Cyber Attacks & Defense

Buffer Overflow

Dr. Yeongjin Jang

Oregon State University

Week1

- Due: 1/23 10:00am
- Late submission due: 1/30 10:00am (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 (ABI)
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

ABI: Application Binary Interface; Similar to API, Application Programming Interface
API is for human developers, and ABI is for the CPU

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`

If a function has more than 6 arguments, then amd64 ABI uses stack to store 7th argument and on

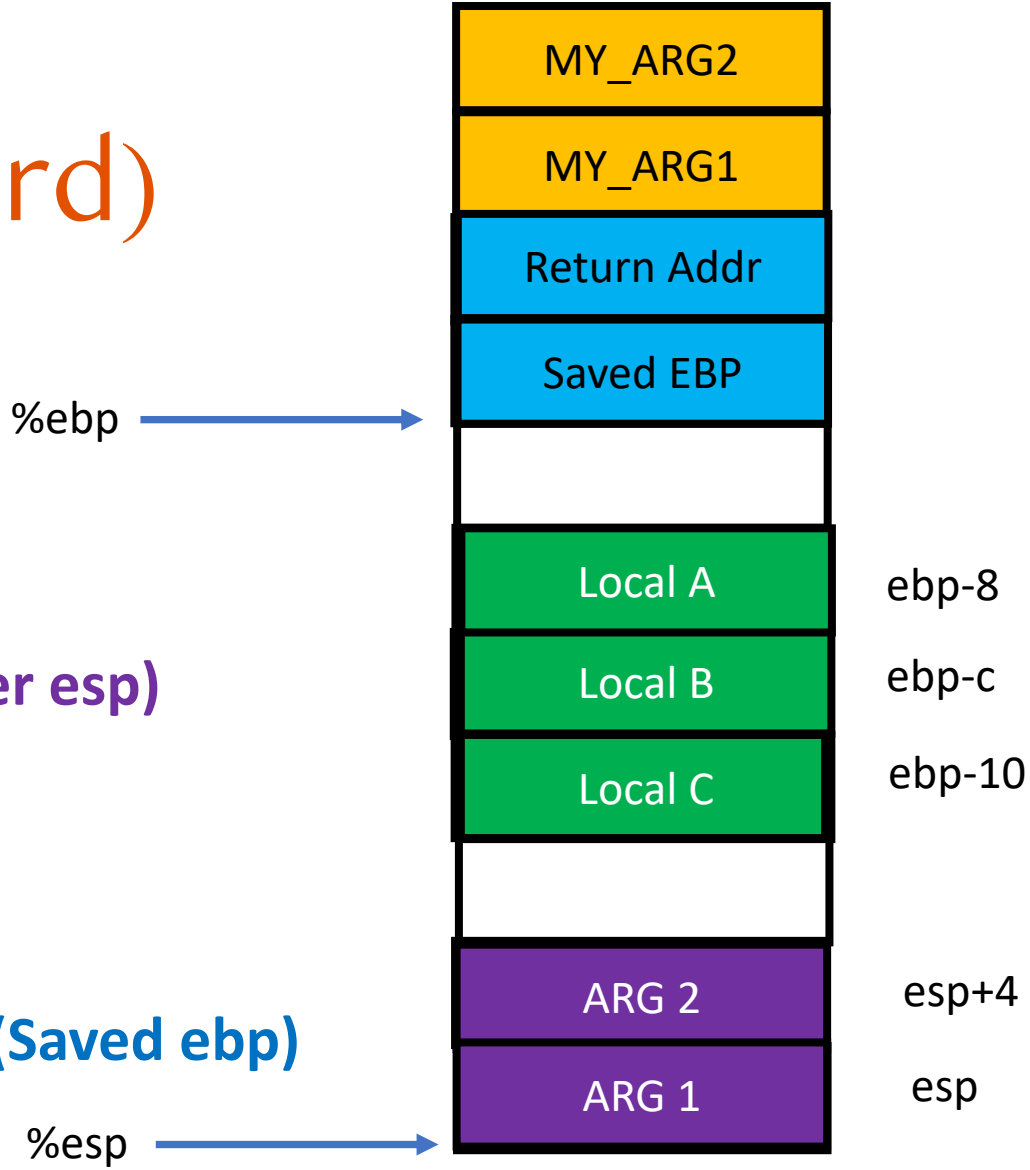
amd64 Calling Convention

%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)

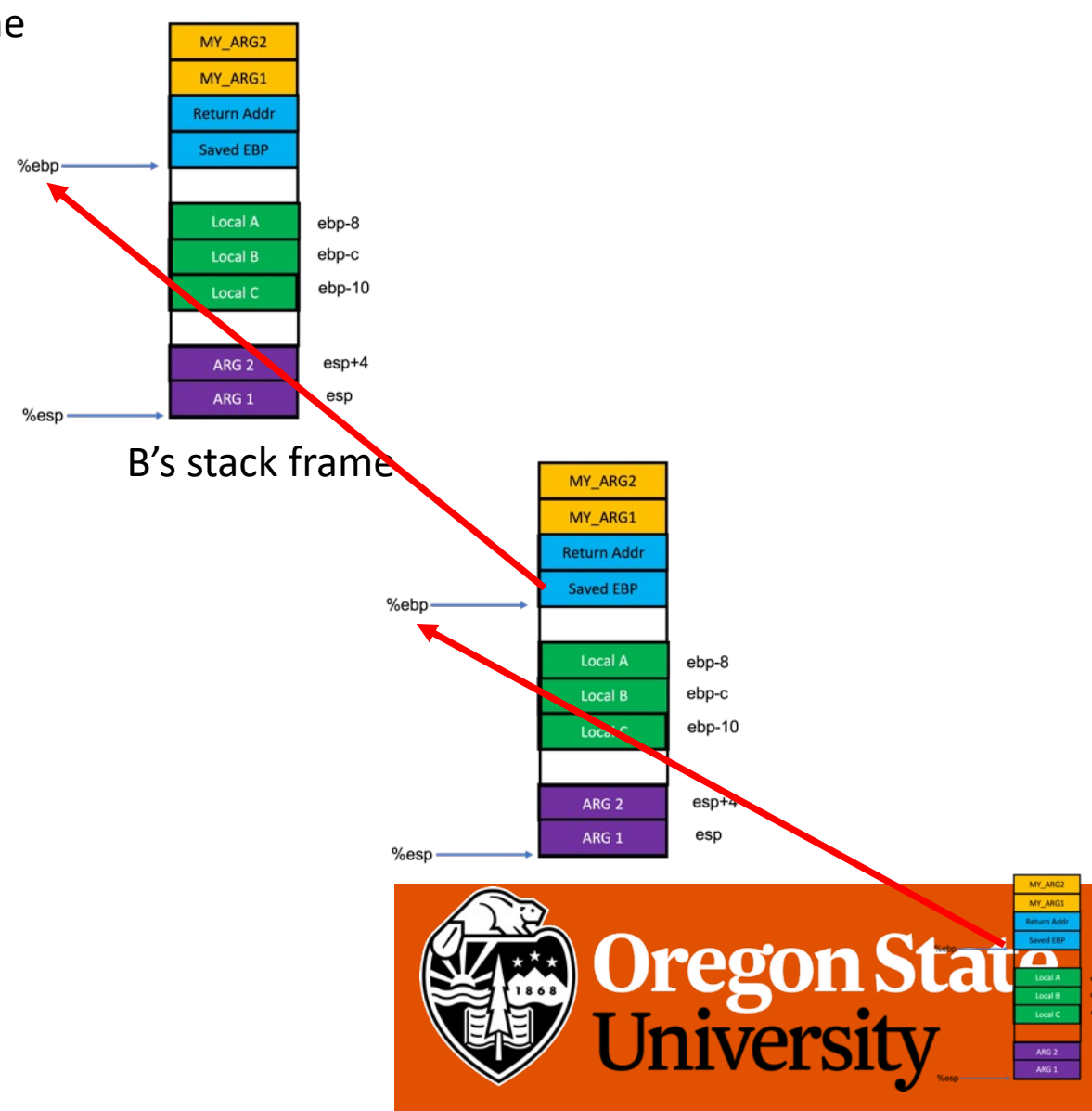
- Defines a variable scope of a function
  - **Local variables (negative index over ebp)**
  - **Arguments (positive index over ebp)**
  - **Function call arguments (positive index over esp)**
- Maintains nested function calls
  - **Return target (return address)**
  - **Local variables of the upper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

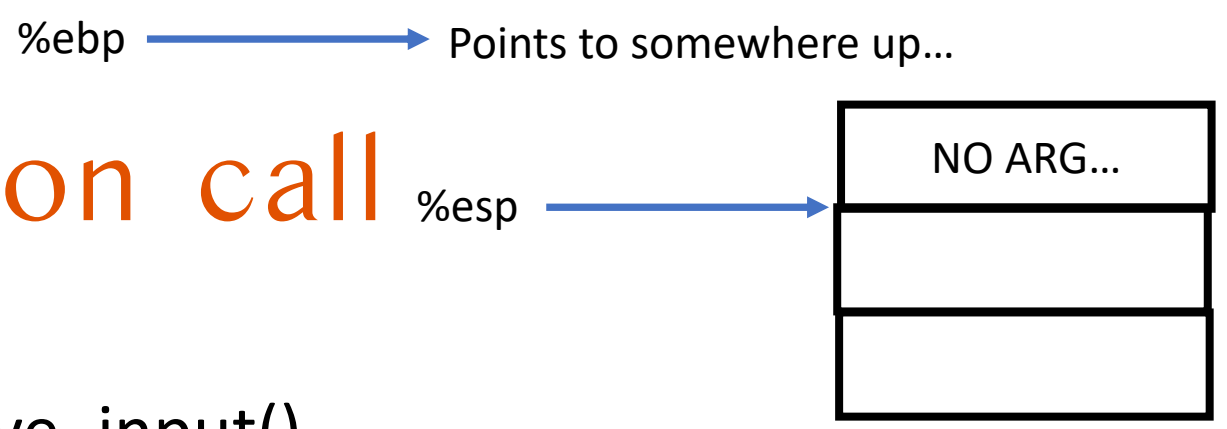


Stack (Grows Downward)

If a() calls b() and then b() calls c()….

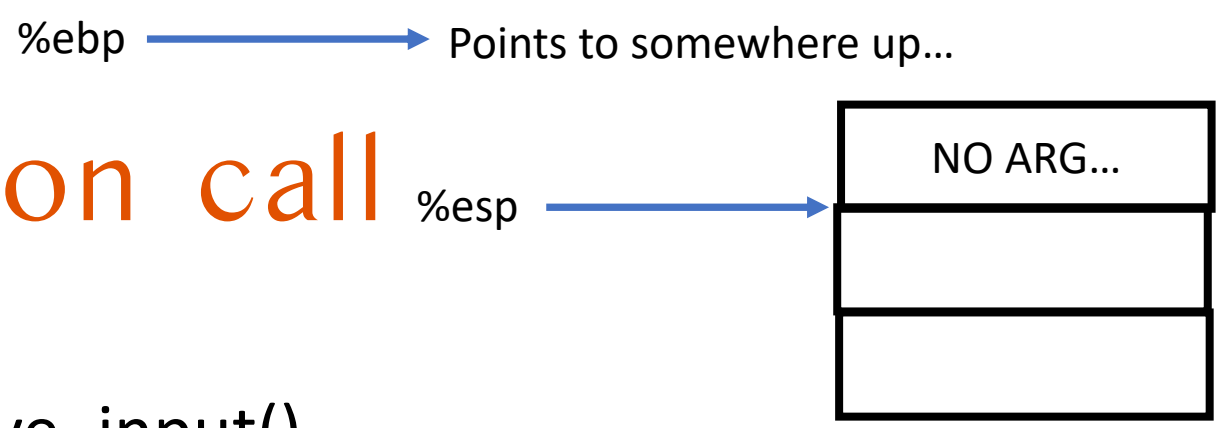
When it returns, we restore ebp!





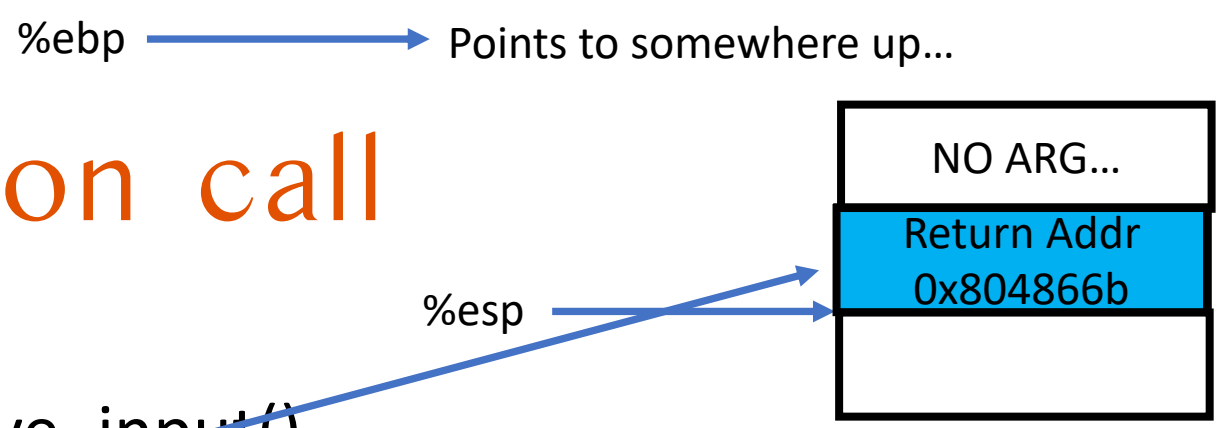
Example – Function call

- 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`

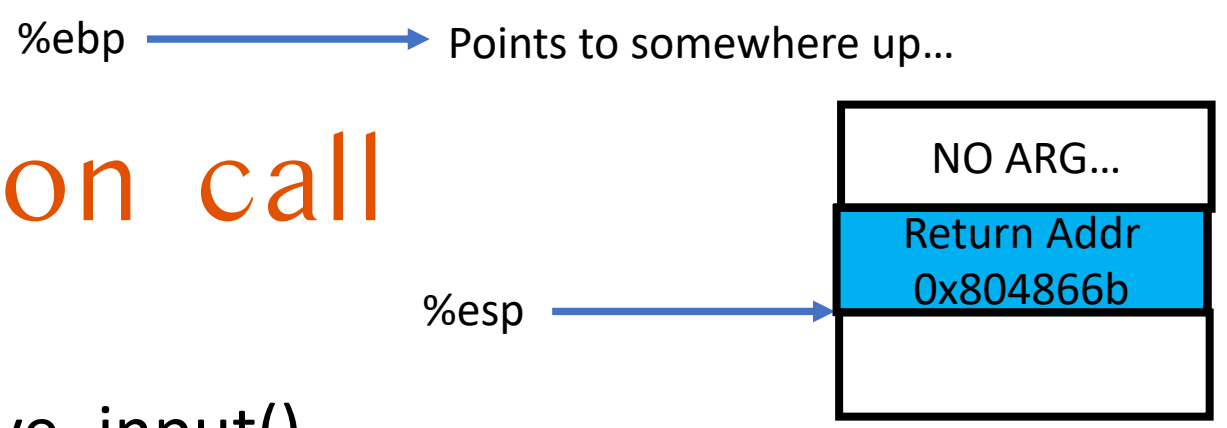


Example – Function call

- 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!

