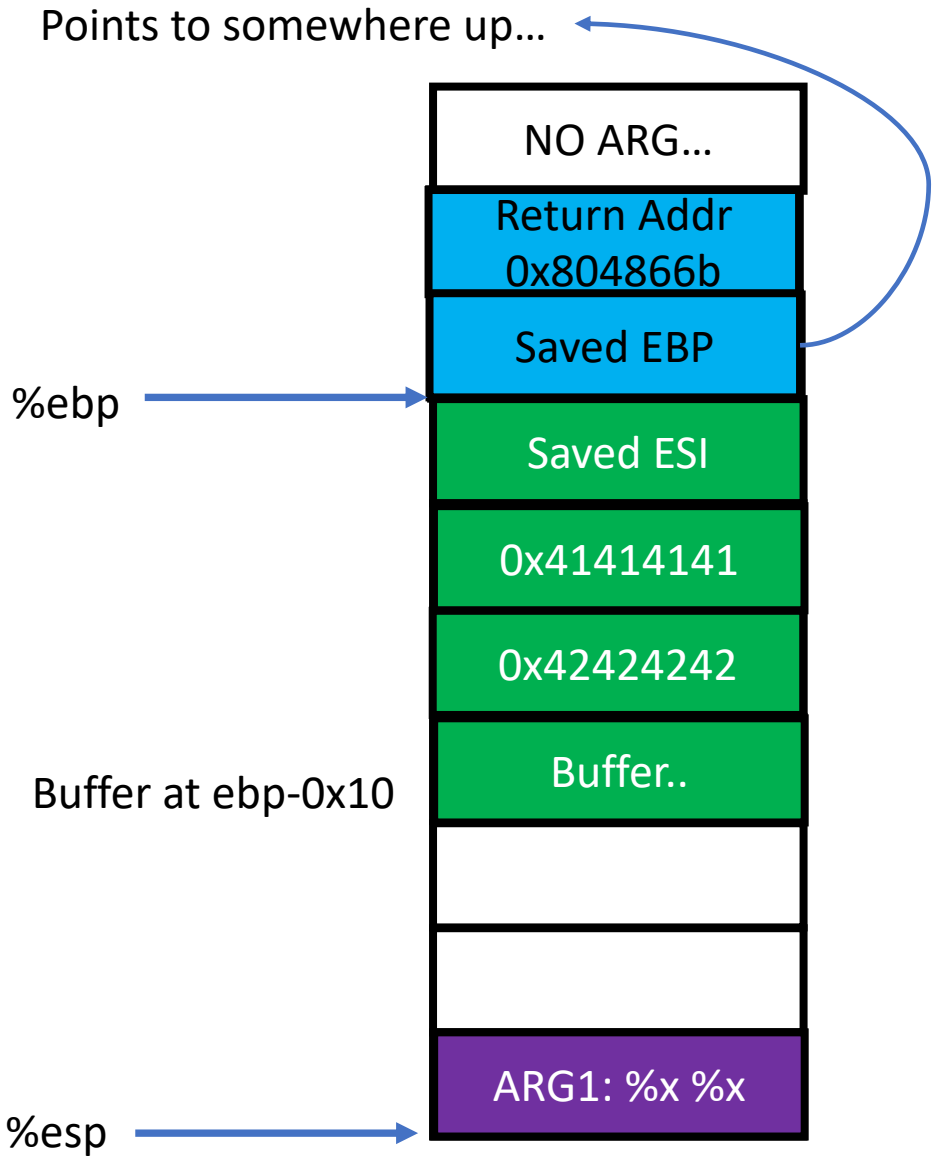
- Example

```
void receive_input() {
    int a = 0x41414141, b = 0x42424242;
    char buf[20];

    printf("Values in two local variables are:\n"
           "a = 0x%08x and b = 0x%08x\n", a, b);

    printf("Can you change these values to:\n"
           "a = 0x48474645 and b = 0x44434241?\n");

    printf("Type YES if you agree with this... "
           "(a fake message, you may overflow the input buffer).\n");
    fgets(buf, 128, stdin);
```

- char buf[20];
- fgets(buf, 128, stdin);

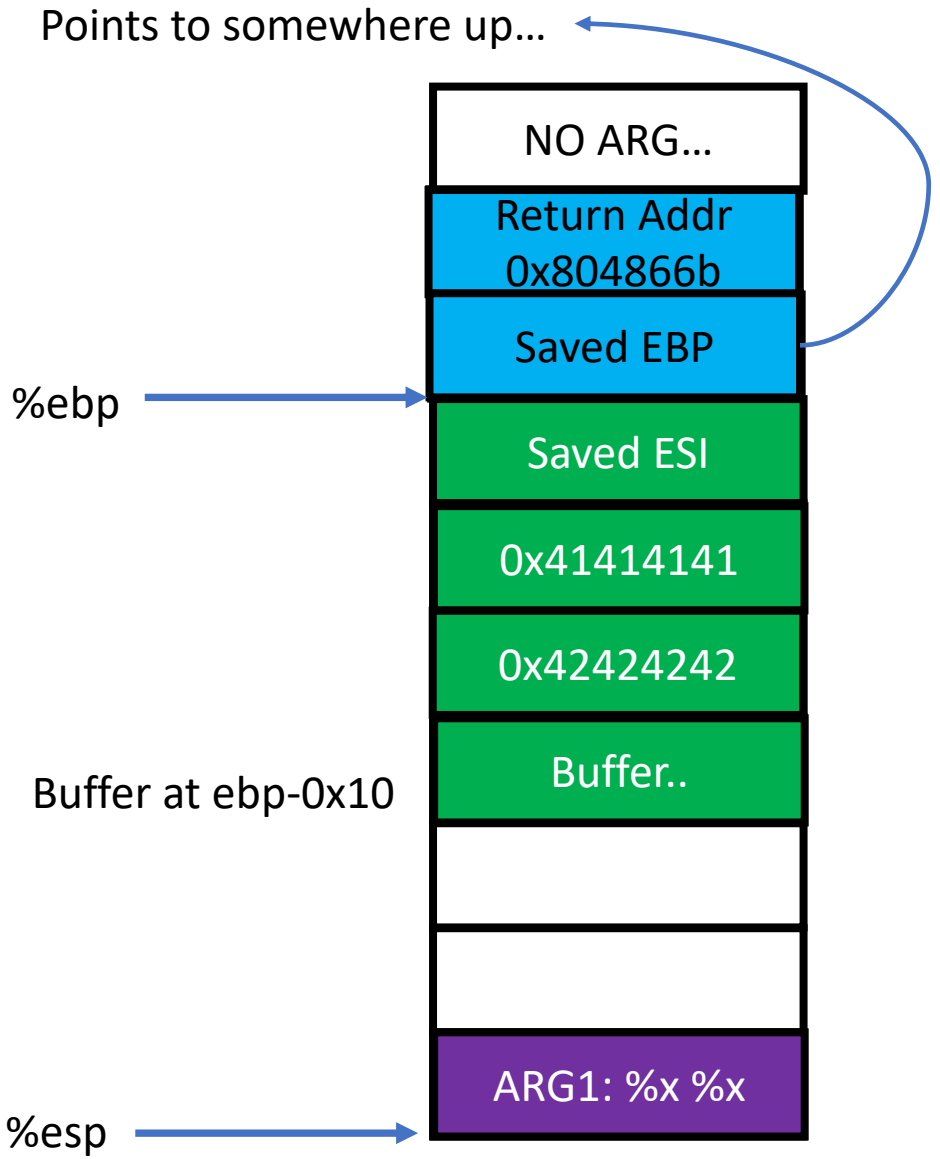
Buffer Overflow

- Overwrite values in stack by overflowing
  - A local variable buffer
- Suppose we have char buffer[4];
  - -0x8(%ebp) stores 0x41414141
  - -0xc(%ebp) stroes 0x42424242
  - **-0x10(%ebp) is a buffer, size 4 byte.**
- Program gets input from you via
  - `fgets(buffer, 128, stdin);`
  - Read **128** bytes..
- What if you type “1111aaaabbbb”?