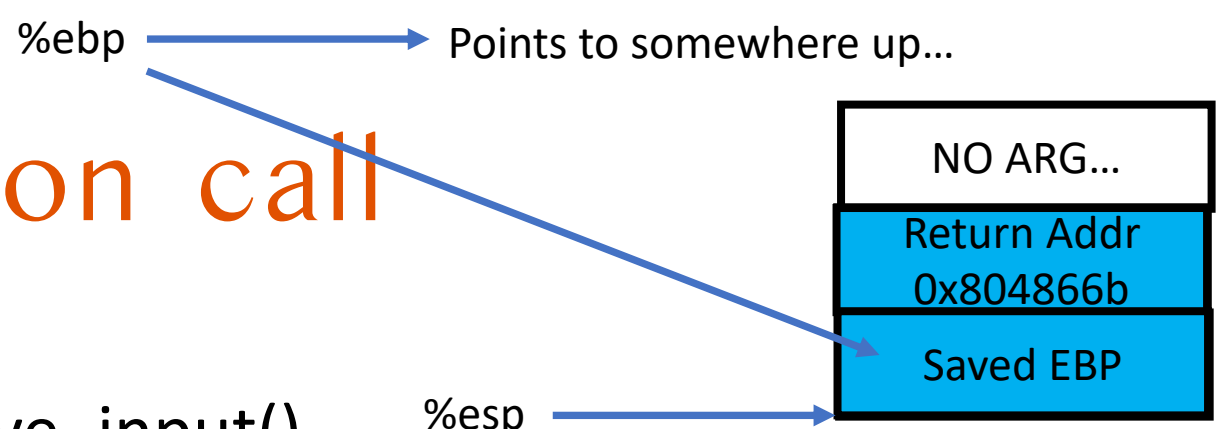
```
push %eip   -- points the next instruction
jmp 0x8048570 <receive_input>
```



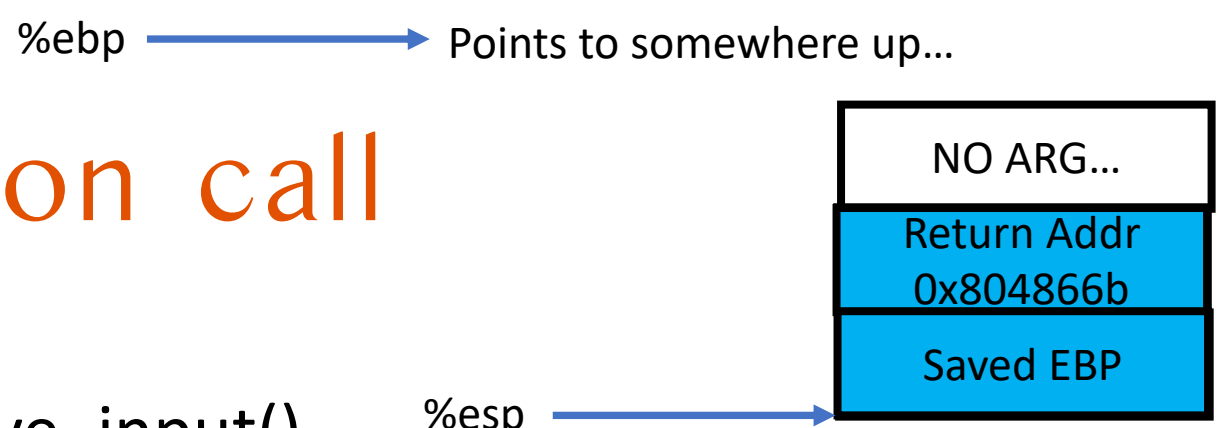
Example – Function call

- 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



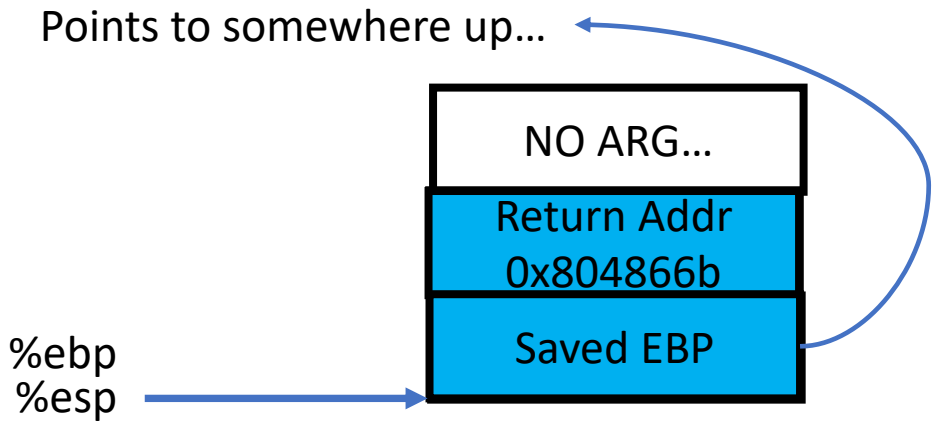
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - **`0x08048570 <+0>: push  %ebp`**
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`



Example – Function call

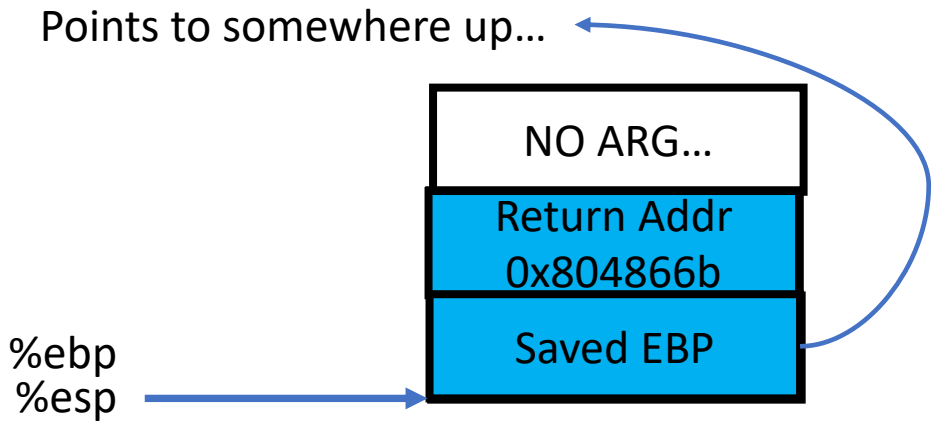
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - **`0x08048571 <+1>: mov  %esp,%ebp`**
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`

Example – Function call



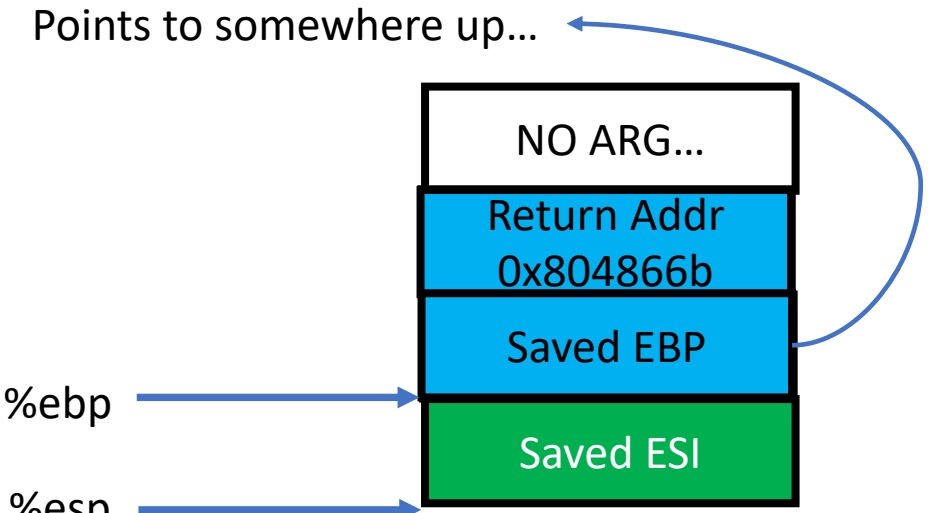
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - **`0x08048571 <+1>: mov  %esp,%ebp`**
  - `0x08048573 <+3>: push  %esi`
  - `0x08048574 <+4>: sub  $0x54,%esp`

Example – Function call



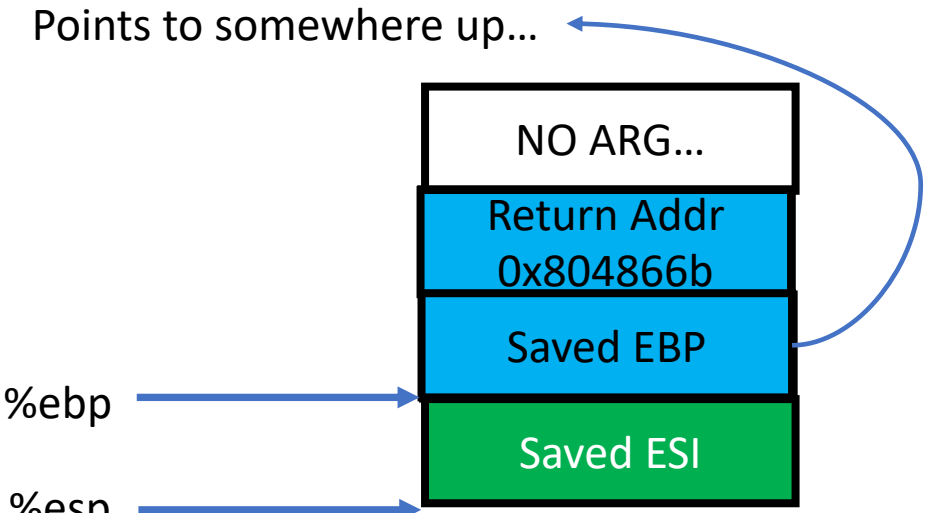
- In bof-level0, main() calls receive_input()
  - `call 0x8048570 <receive_input>`
  - `0x0804866b <+11>: xor %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push %ebp`
  - `0x08048571 <+1>: mov %esp,%ebp`
  - **`0x08048573 <+3>: push %esi`**
  - `0x08048574 <+4>: sub $0x54,%esp`

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`

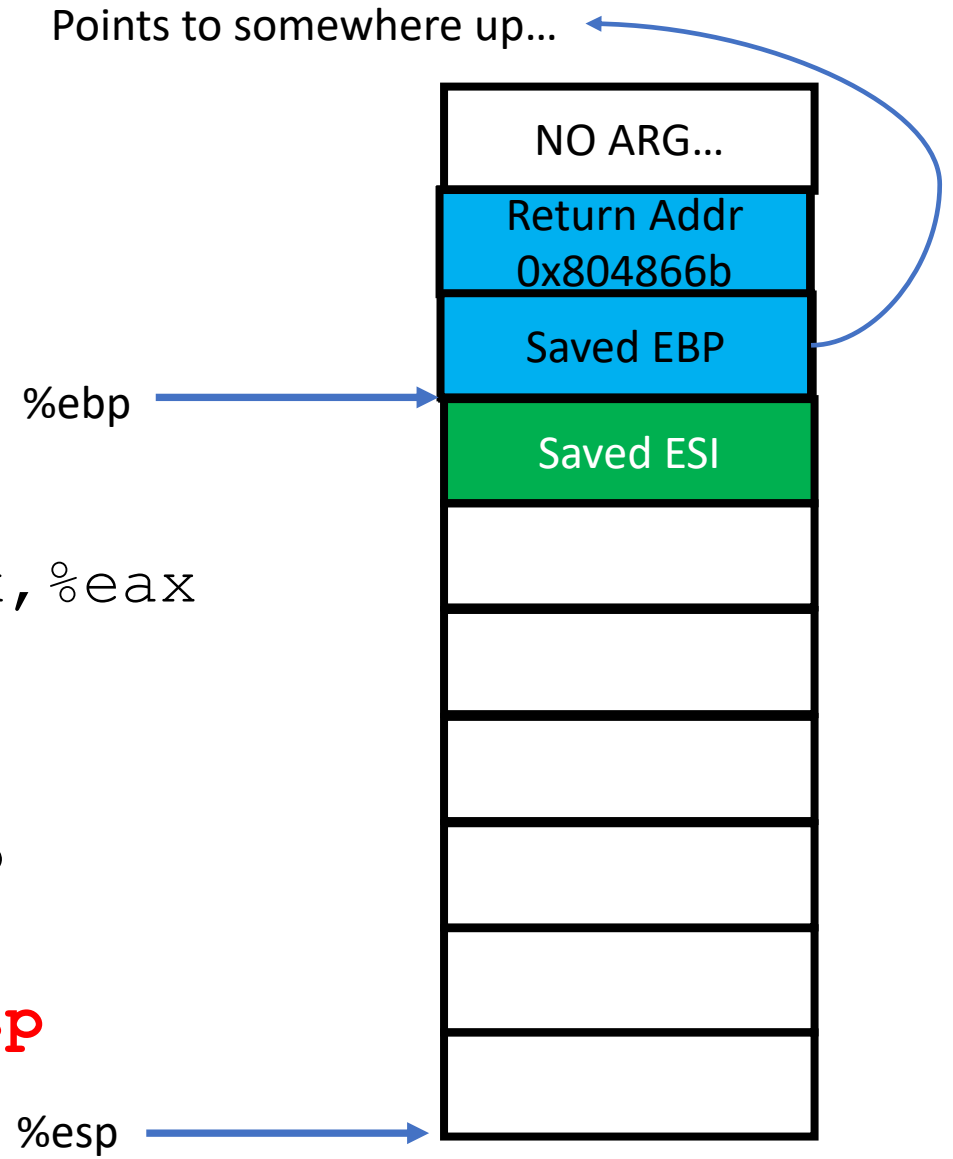
Example – Function call



- 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

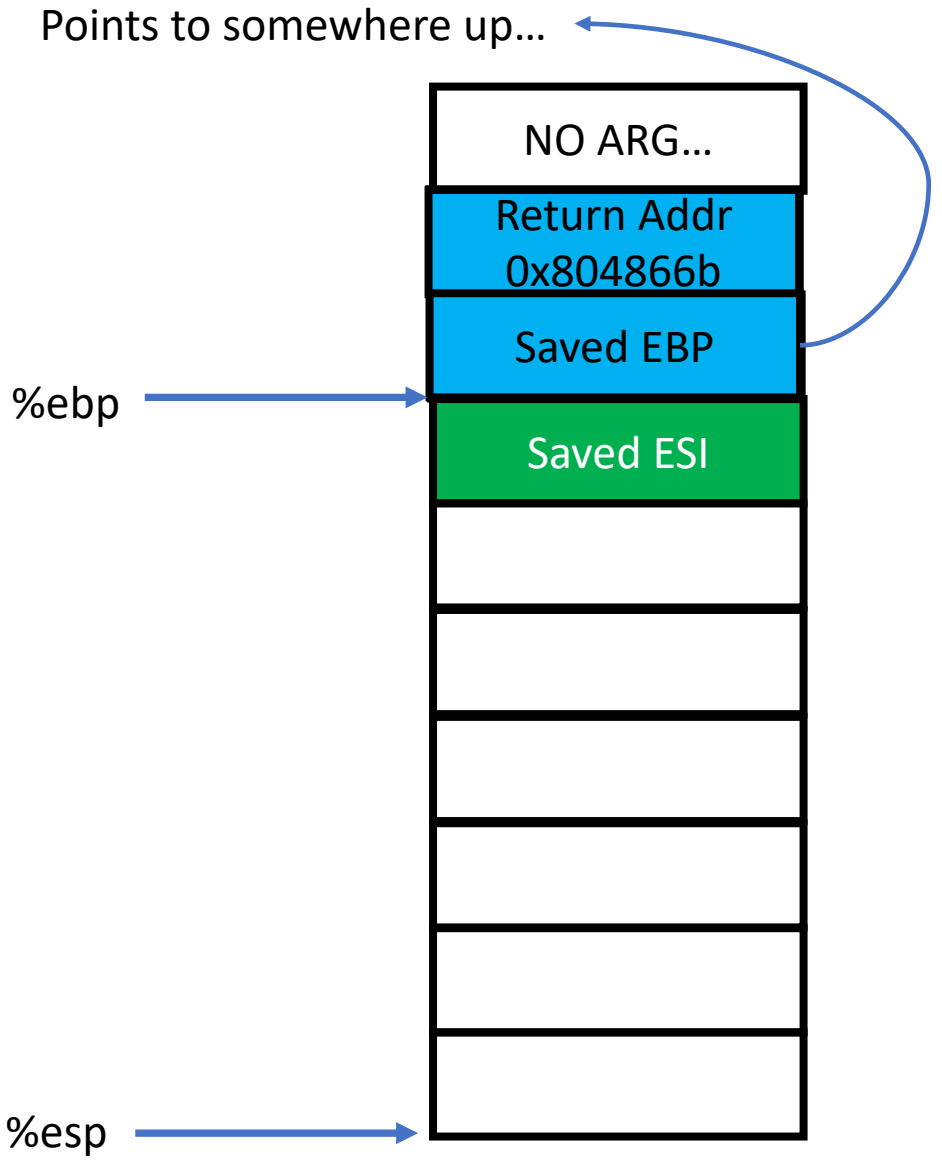
- In bof-level0, main() calls receive_input()
  - `call  0x8048570 <receive_input>`
  - `0x0804866b <+11>:  xor  %eax,%eax`
- Head of receive_input
  - `0x08048570 <+0>: push  %ebp`
  - `0x08048571 <+1>: mov  %esp,%ebp`
  - `0x08048573 <+3>: push  %esi`
  - **`0x08048574 <+4>: sub  $0x54,%esp`**



Example – Function call

- Call printf?

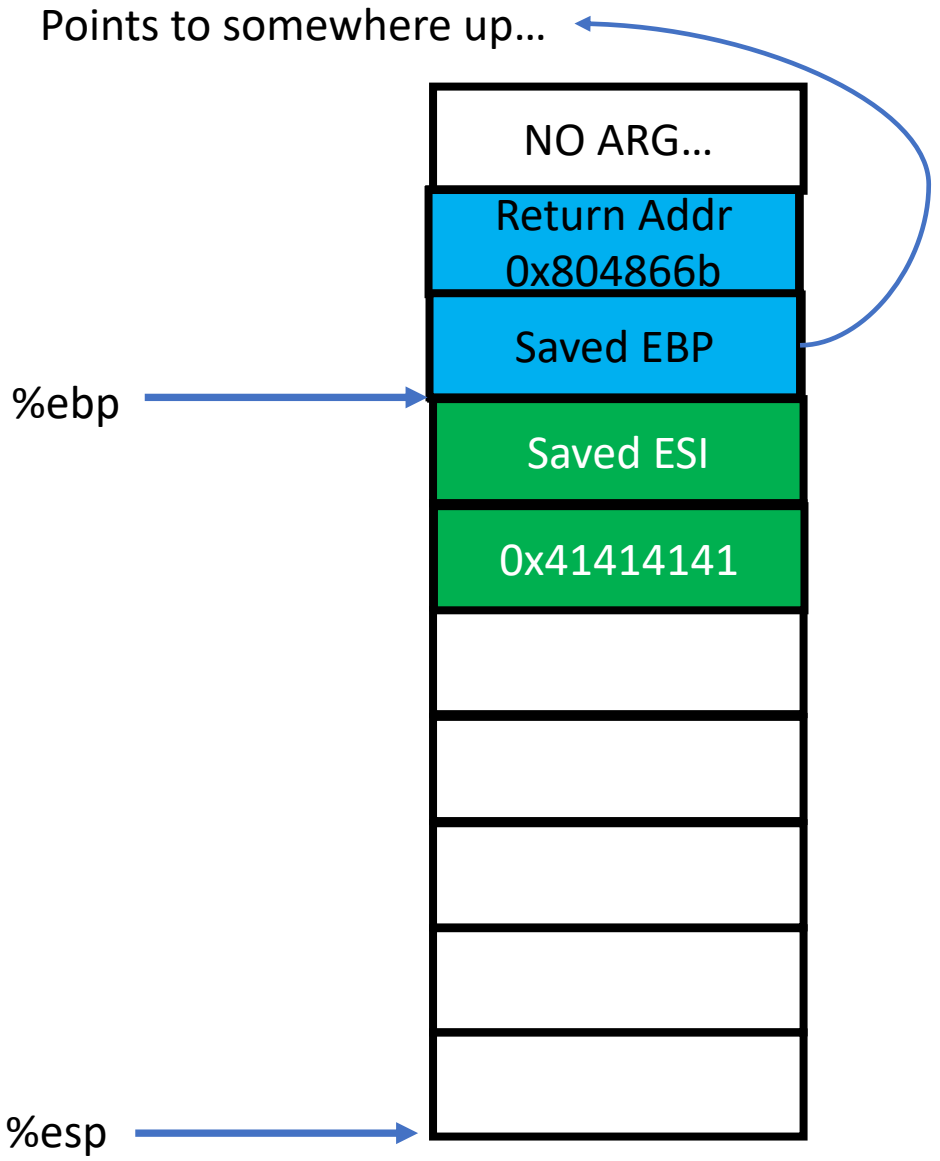
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

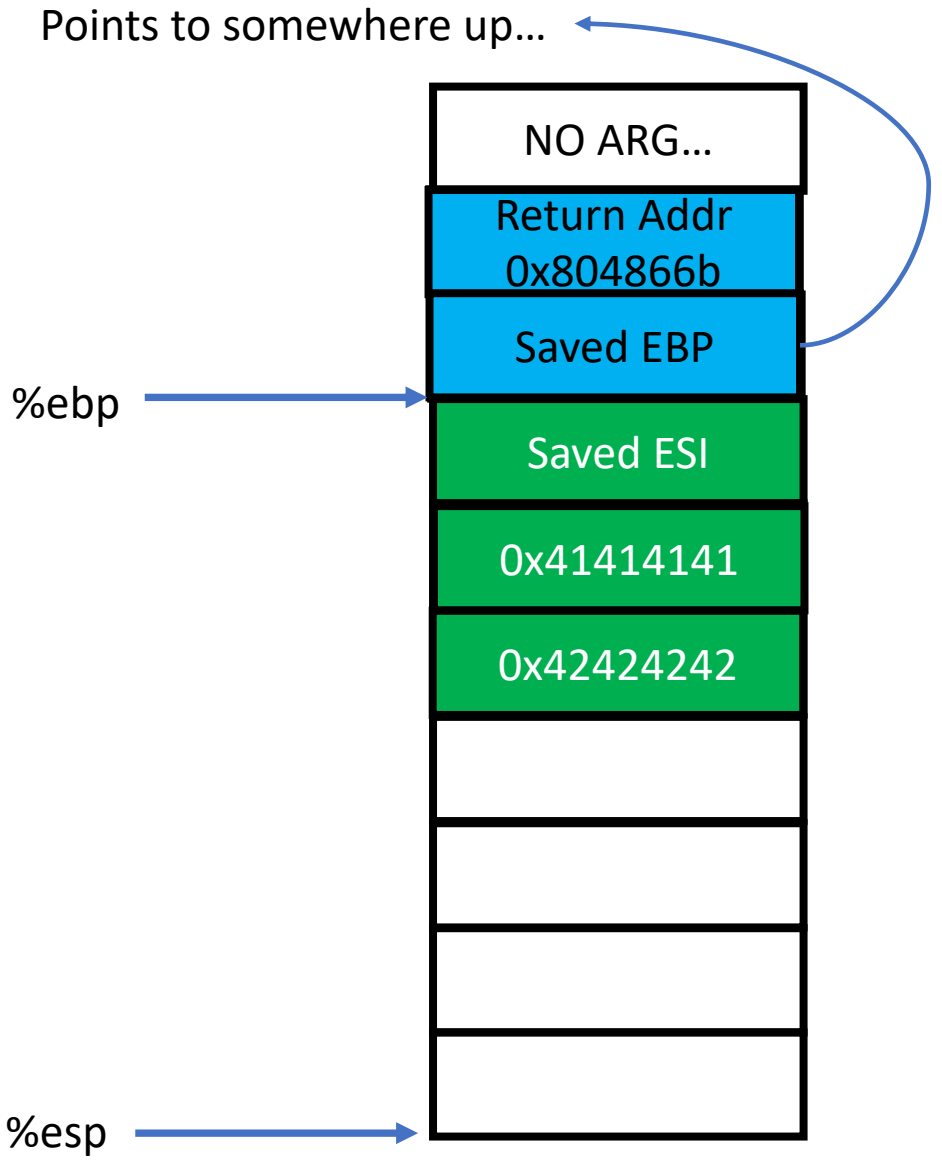
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

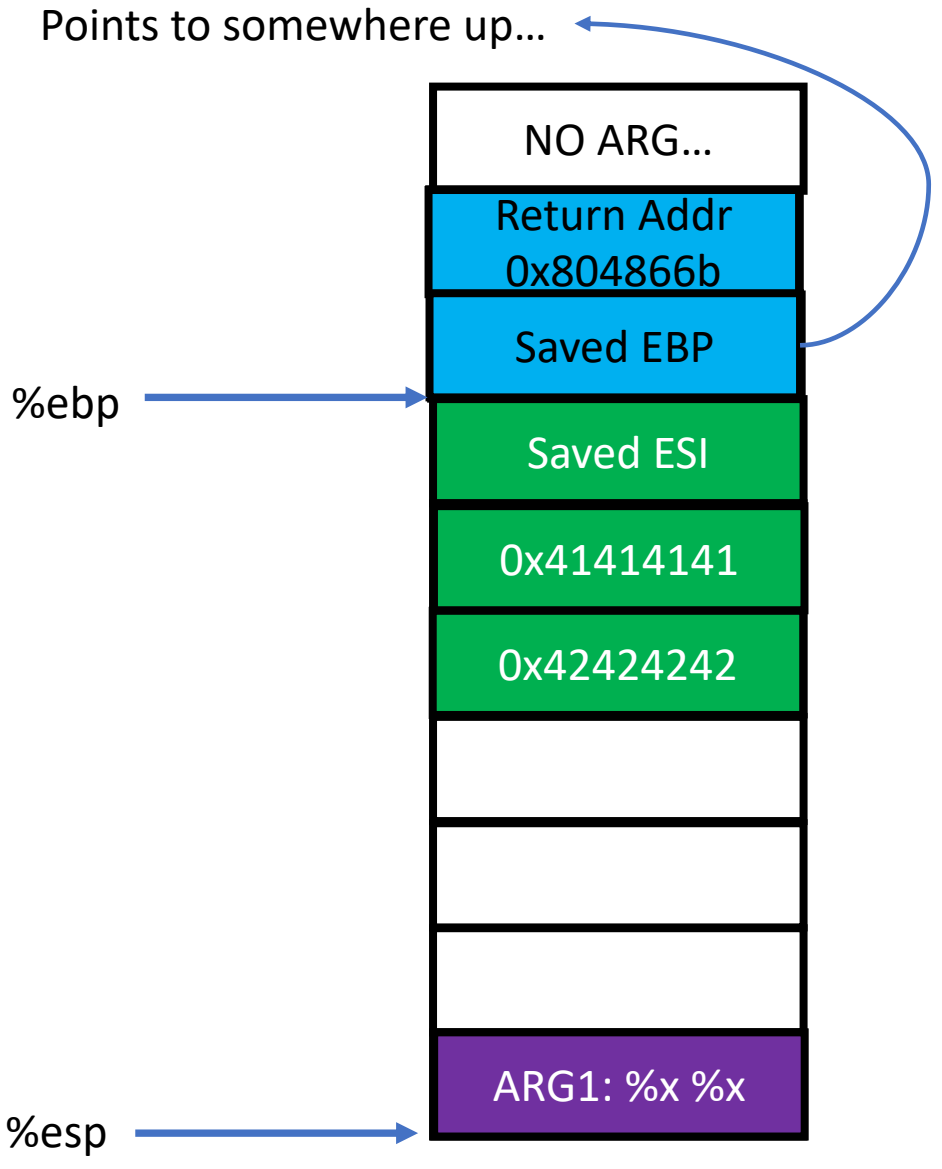
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

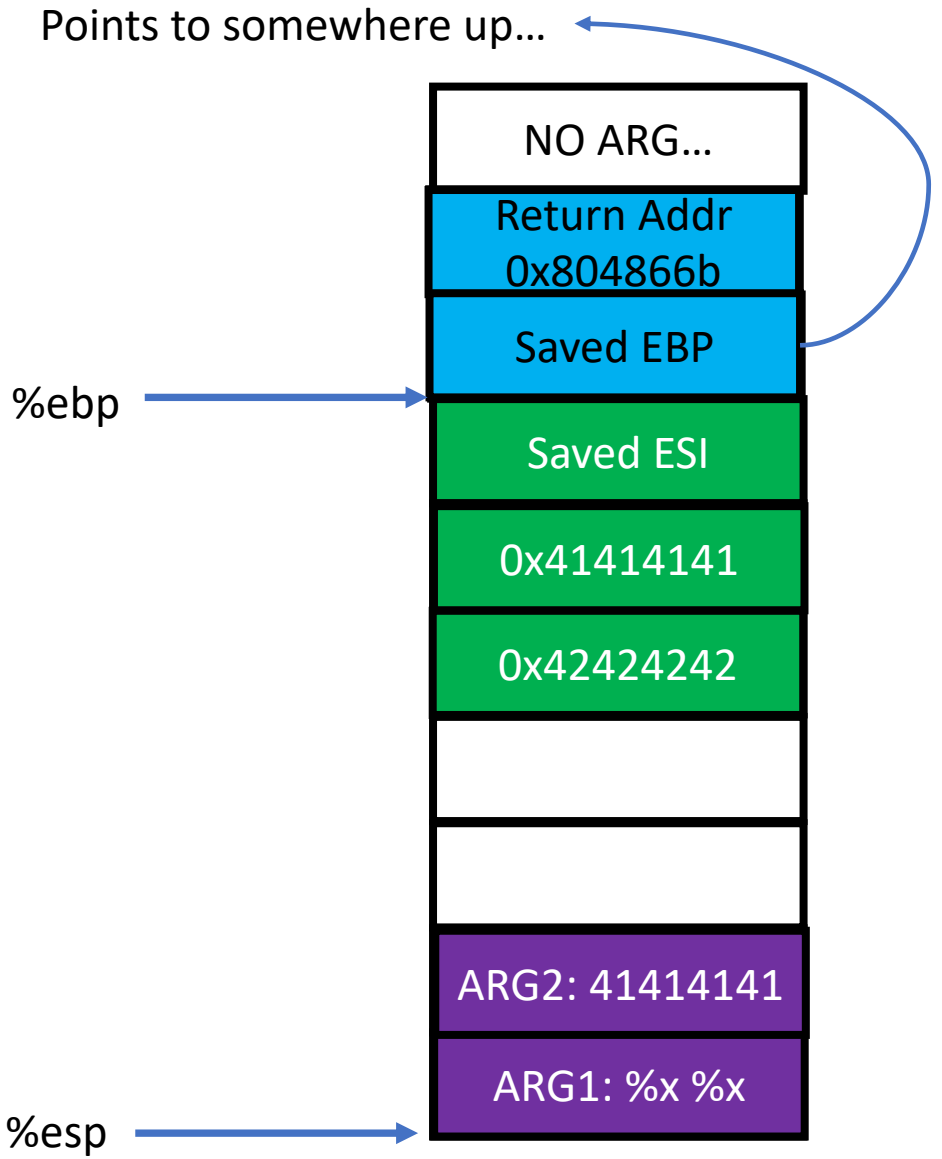
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

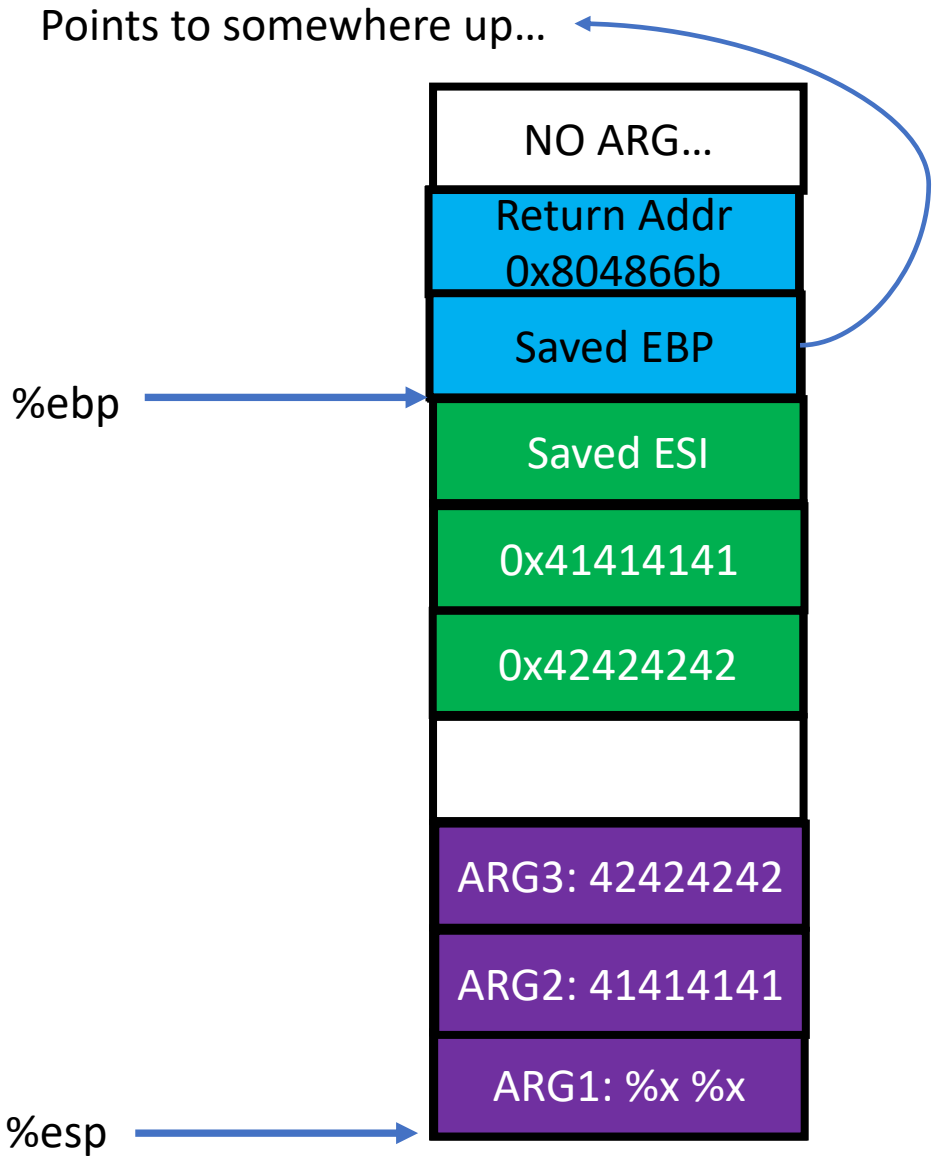
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