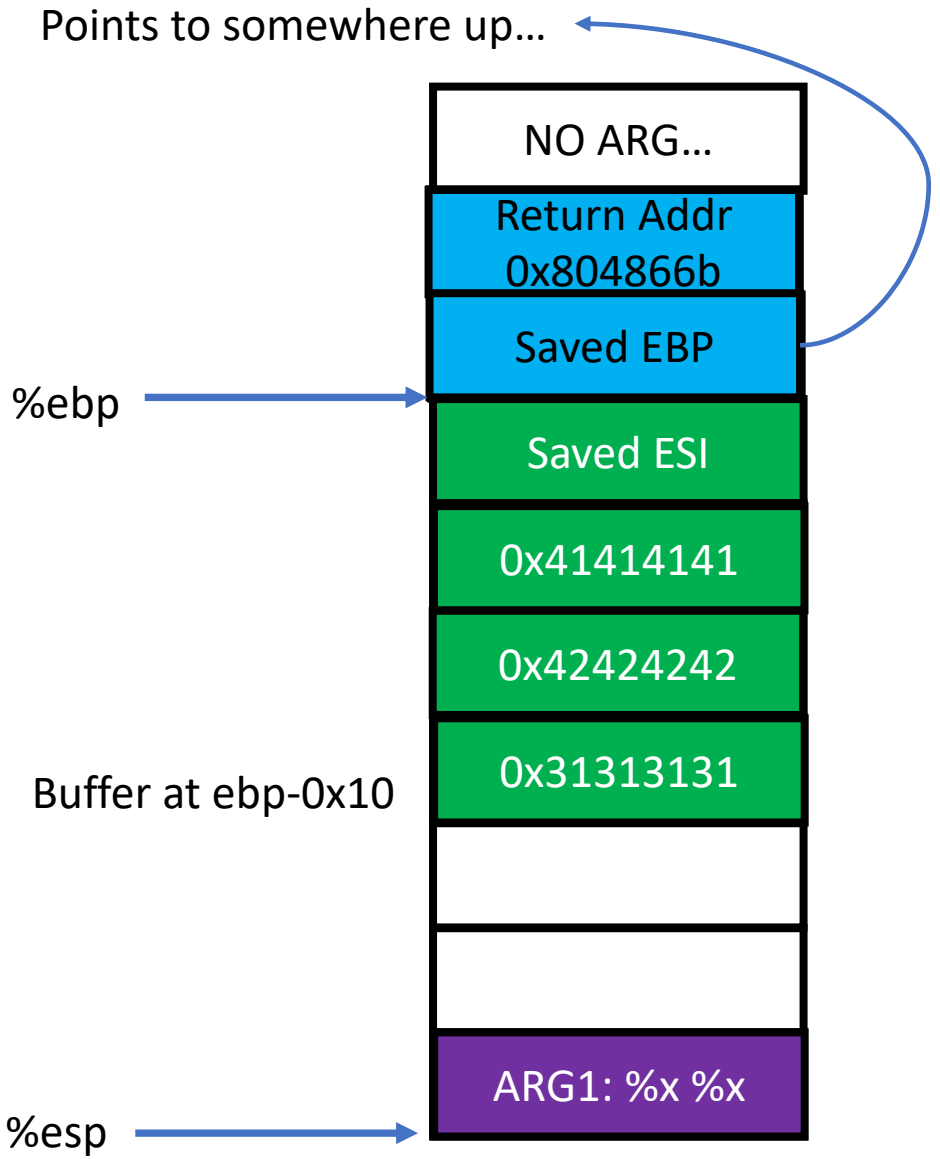
Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbb”?



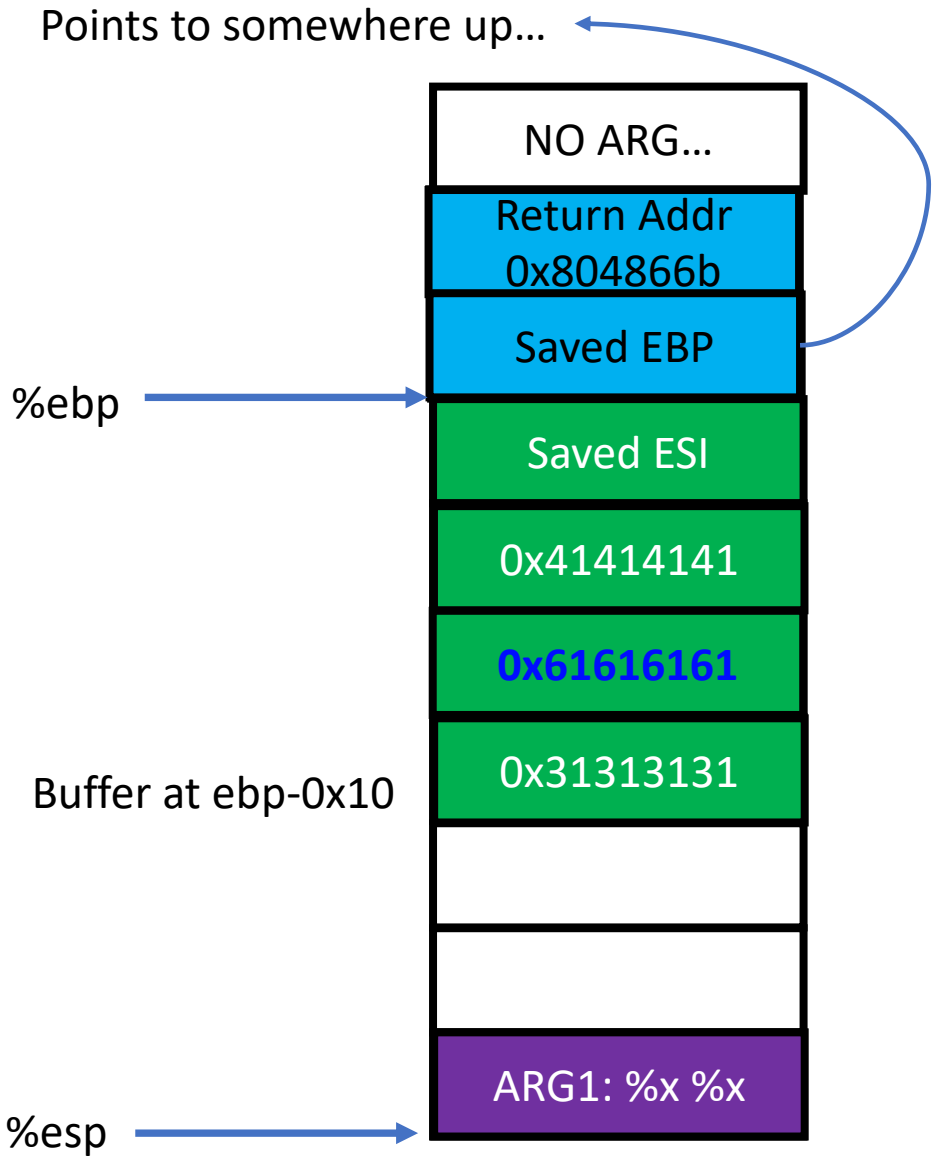
Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbb”?