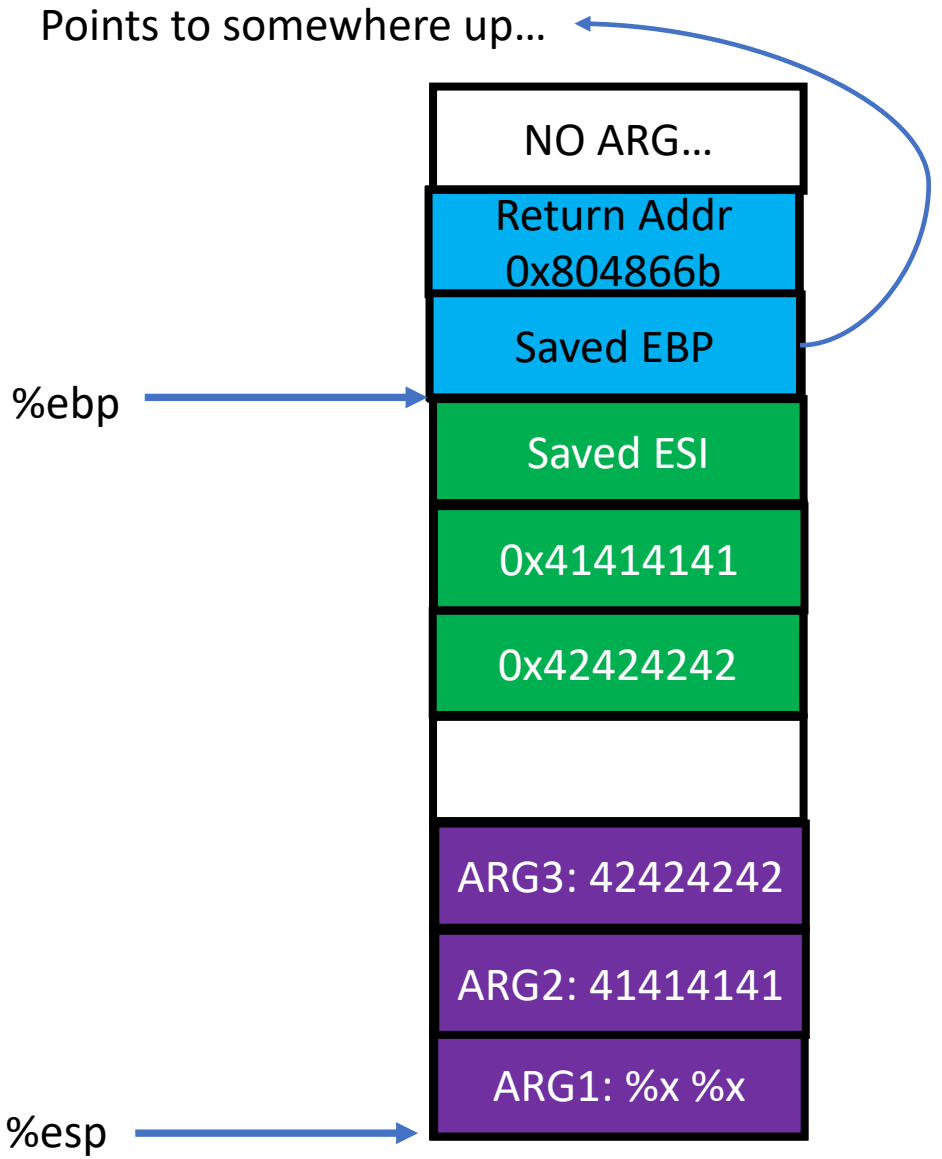
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
```



Example – Function call

- Call printf?

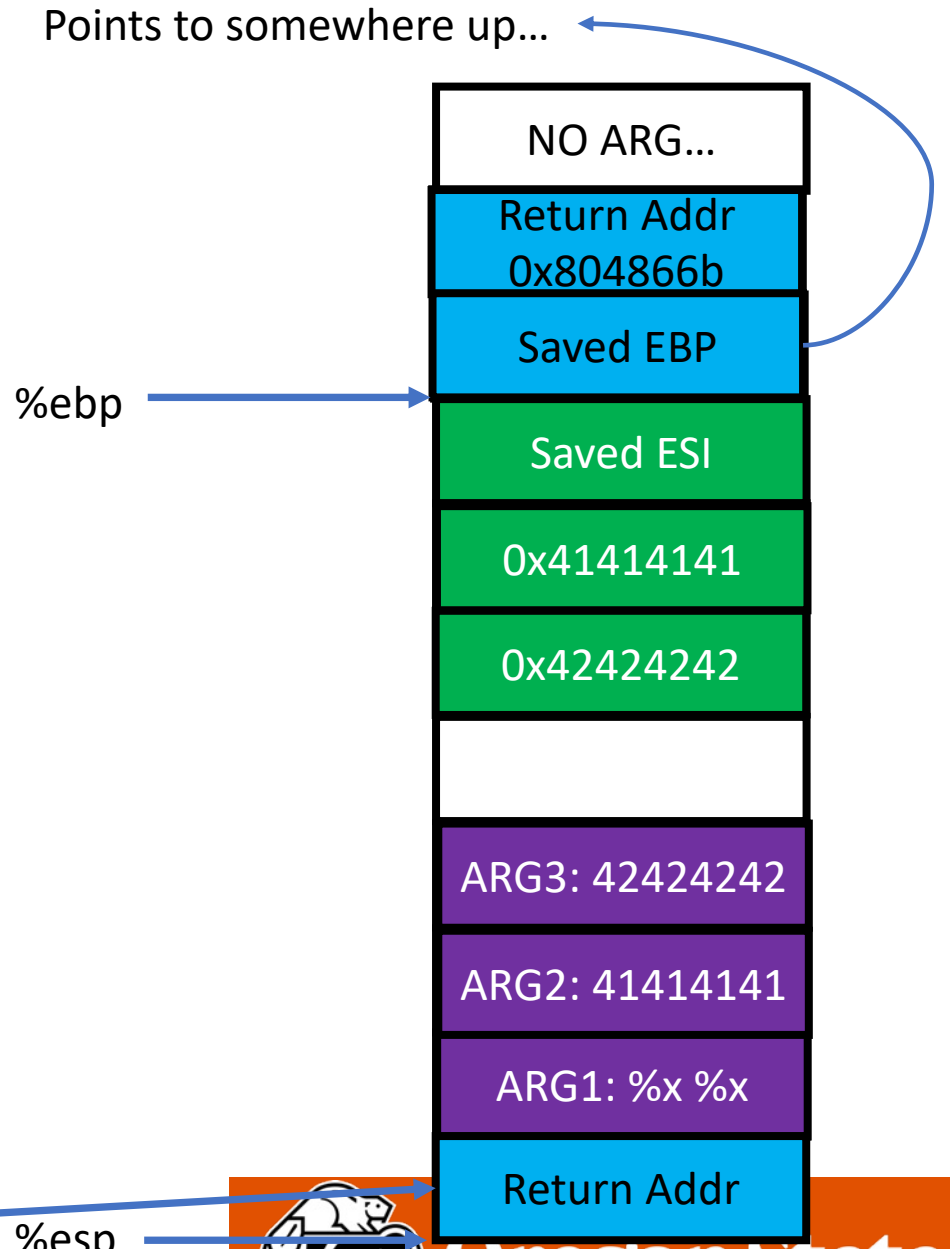
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
lea     0x8048765,%ecx
```



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
```

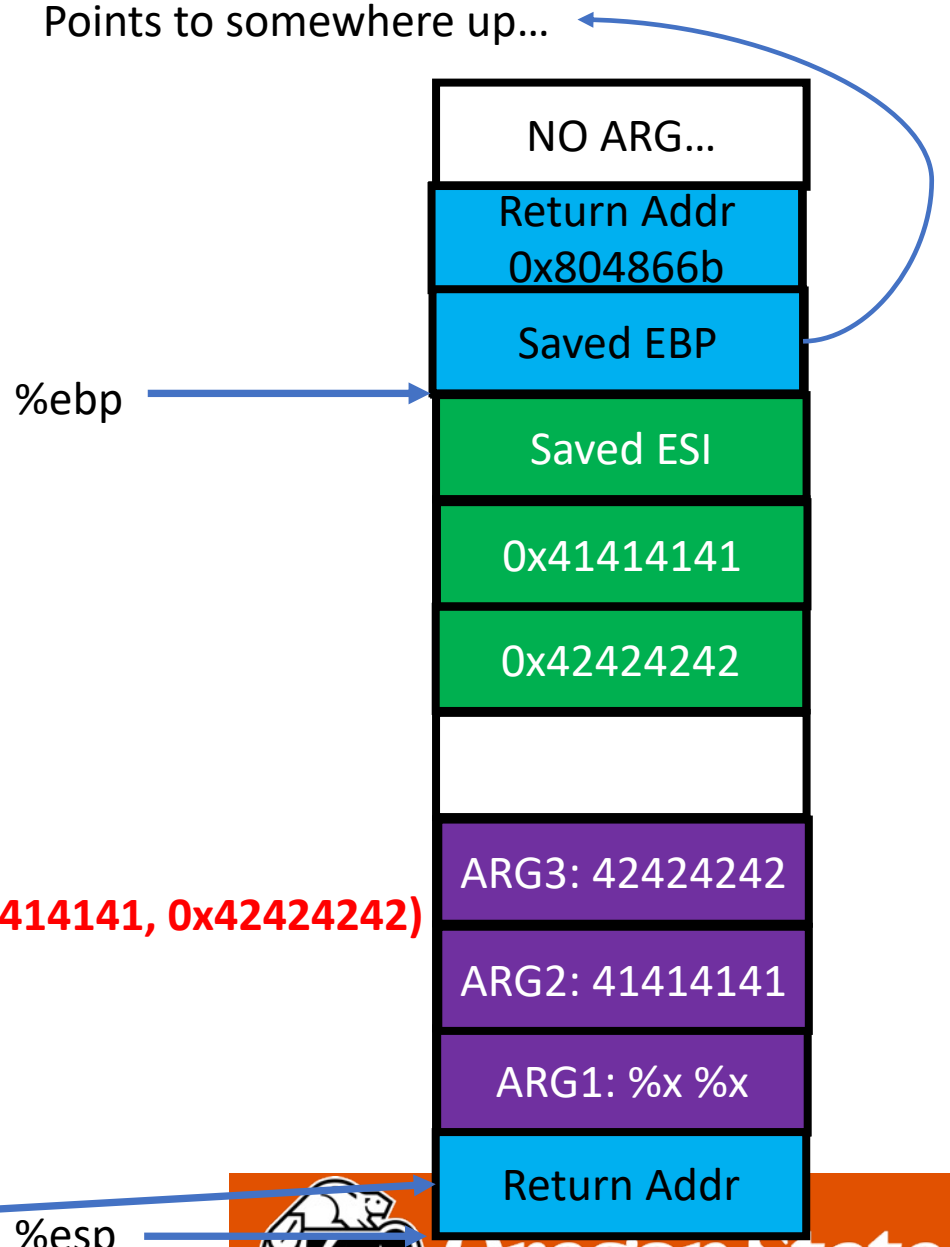


Example – Function call

- Call printf?

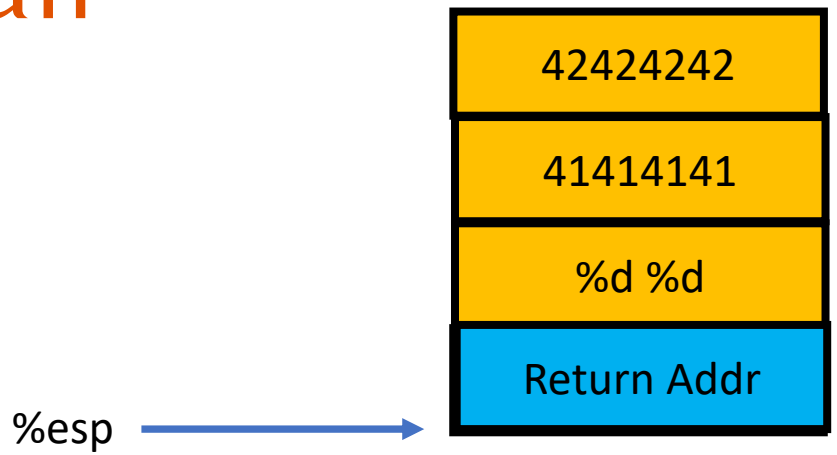
```
movl    $0x41414141,-0x8(%ebp)
movl    $0x42424242,-0xc(%ebp)
lea     0x8048727,%eax
mov     -0x8(%ebp),%ecx
mov     -0xc(%ebp),%edx
mov     %eax,(%esp)
mov     %ecx,0x4(%esp)
mov     %edx,0x8(%esp)
call    0x8048370 <printf@plt>
lea     0x8048765,%ecx
```

Calls printf("%x %x", 0x41414141, 0x42424242)



Example – Function call

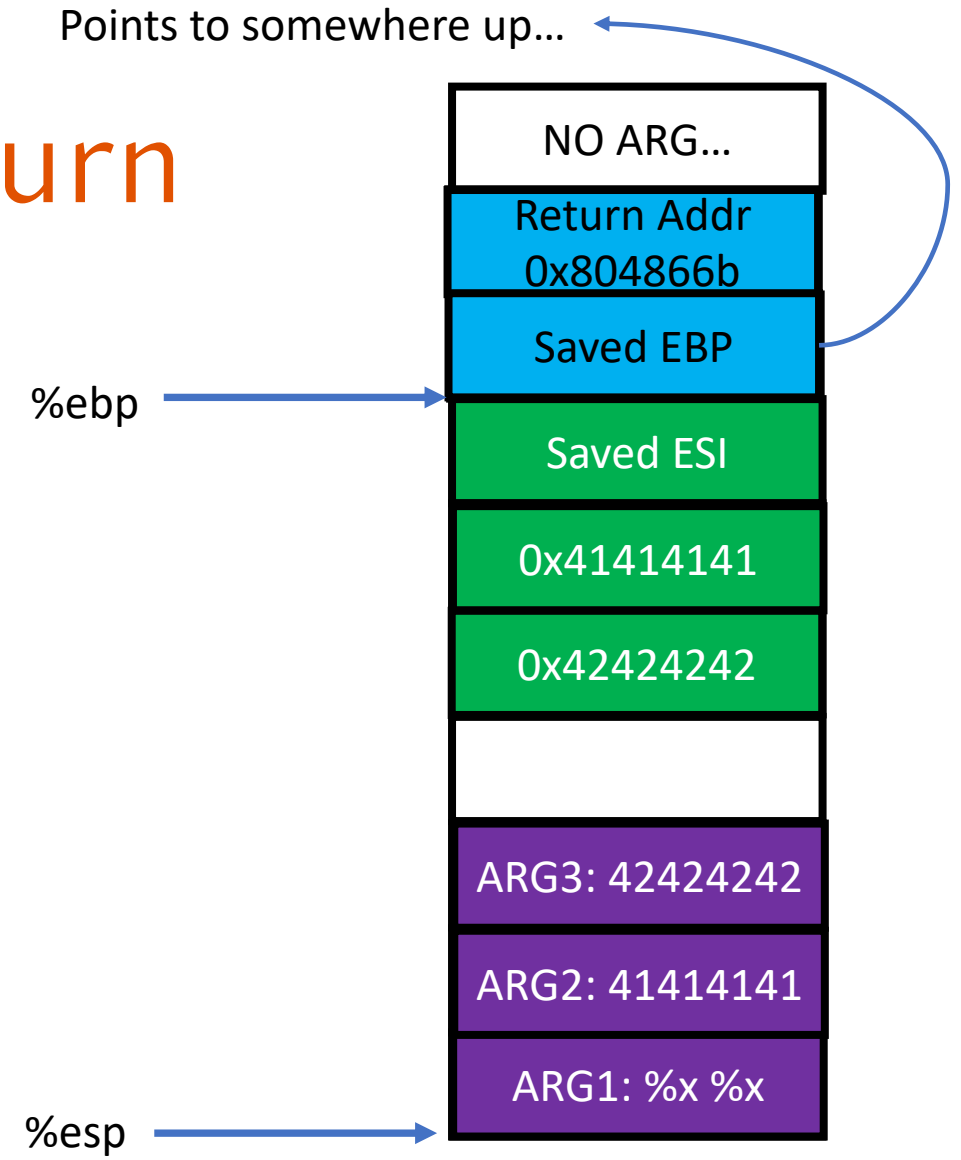
- What printf will see?



Example – Function Return

- Function return of receive_input

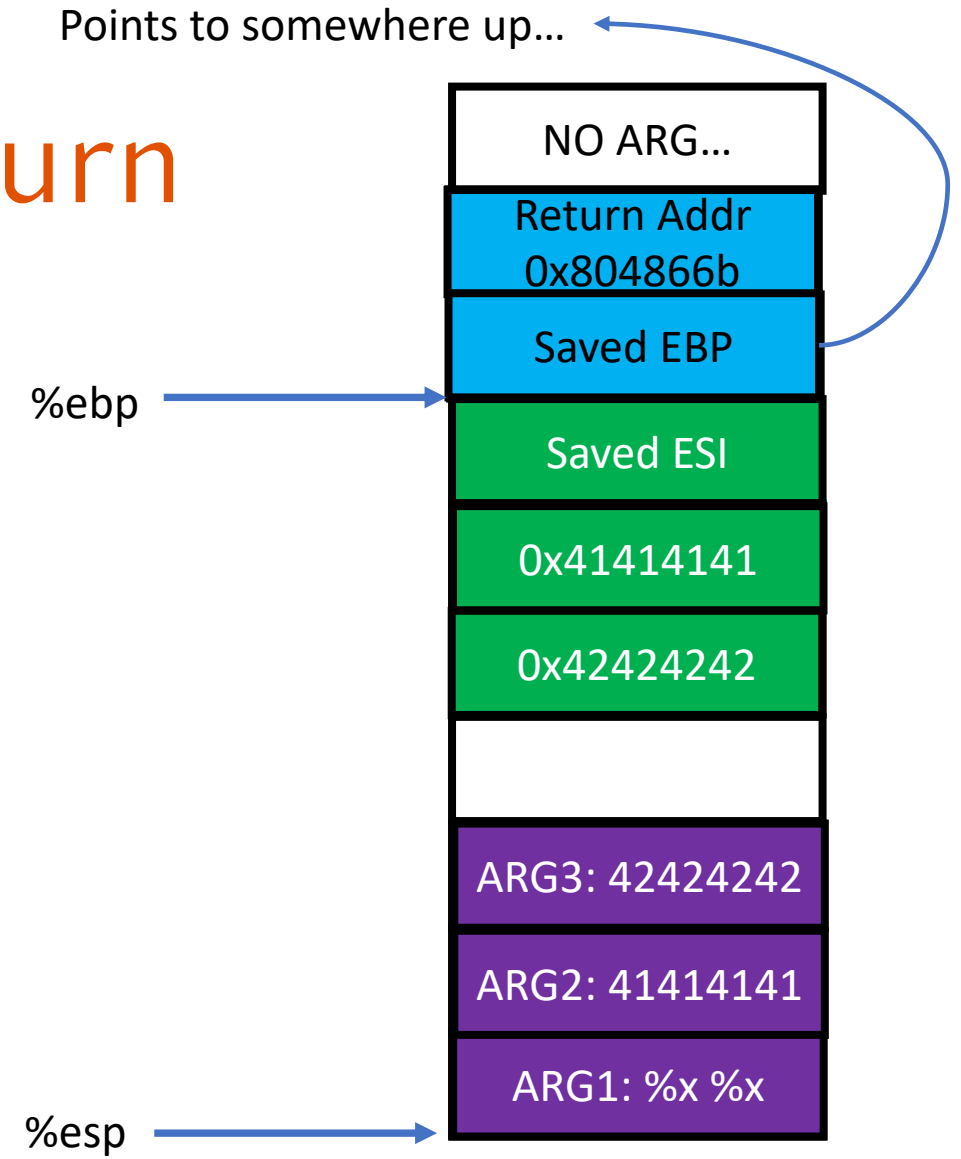
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

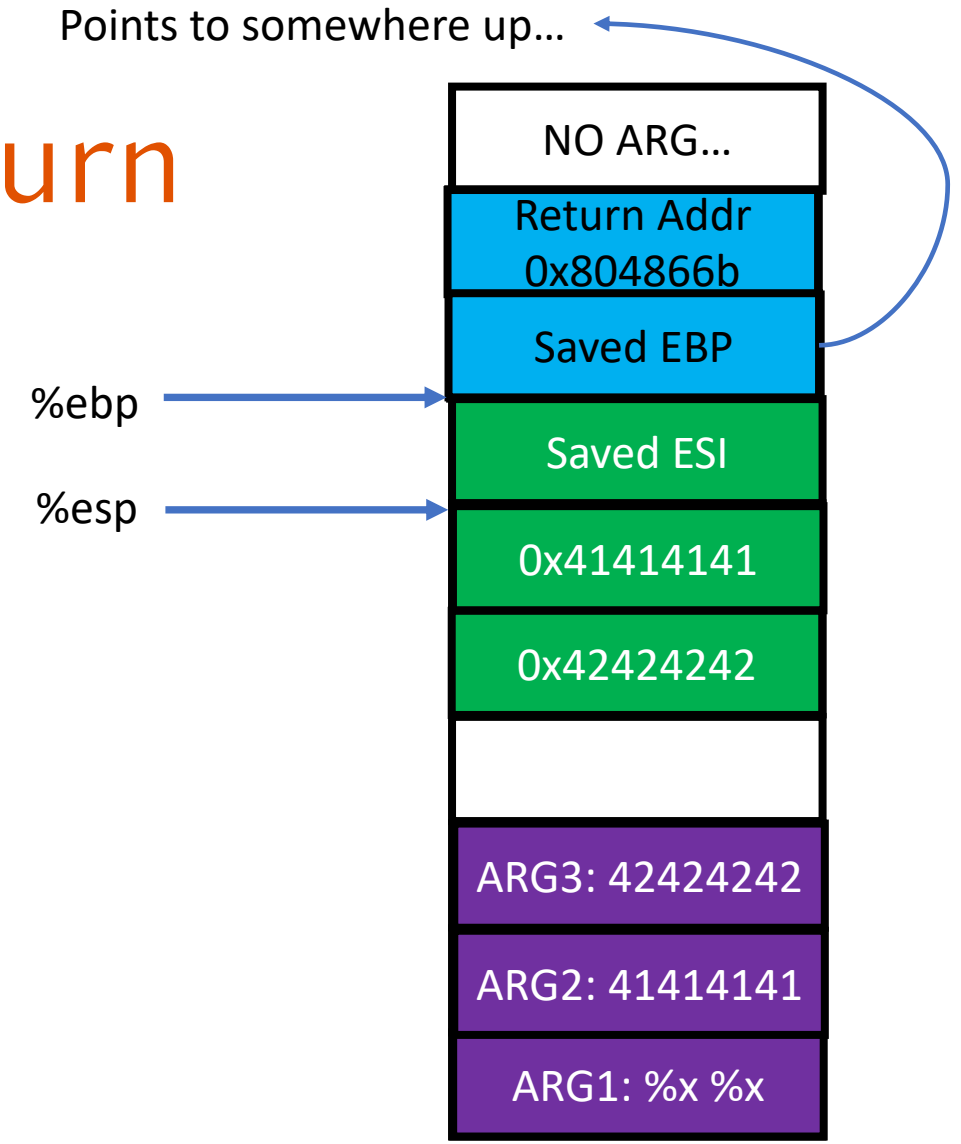
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

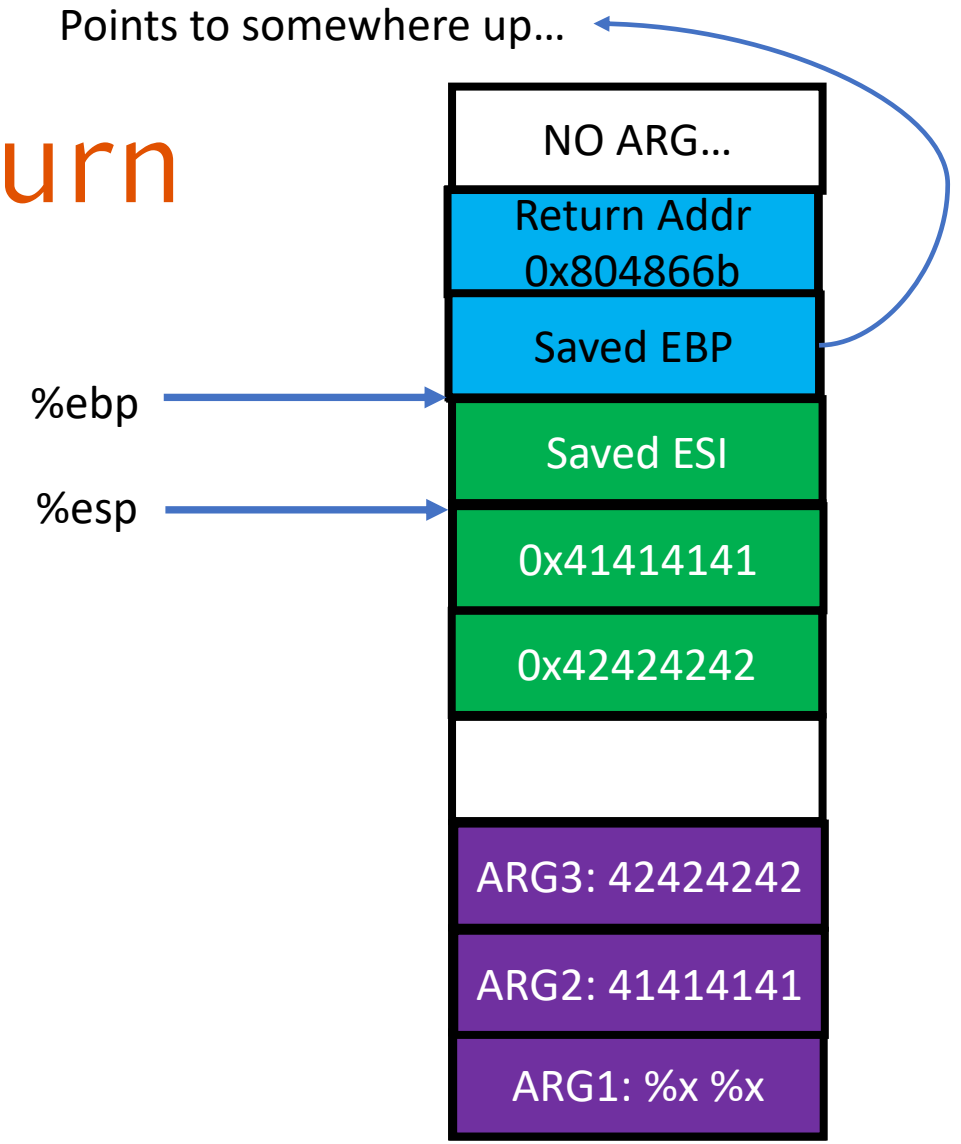
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

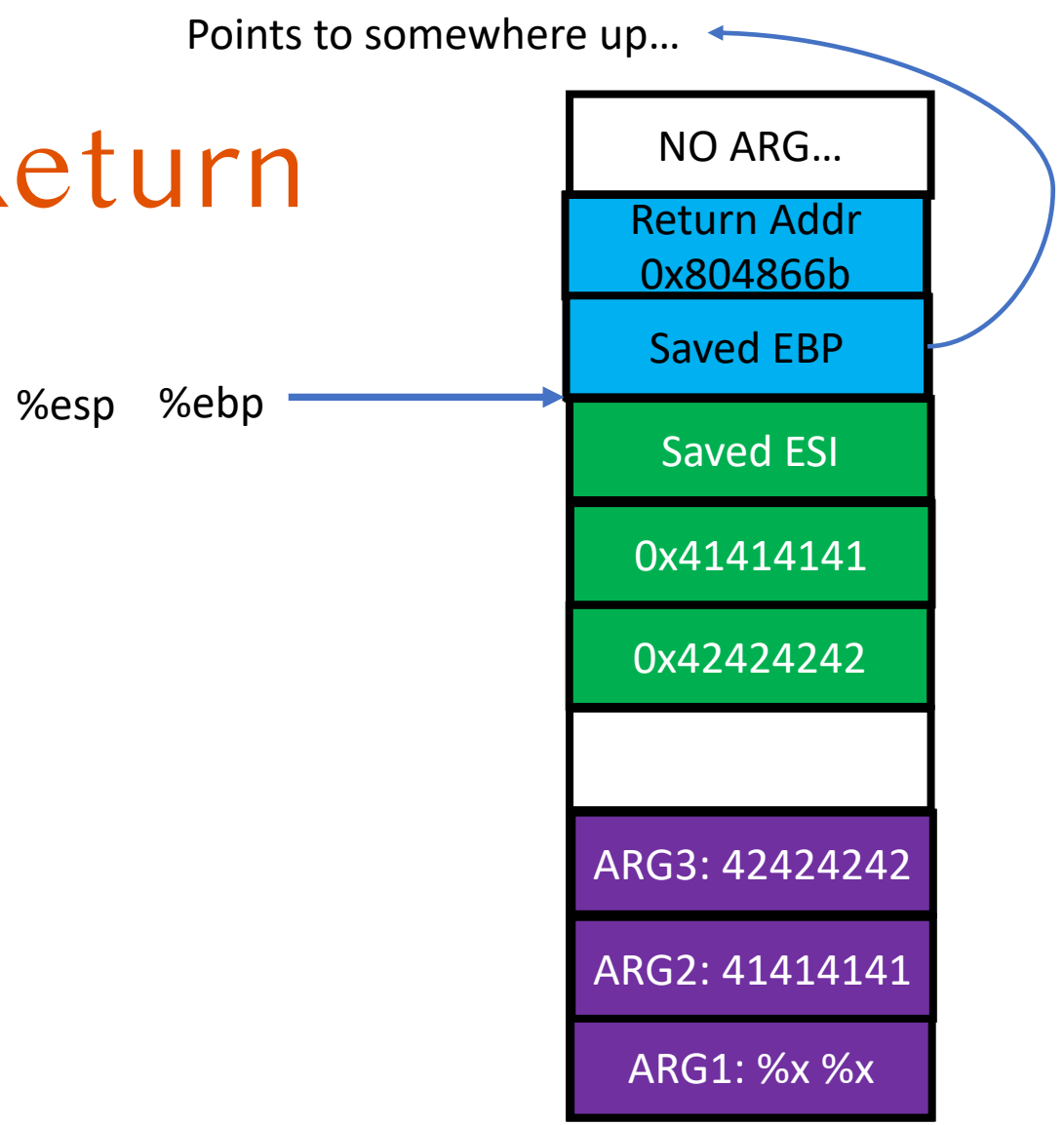
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

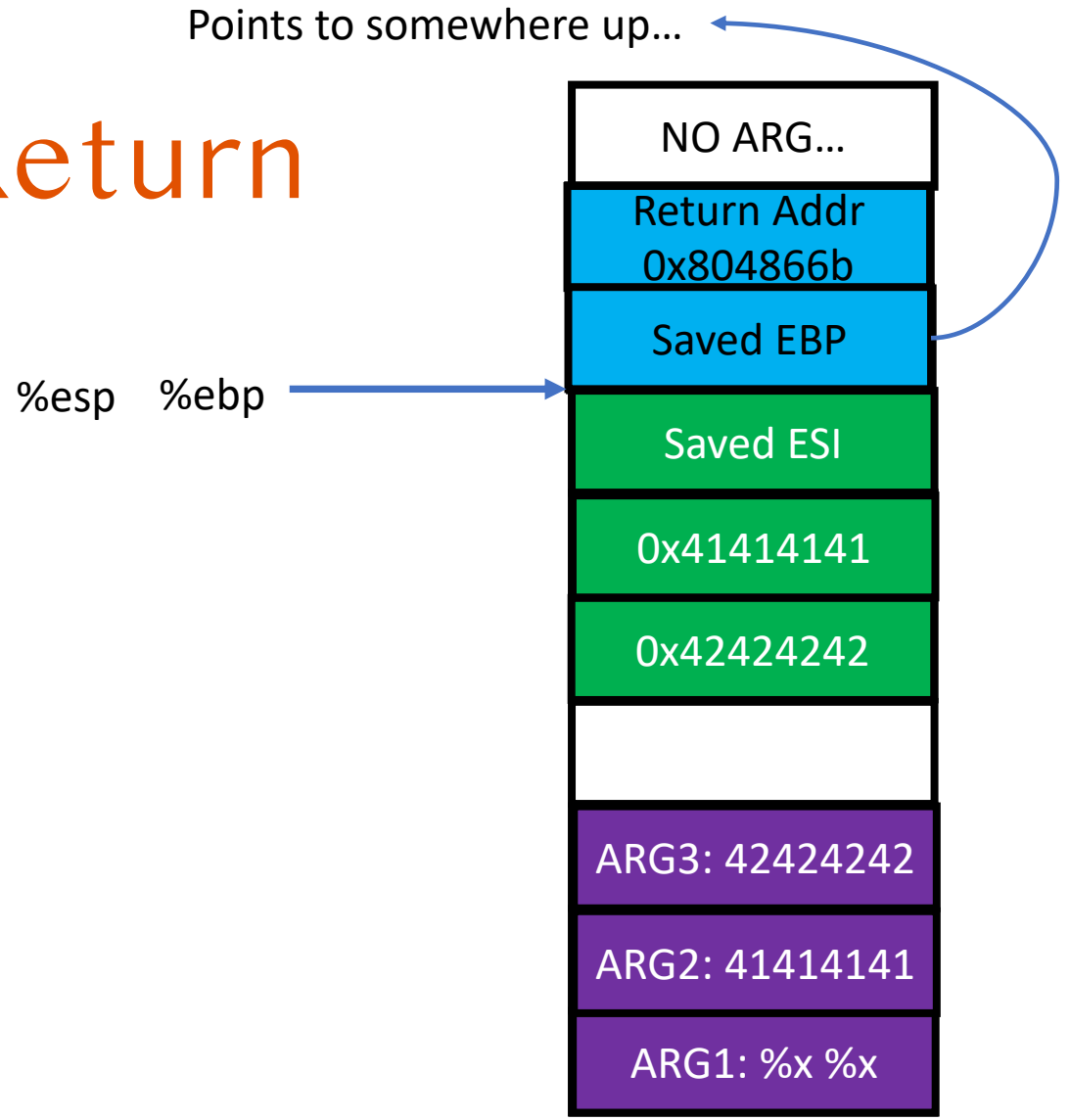
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Example – Function Return

- Function return of receive_input

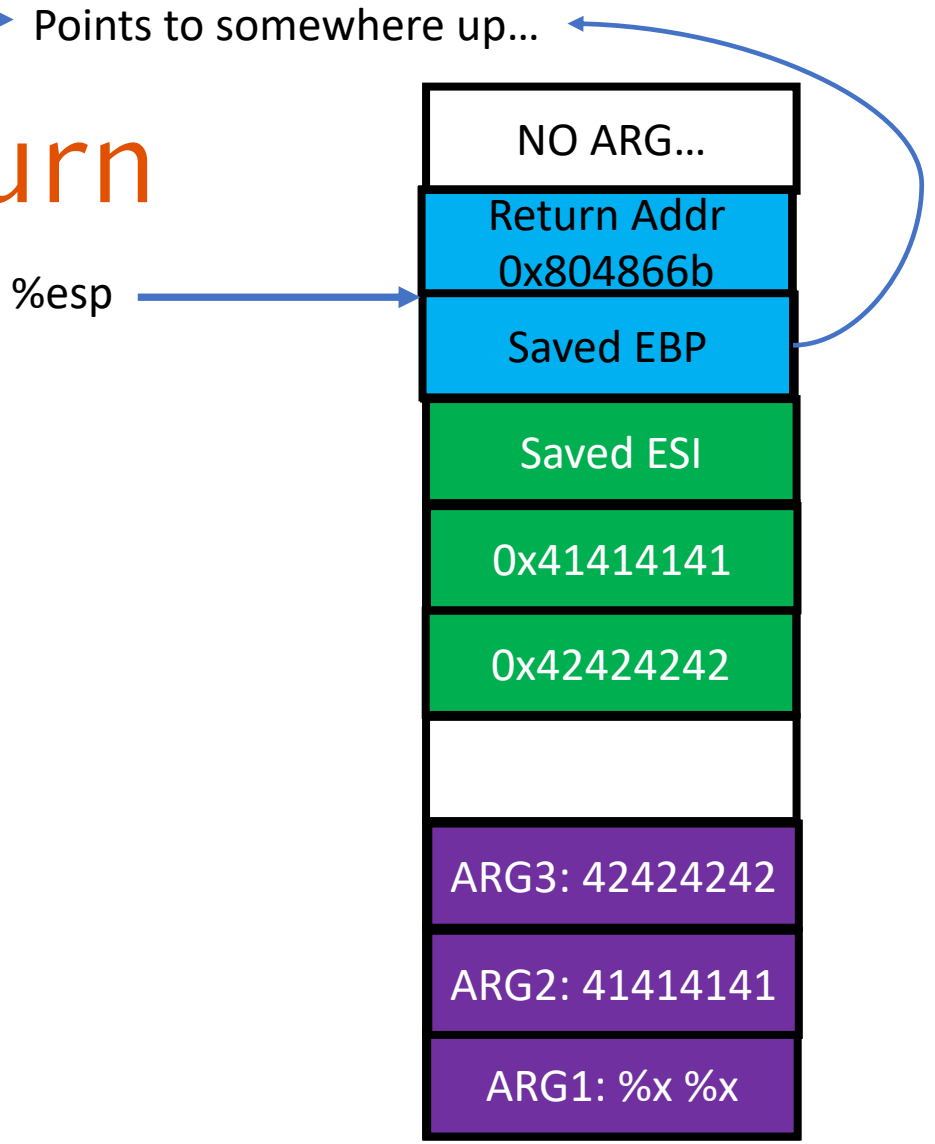
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



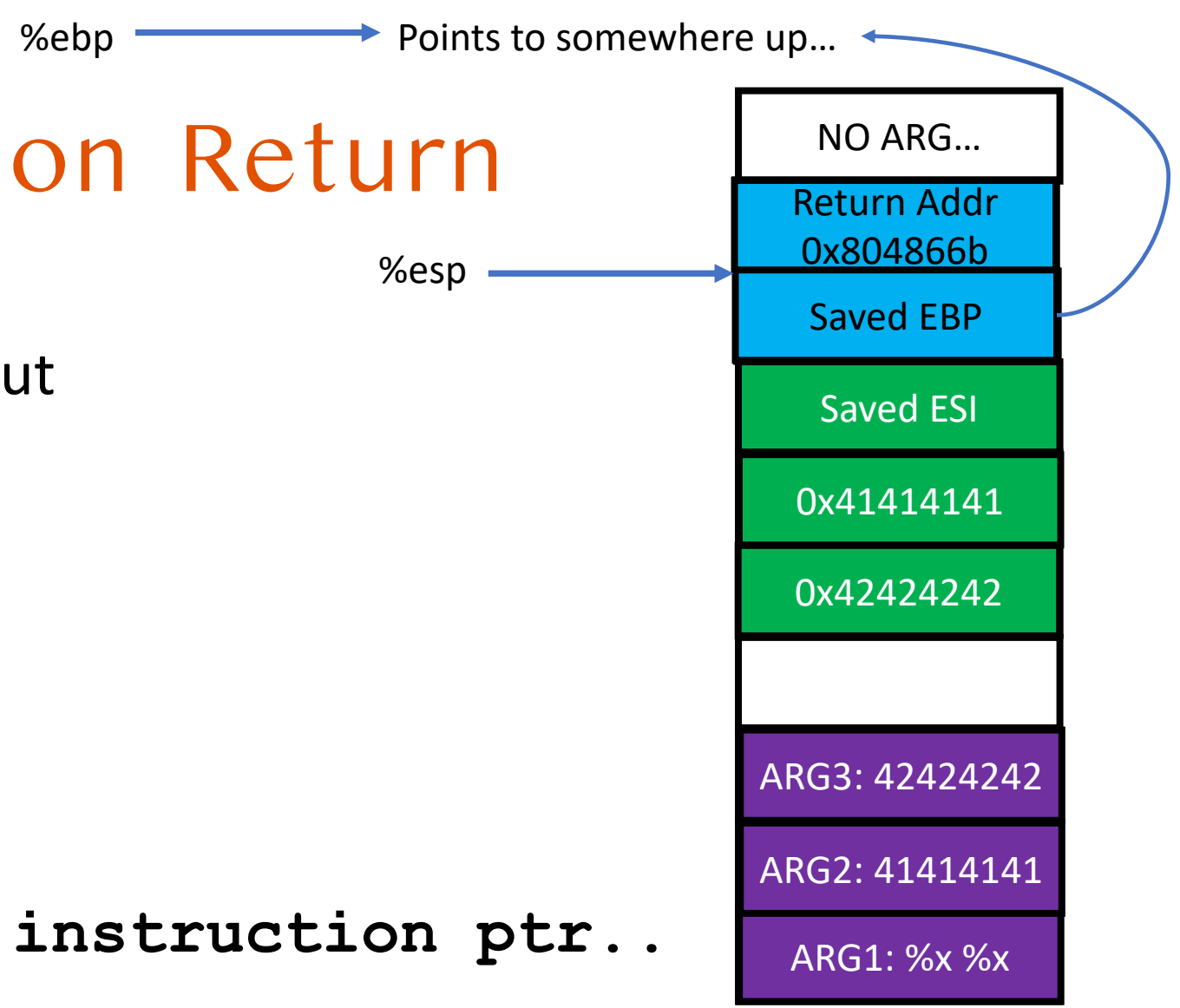
Example – Function Return

- Function return of receive_input

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



Example – Function Return

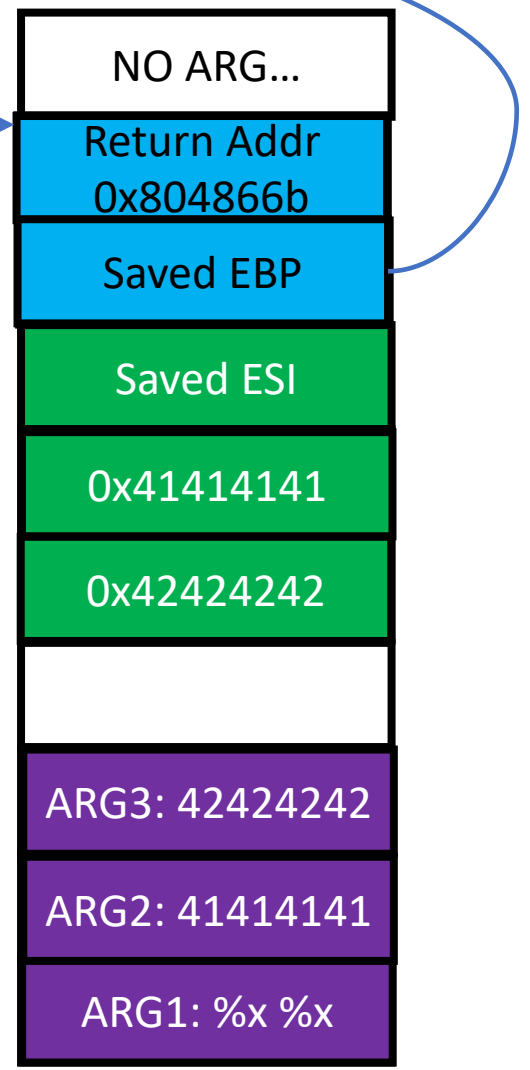


- Function return of receive_input