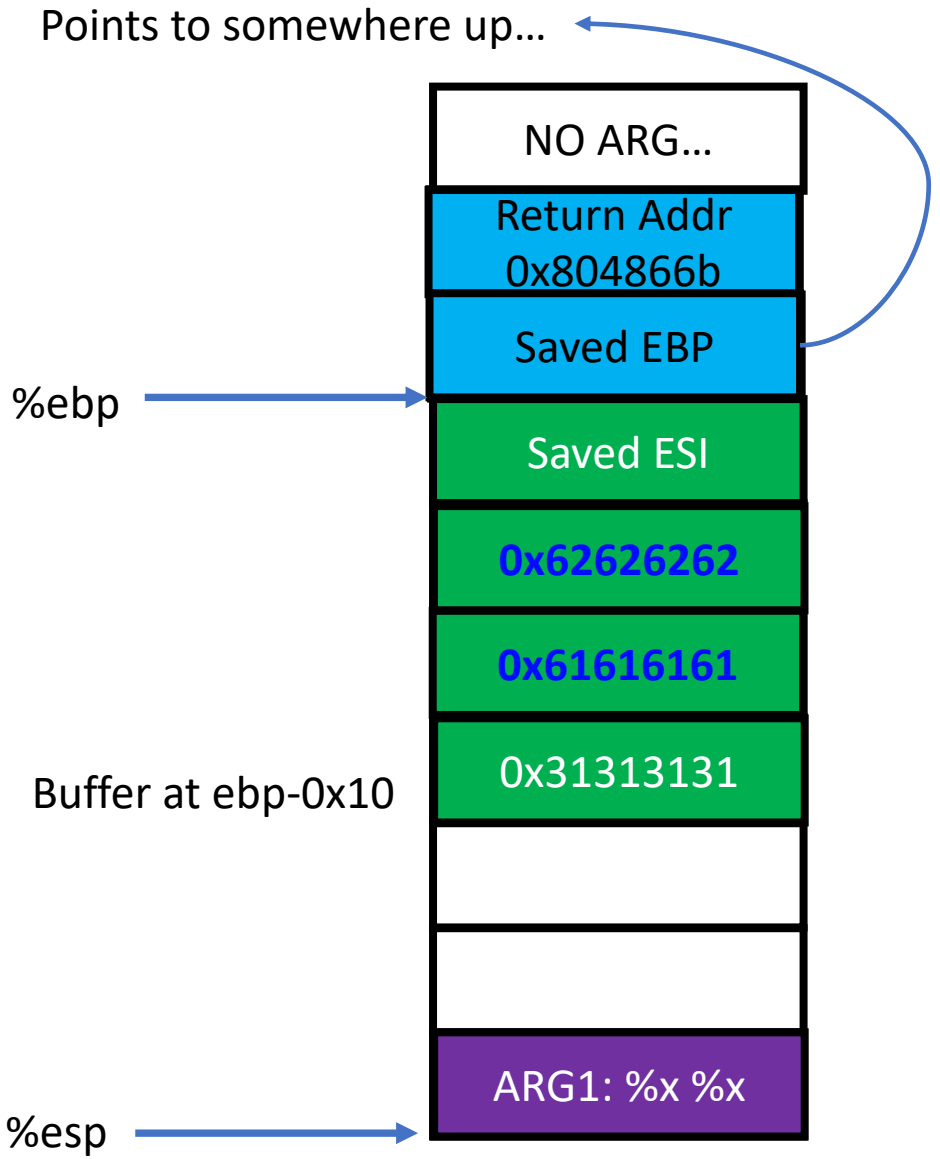
Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbb”?



Buffer Overflow

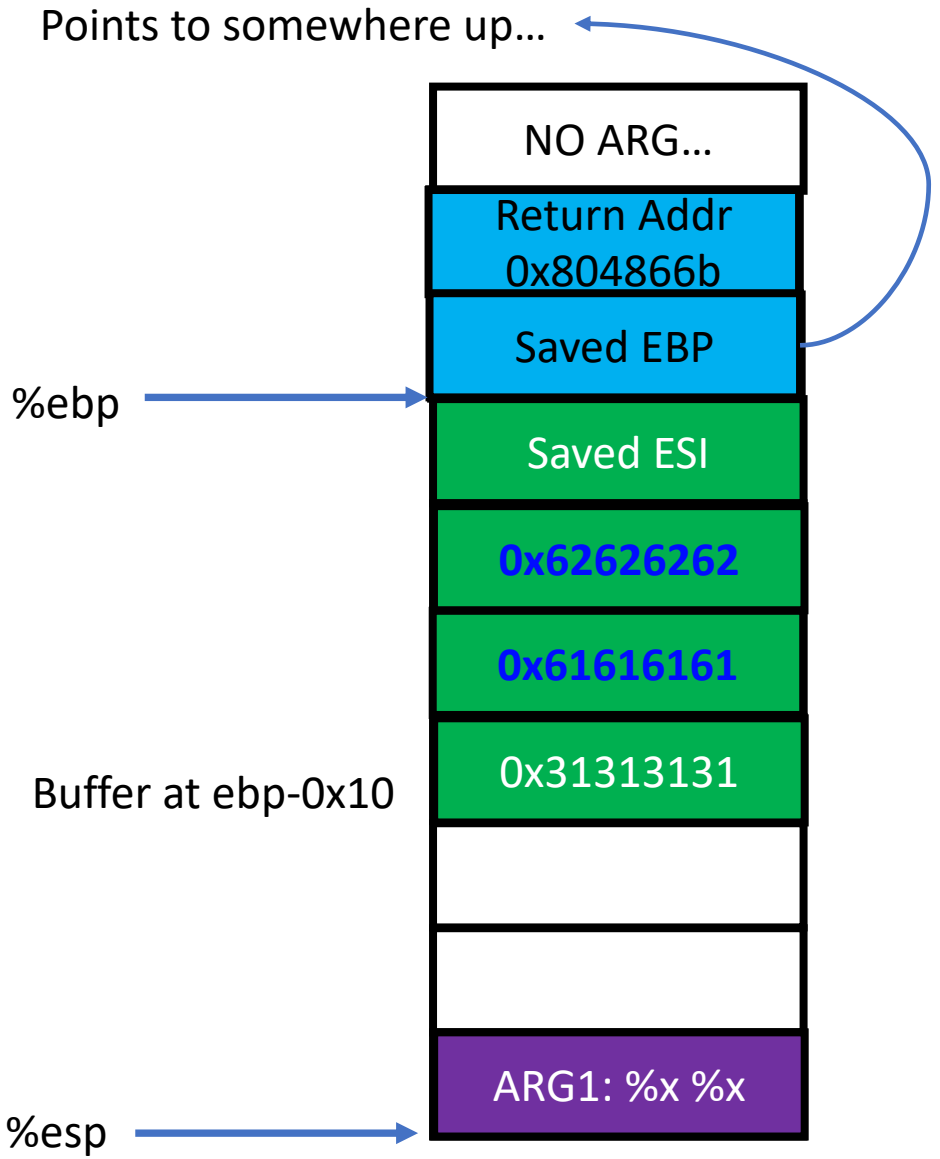
- 4 byte buffer…
- What if you type “1111aaaabbbb”?



Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbb”?

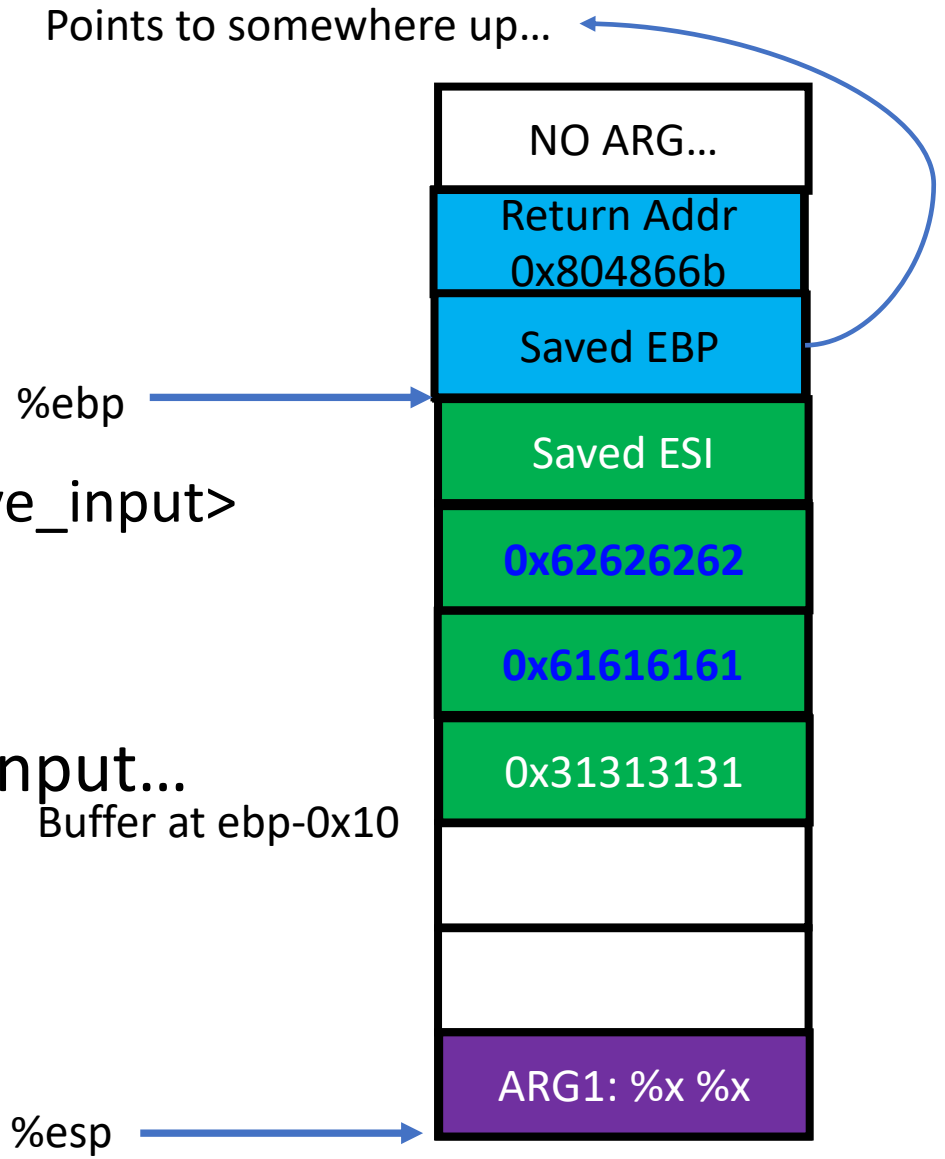
- You can change variables!