```
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
```

Oregon State University

Buffer Overflow

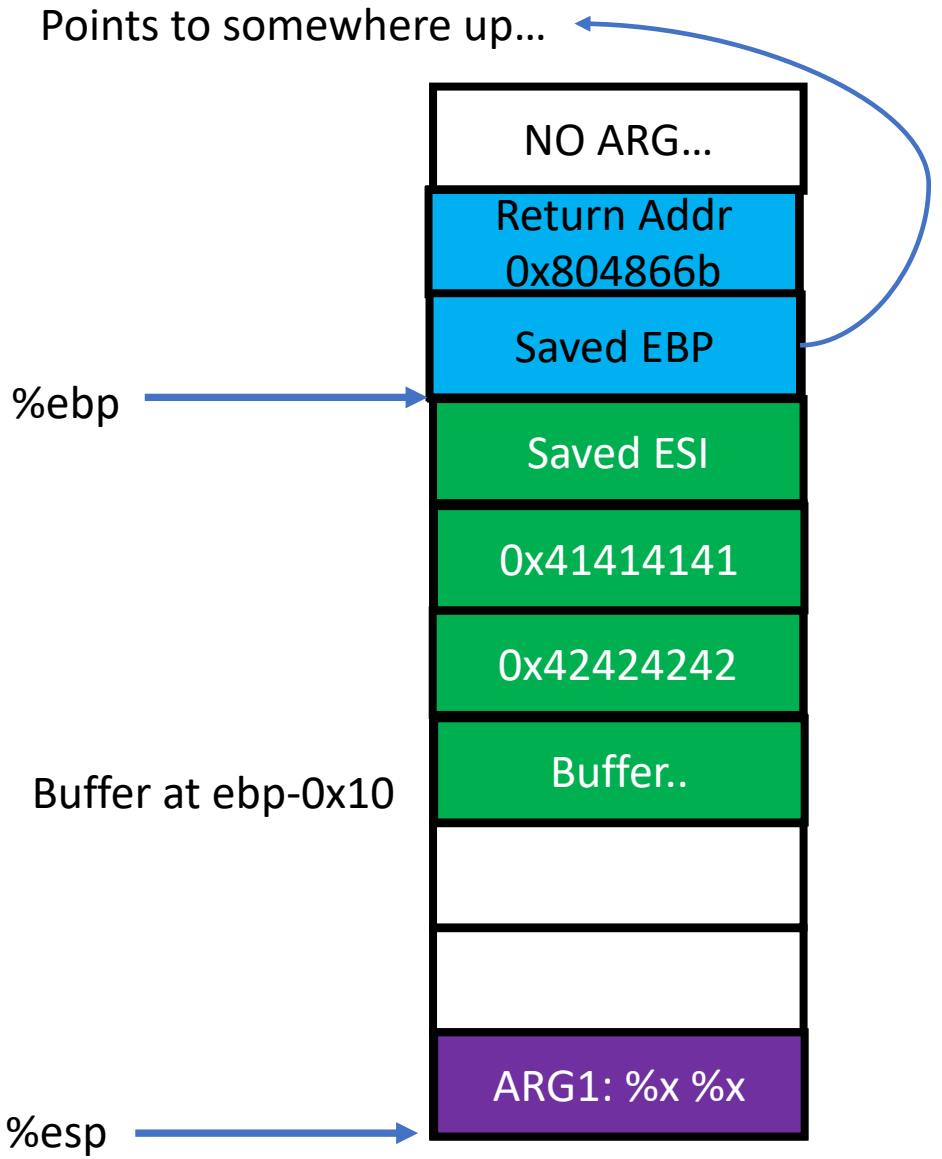
- Example

```
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);

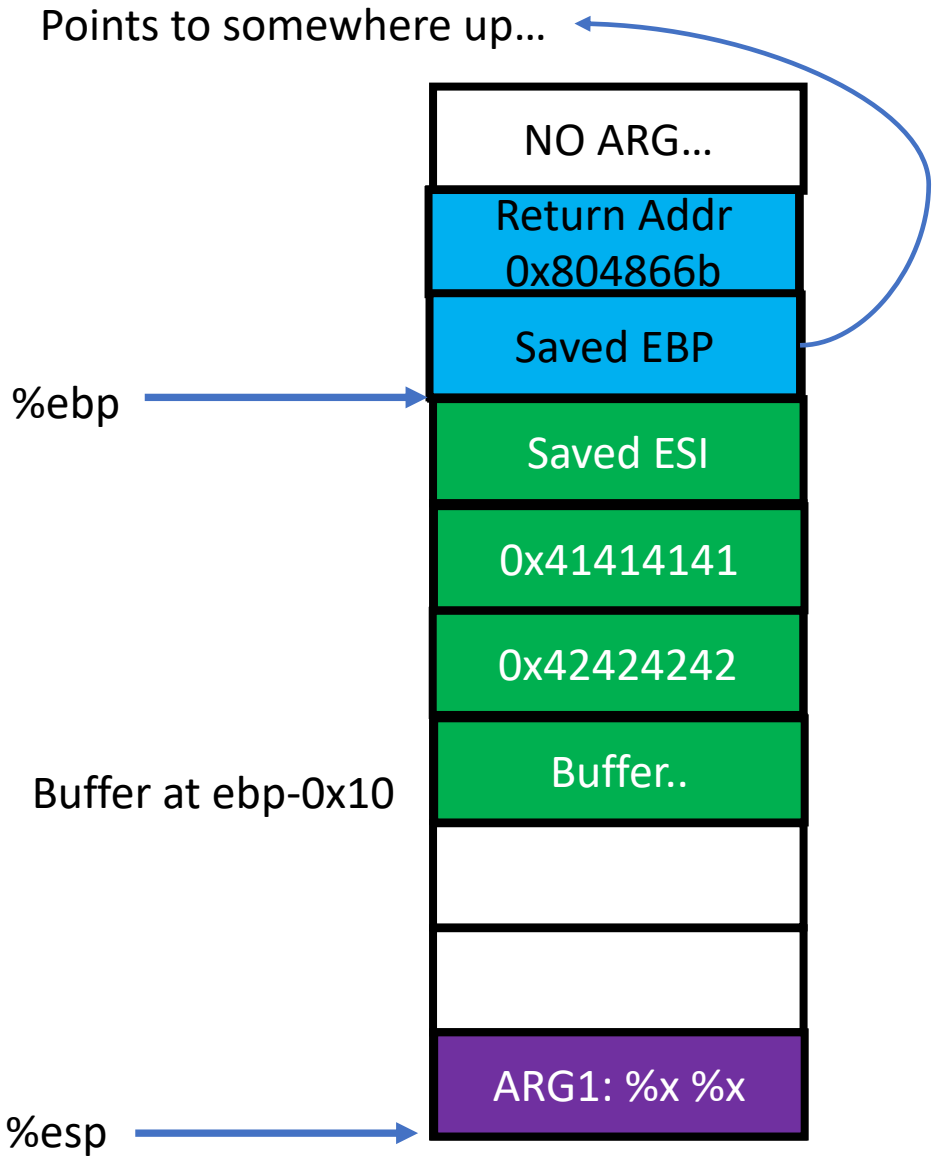
Buffer Overflow

- 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”?



ASCII CODE

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

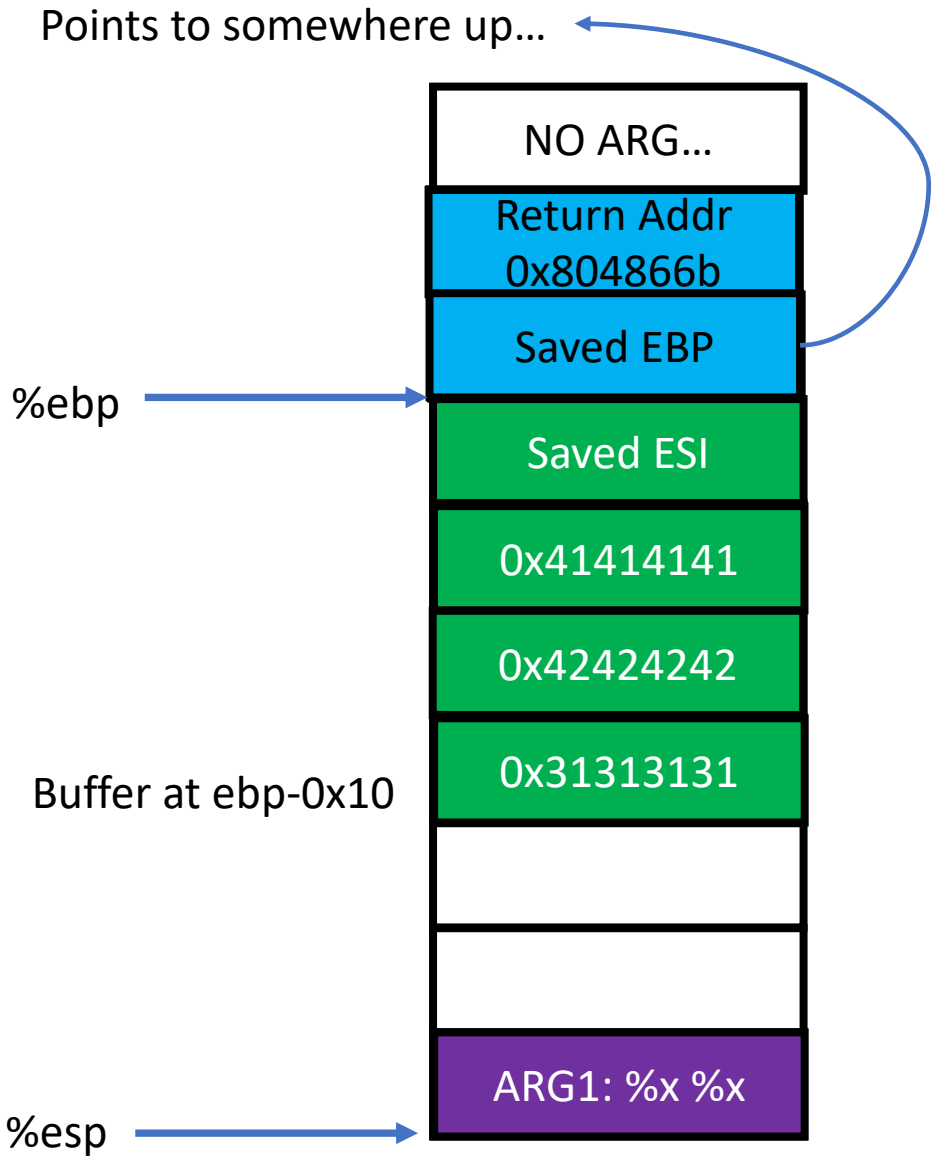
Source: www.LookupTables.com

- Image from: https://cs.smu.ca/~porter/csc/ref/ascii.html and www.LookupTables.com