Return Address

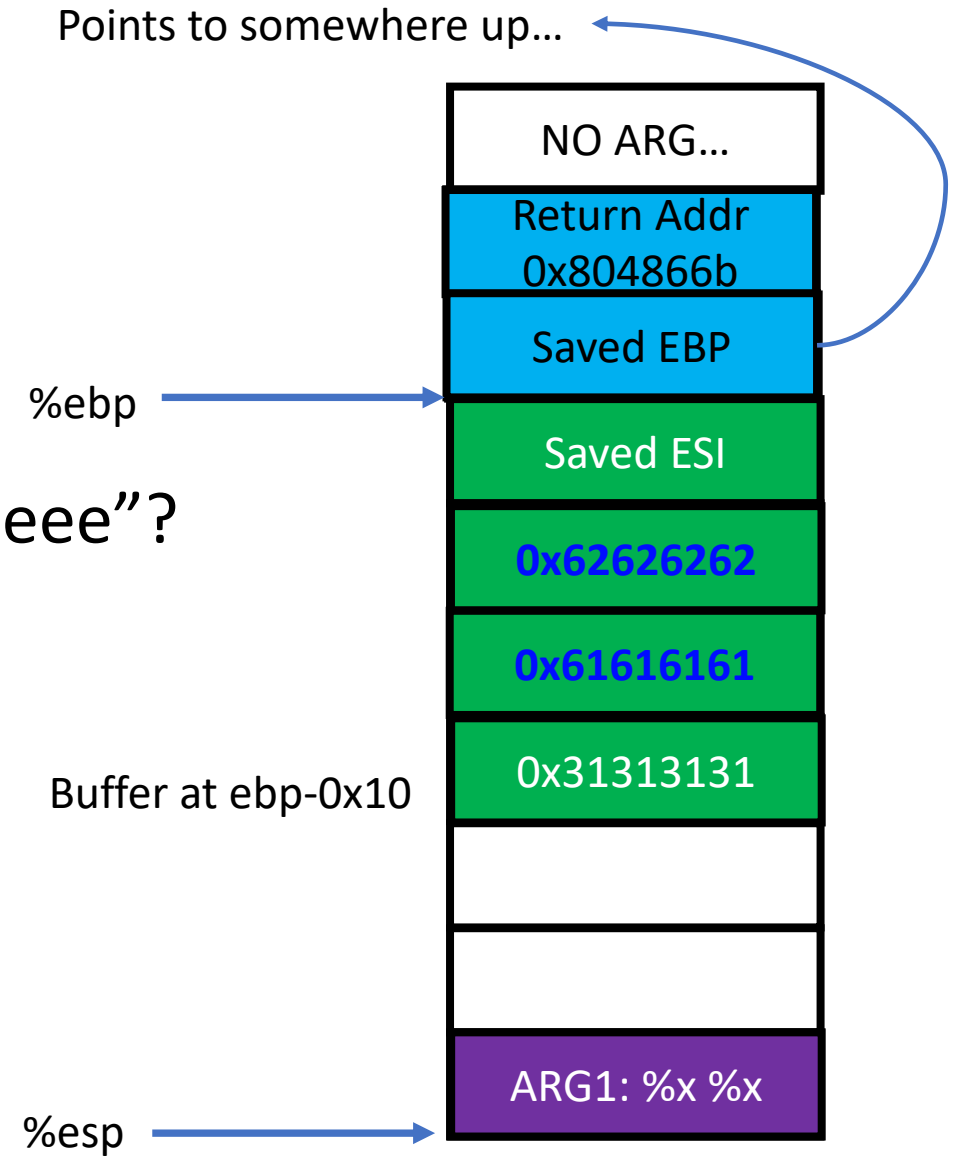
- Store the execution points after the call
  - 0x08048666 <+6>: call 0x8048570 <receive_input>
  - 0x0804866b <+11>: xor %eax,%eax

- Should store 0x804866b on calling receive_input…



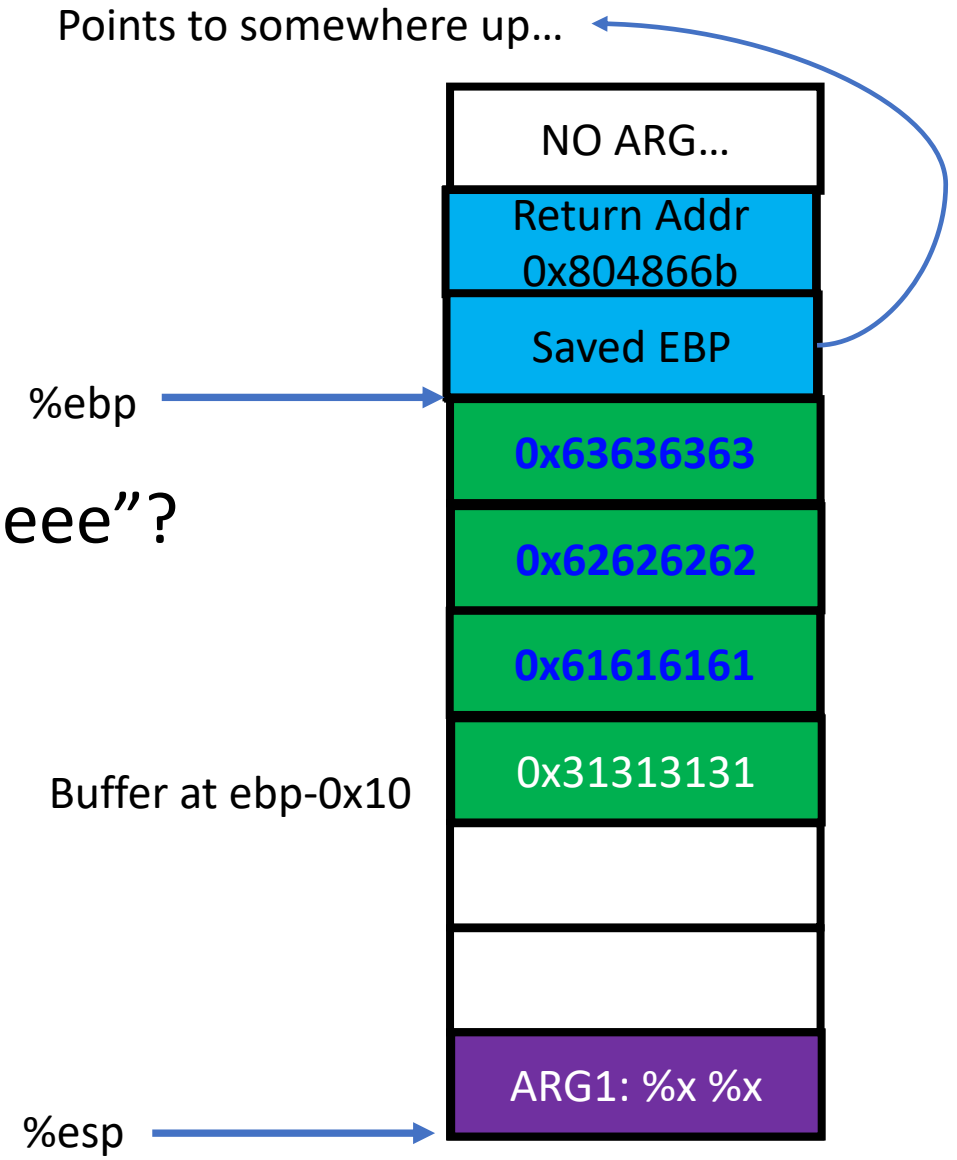
Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?



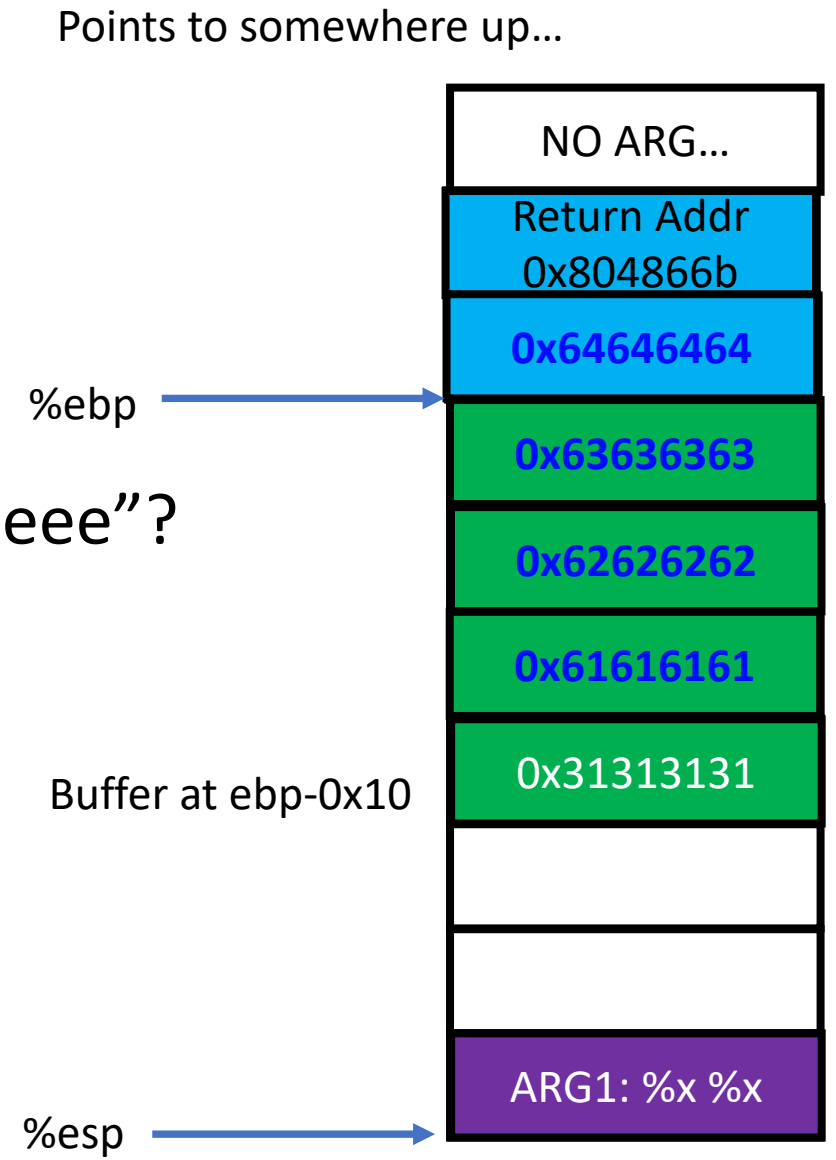
Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?



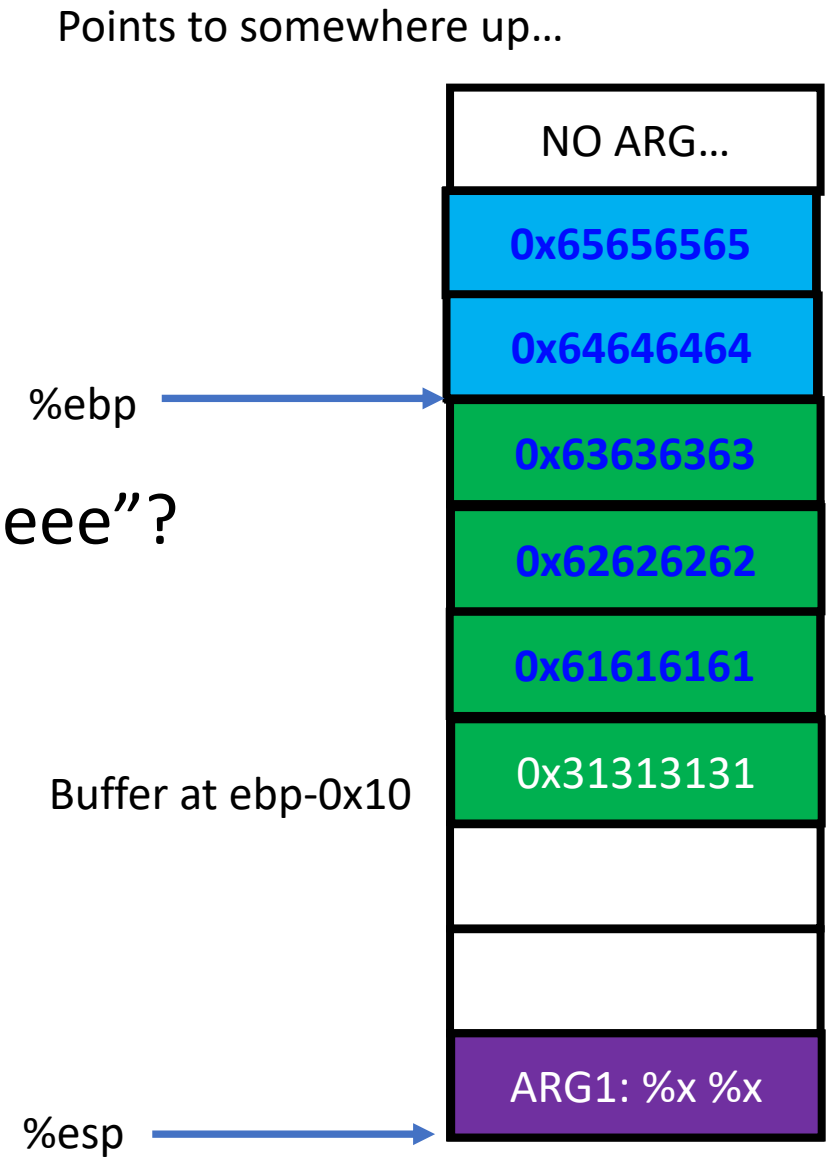
Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?