Oregon State University

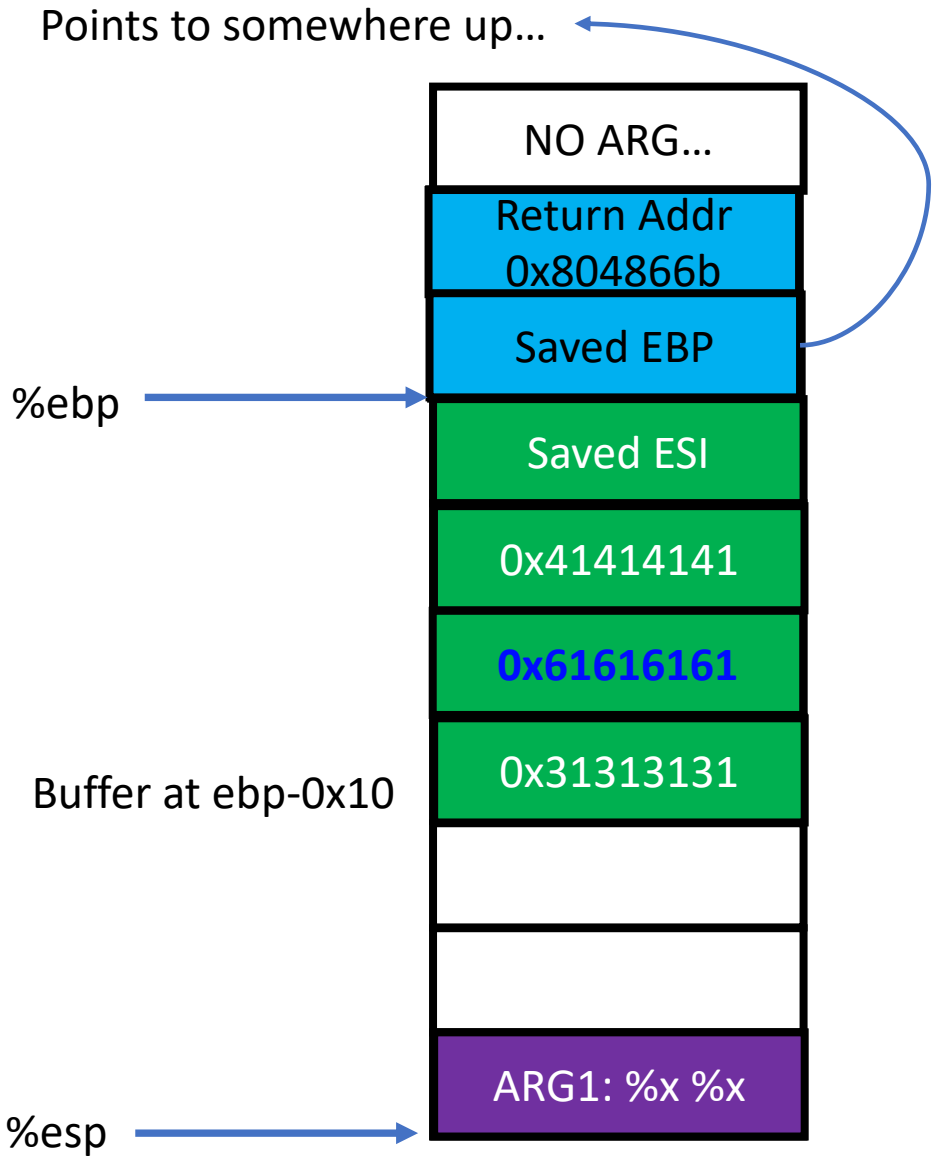
Buffer Overflow

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



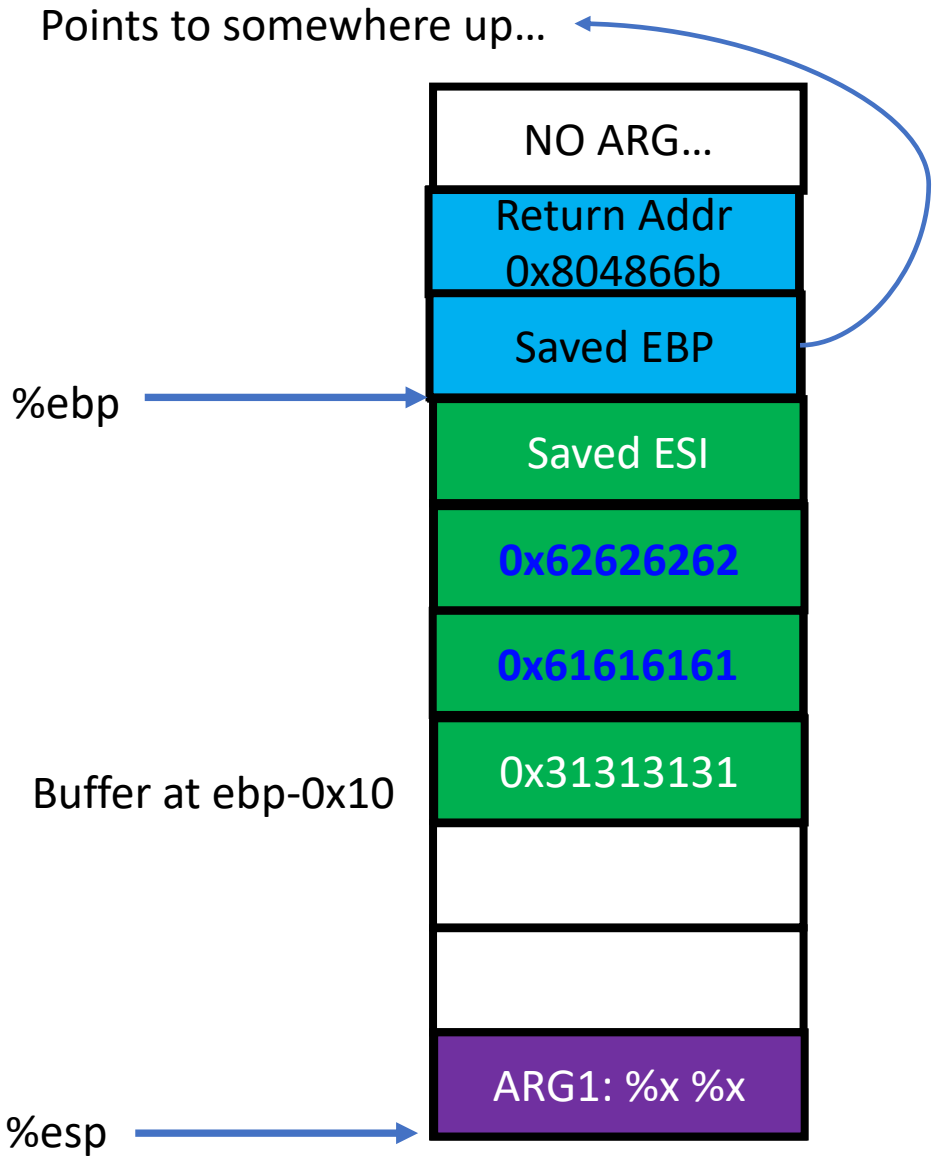
Buffer Overflow

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



Buffer Overflow

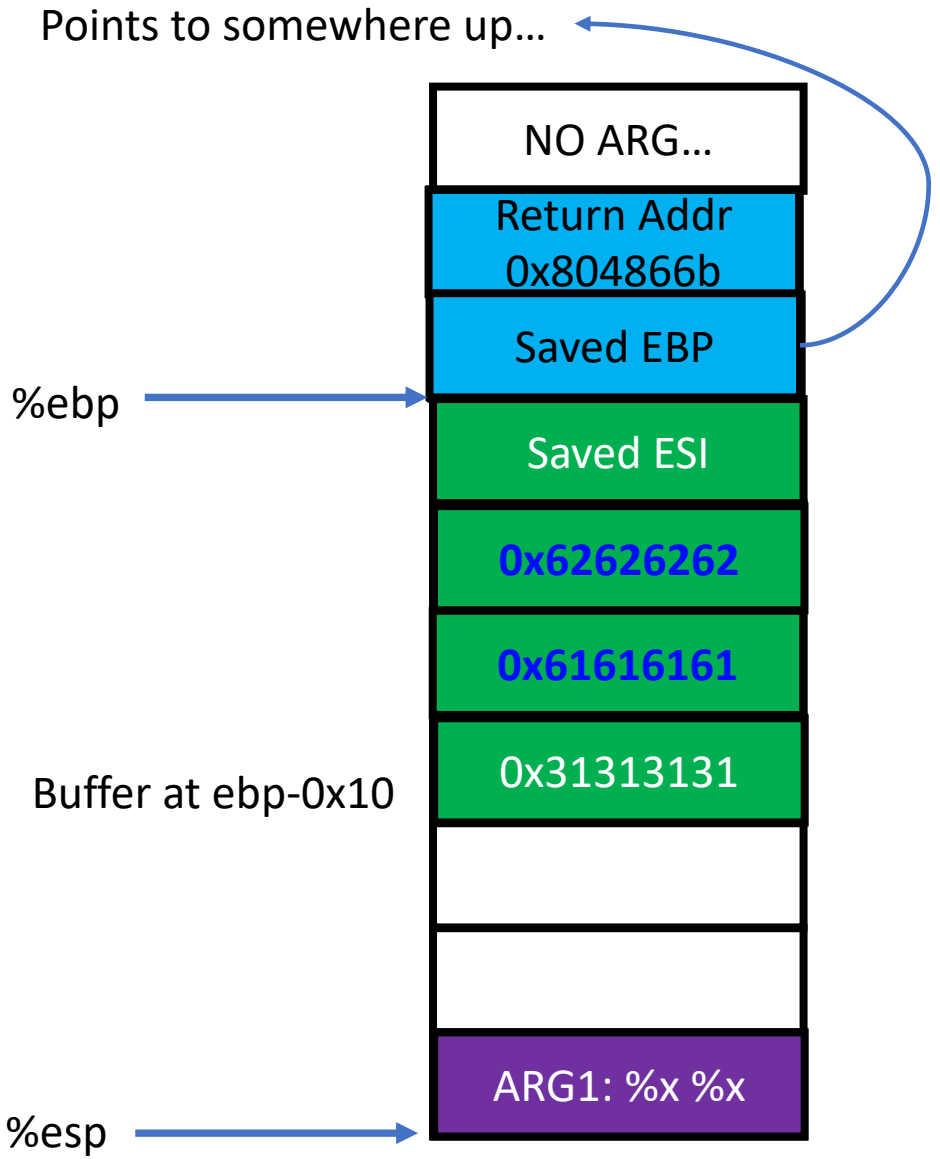
- 4 byte buffer…
- What if you type “1111aaaabbbb”?



Buffer Overflow

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

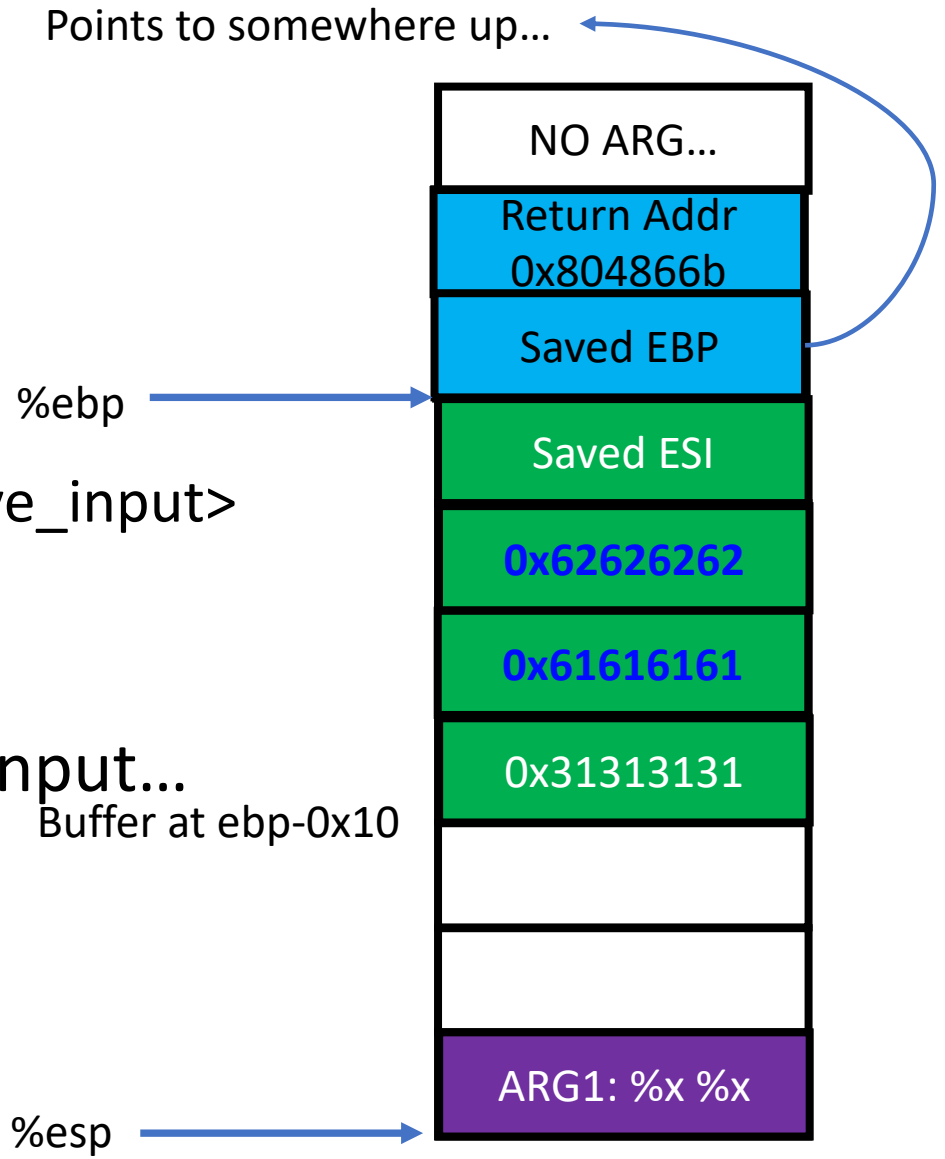
- You can change variables!