Buffer Overflow

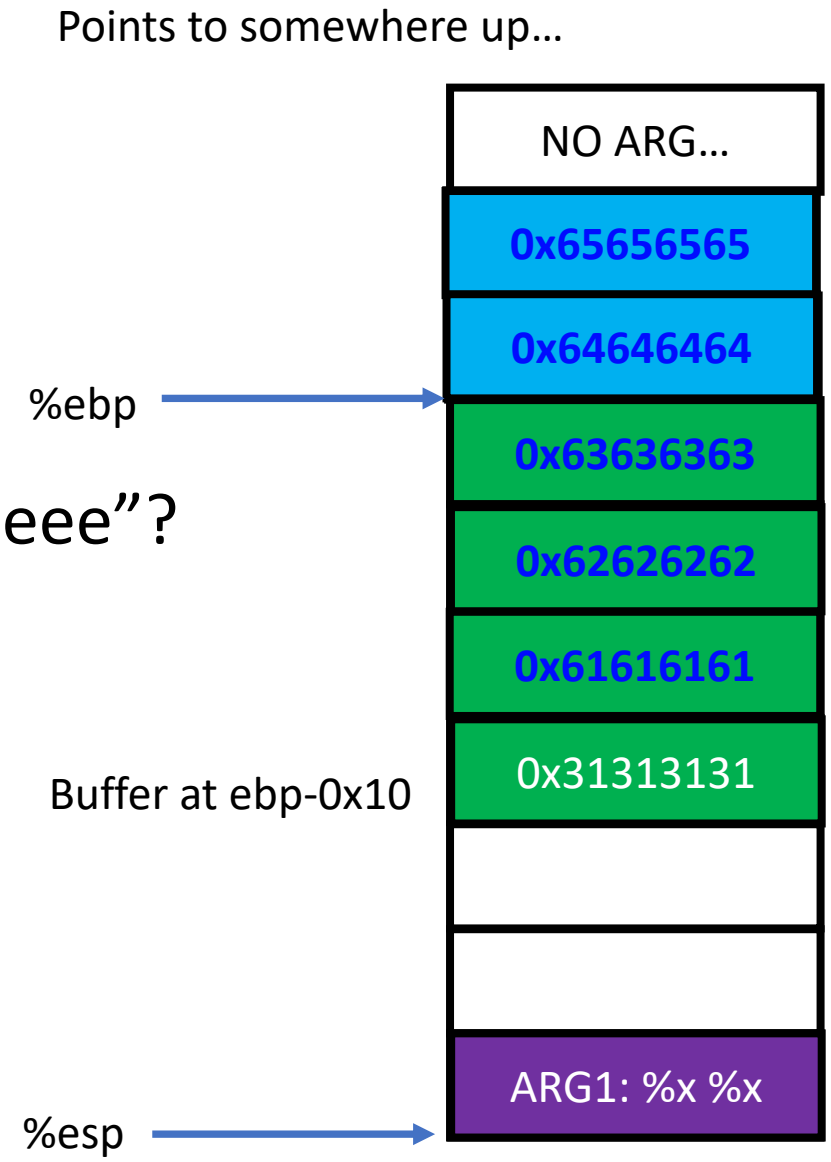
- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?



Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?

- Overwrites the return address!

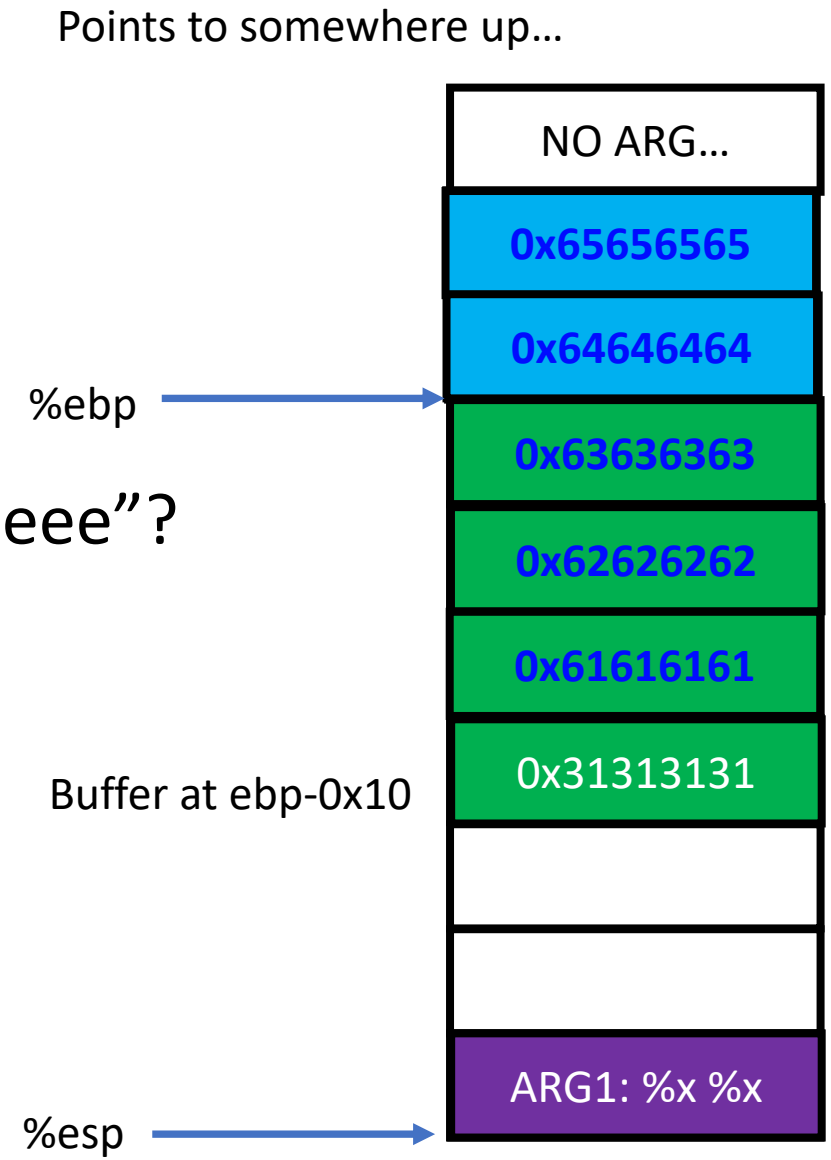


Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?

- Overwrites the return address!

- Can you set that as the address of
  - get_a_shell()?

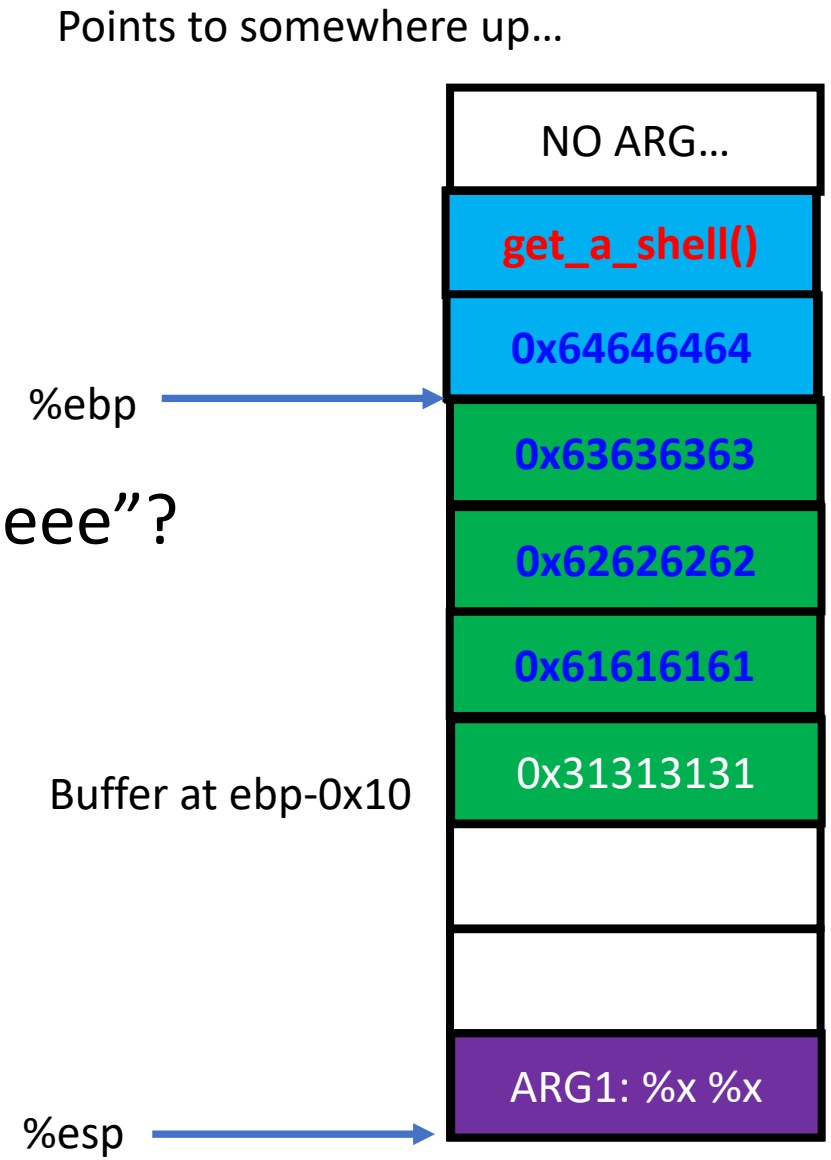


Buffer Overflow

- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?

- Overwrites the return address!

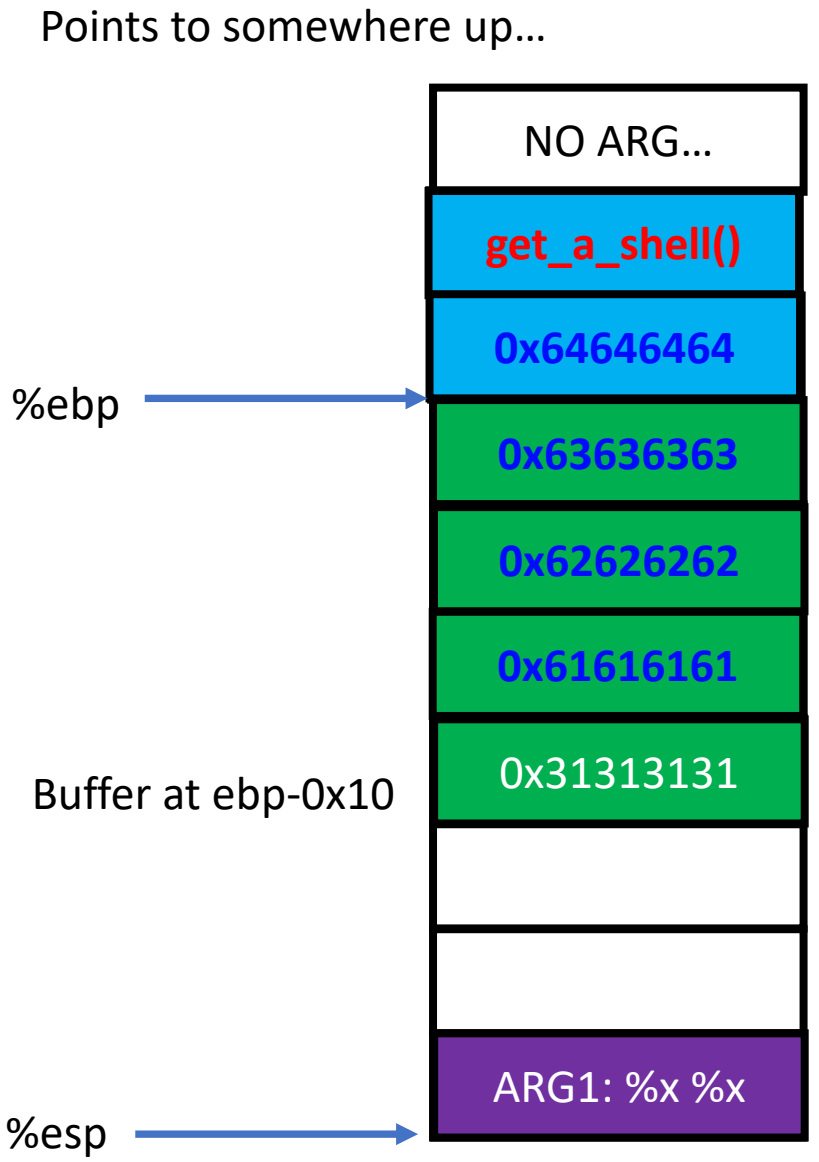
- Can you set that as the address of
  - get_a_shell()?



Buffer Overflow

- Function return of receive_input

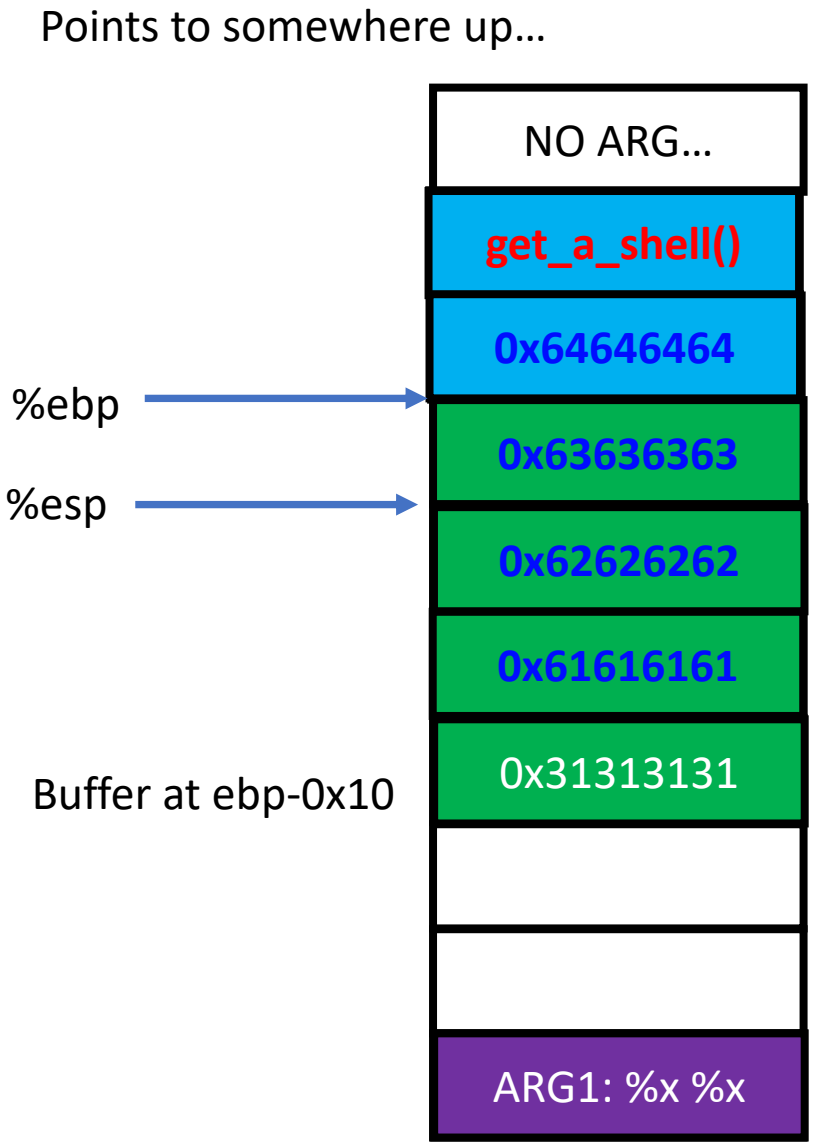
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Buffer Overflow

- Function return of receive_input

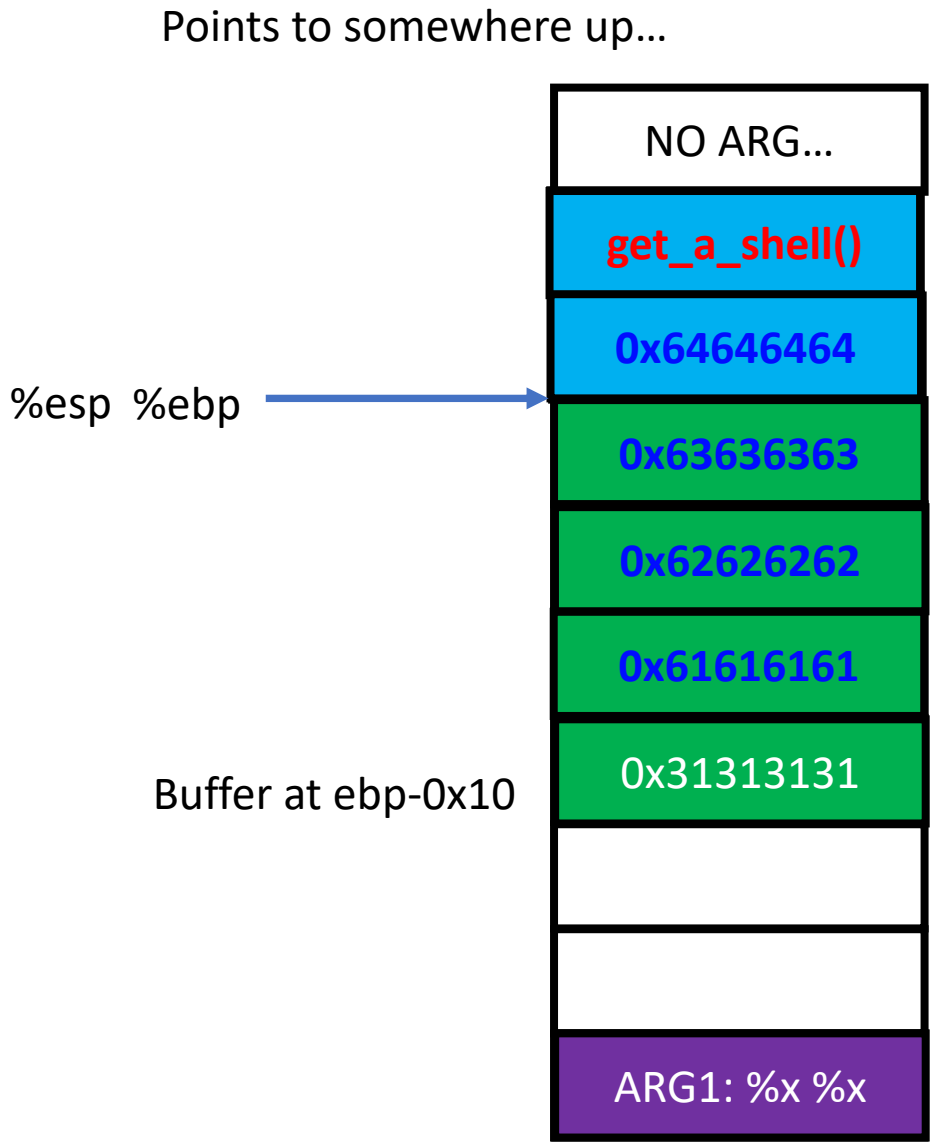
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Buffer Overflow

- Function return of receive_input

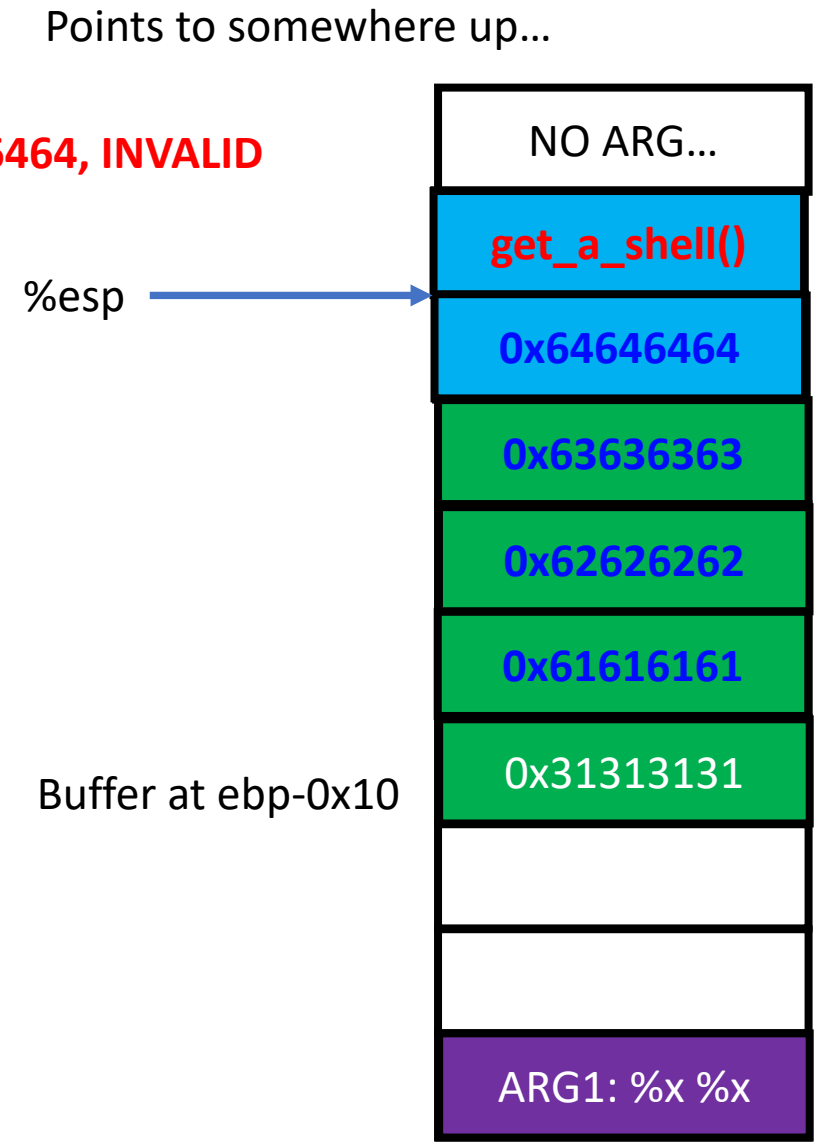
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Buffer Overflow

- Function return of receive_input

```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```

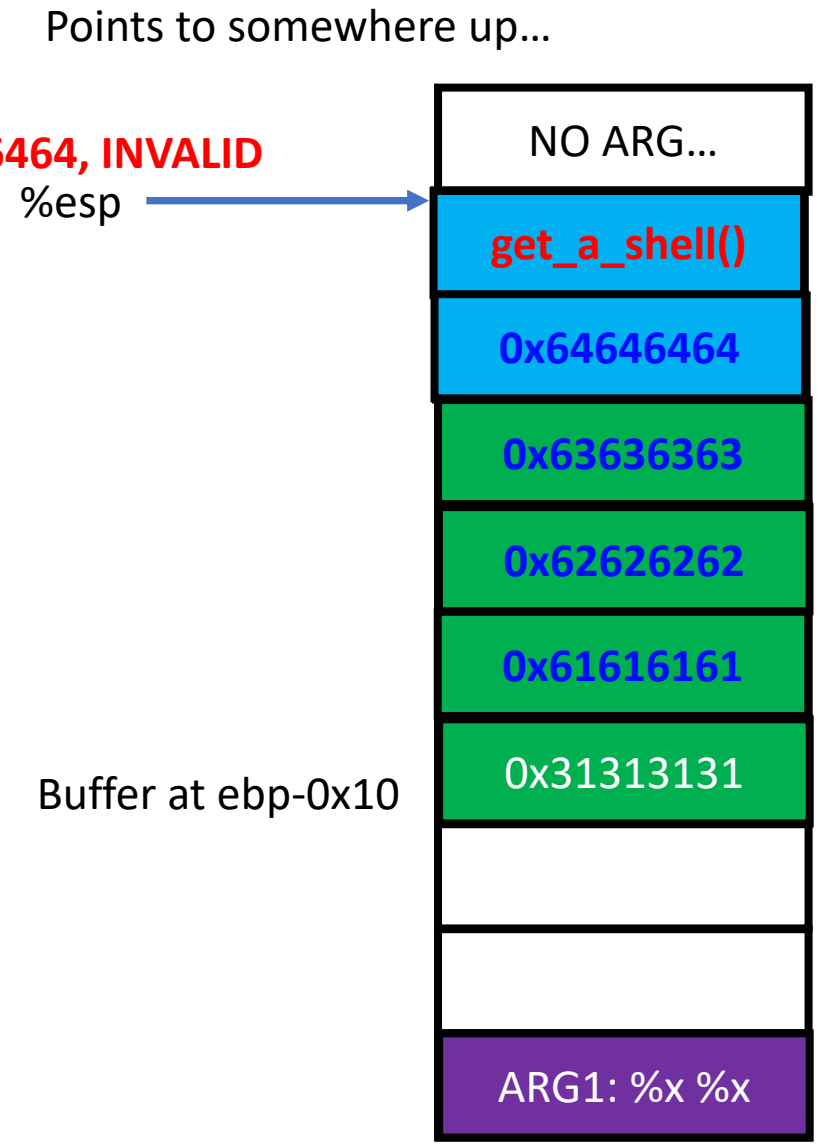


Buffer Overflow

- Function return of receive_input

```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```

Run get_a_shell()!



Objective of Week2 Challenges

- Identify the size of buffer in the program
- Try to overflow the buffer to control local variable/return address values
- Run get_a_shell()

Assignment: Week-2

- Please solve challenges in the `/home/labs/week2` directory
- Level0 – change the value of local variables via BOF
- Level1 – the same as level0 in amd64
- Level2 – change the return address to execute get_a_shell()
- Level3 – change the return address to execute get_a_shell(), 64bit
- Level4 – Defeat a simple defense to return address overwriting
- **Due: 04/20 2:00pm**