Return Address

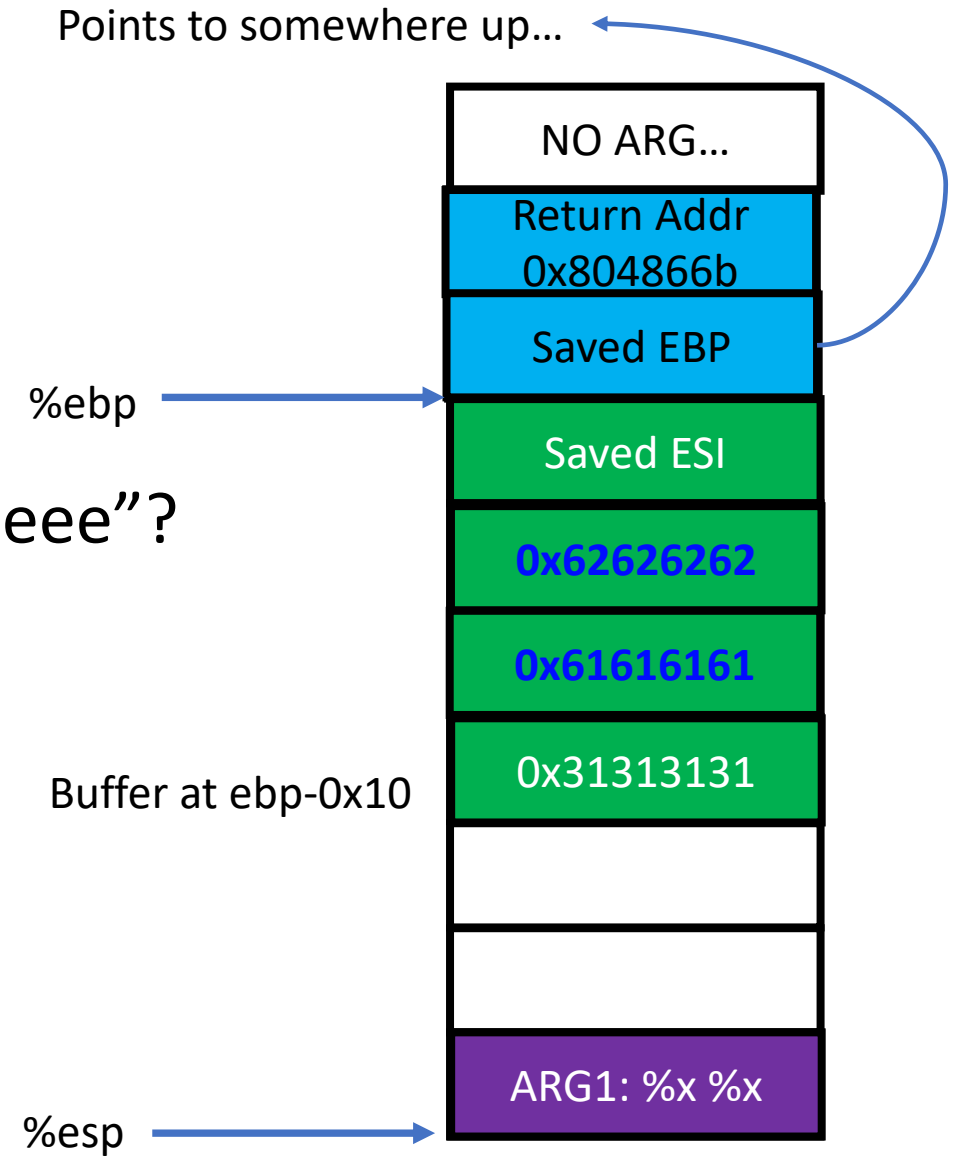
- Store the execution points after the call
  - 0x08048666 <+6>: call 0x8048570 <receive_input>
  - 0x0804866b <+11>: xor %eax,%eax

- Should store 0x804866b on calling receive_input…



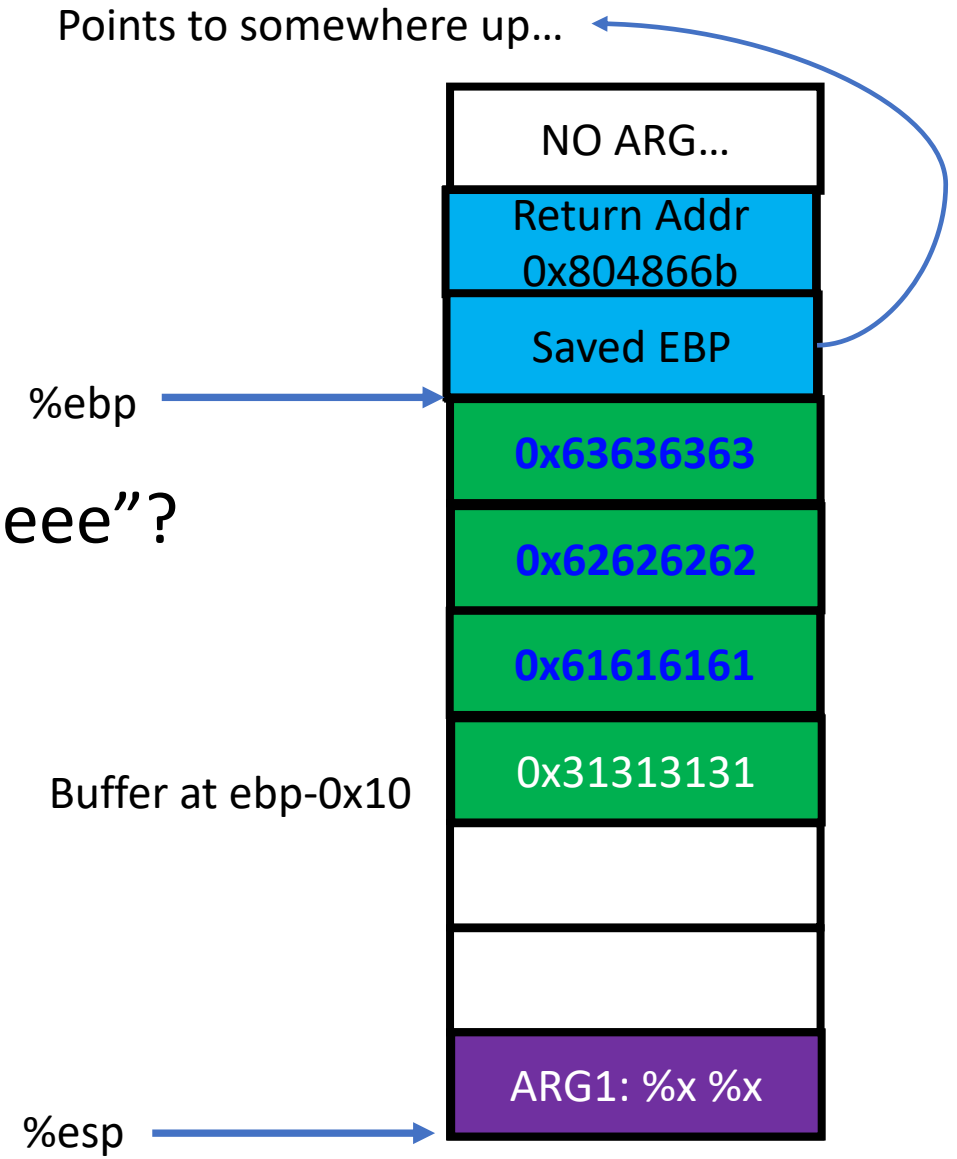
Buffer Overflow

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



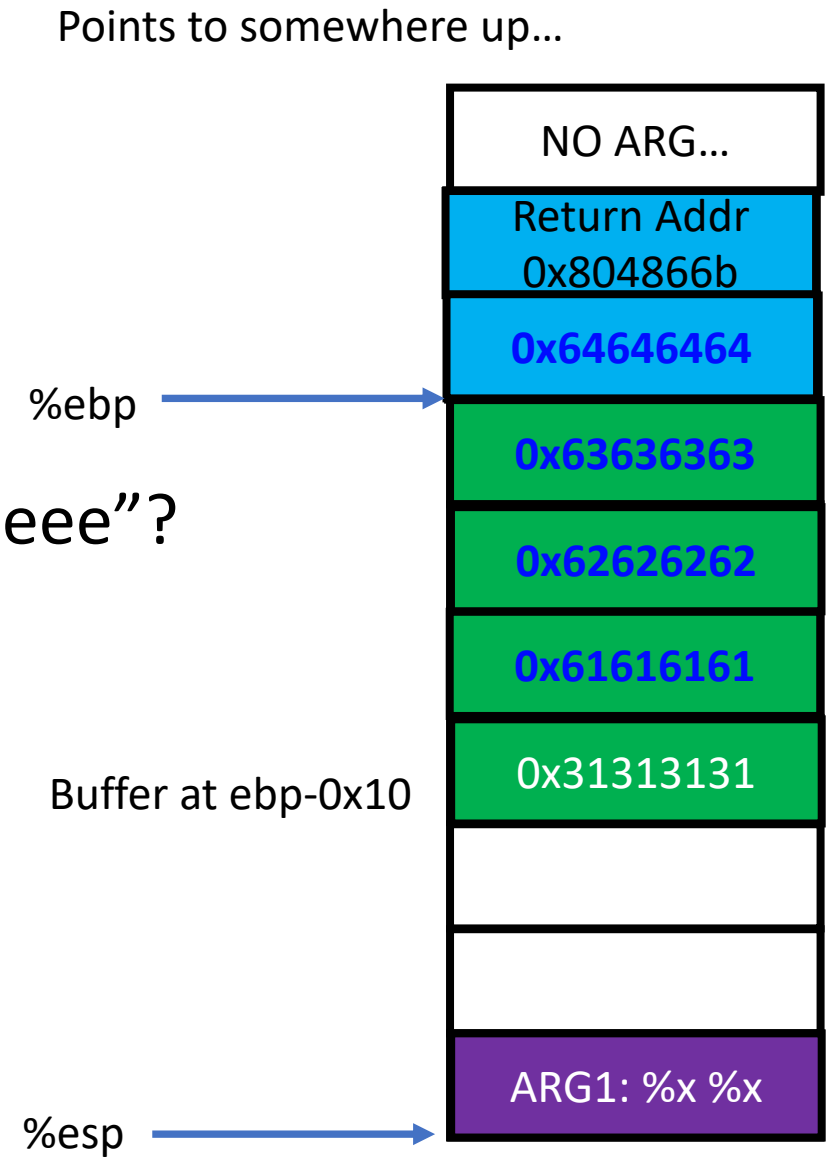
Buffer Overflow

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



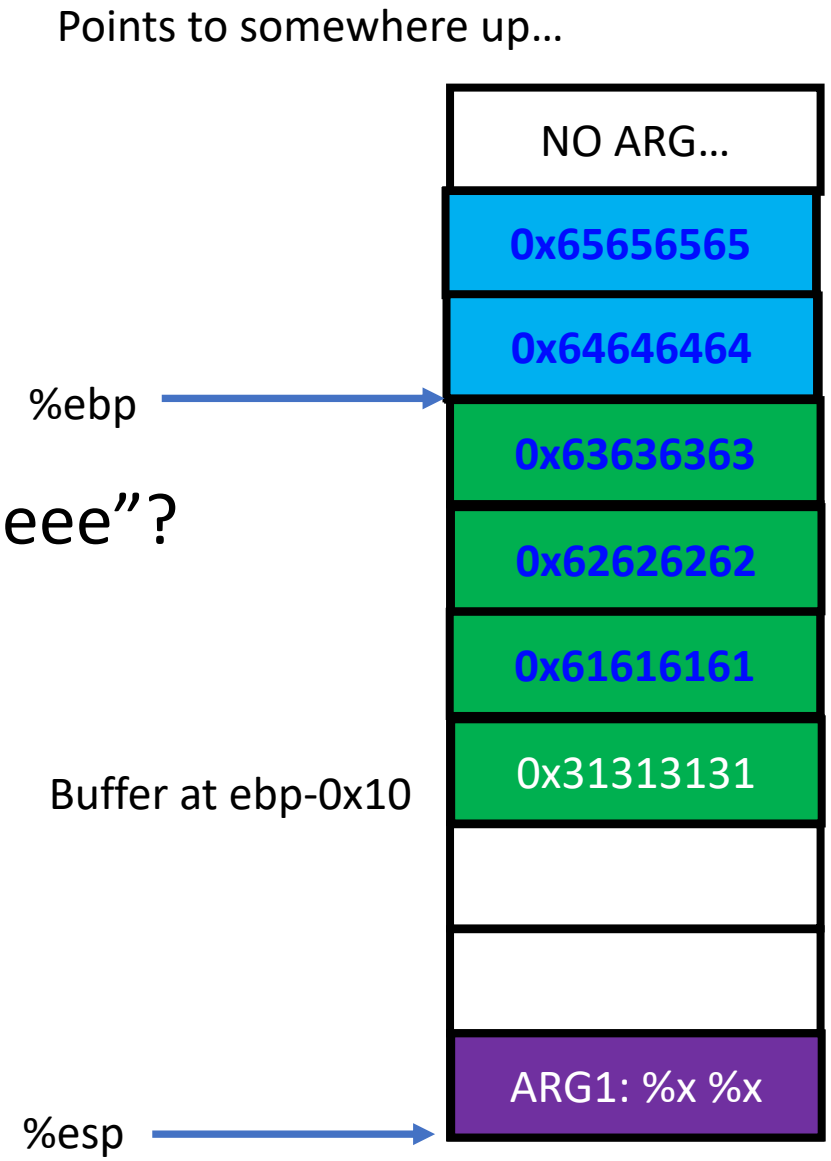
Buffer Overflow

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



Buffer Overflow

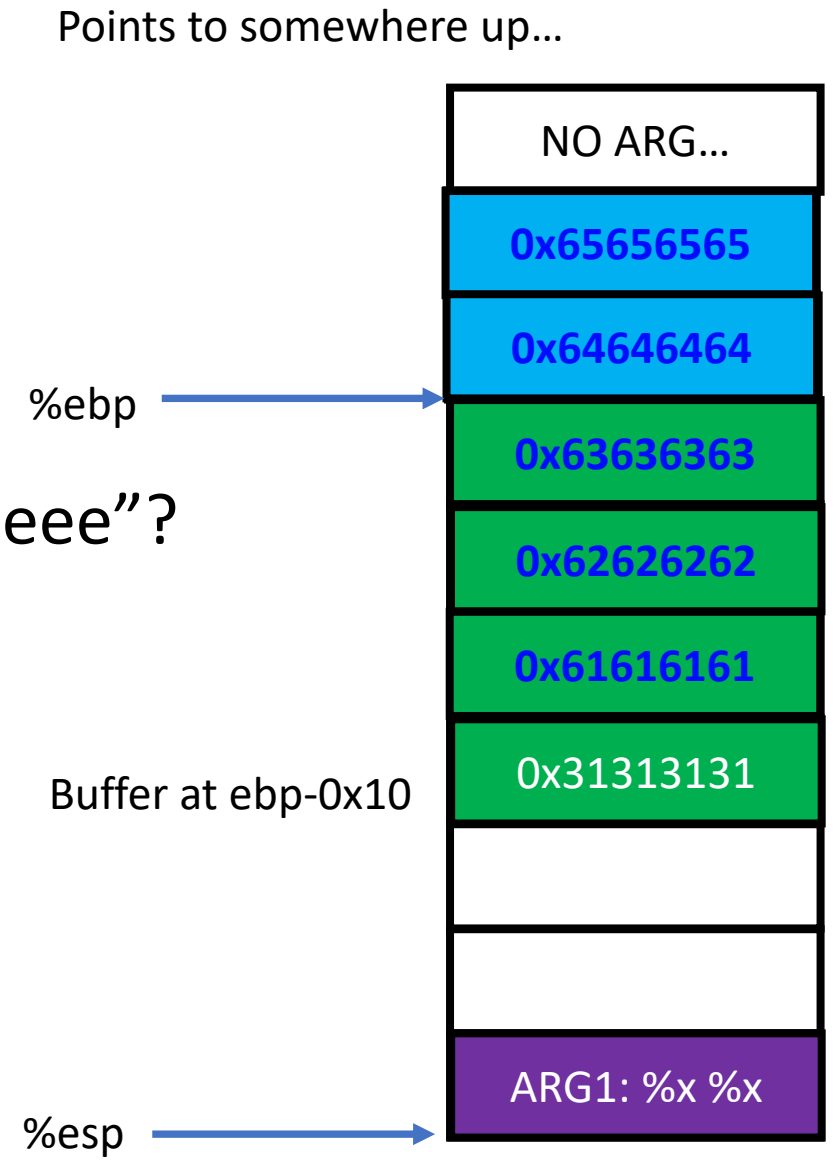
- 4 byte buffer…
- What if you type “1111aaaabbbbccccddddeeee”?



Buffer Overflow

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

- Overwrites the return address!

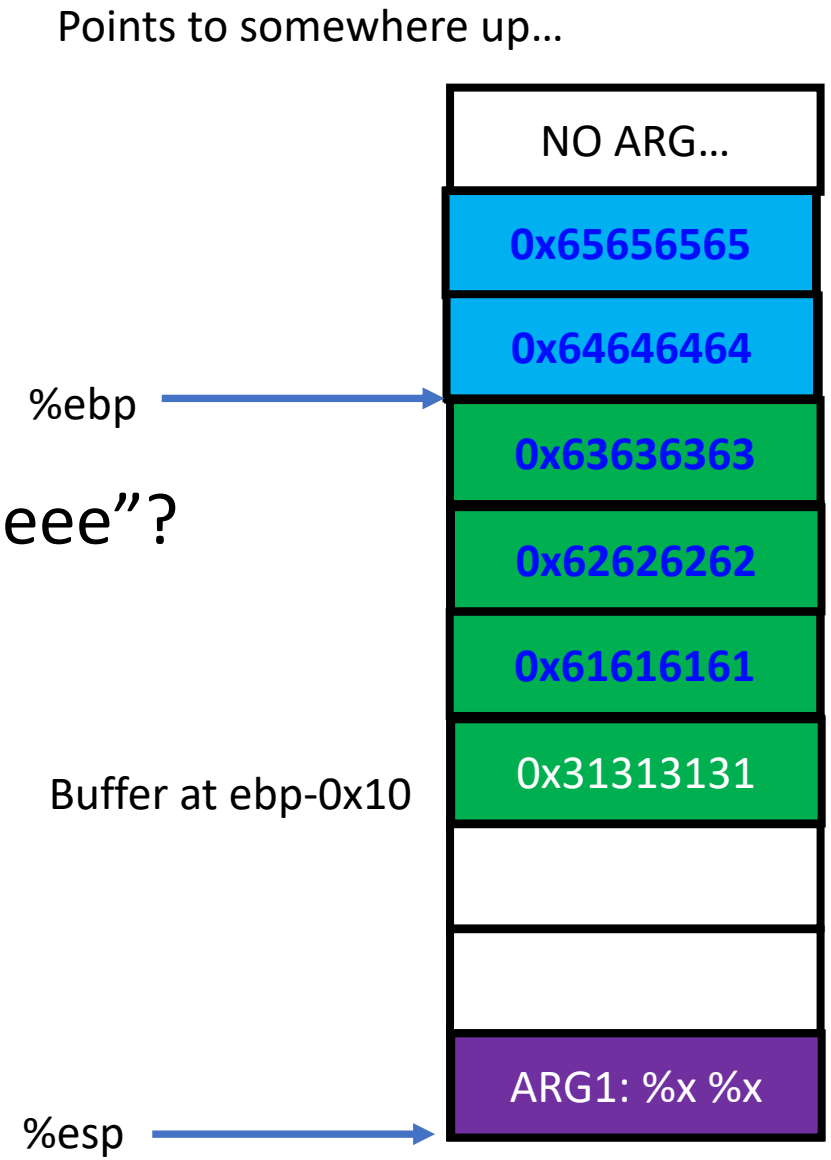


Buffer Overflow

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

- Overwrites the return address!

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

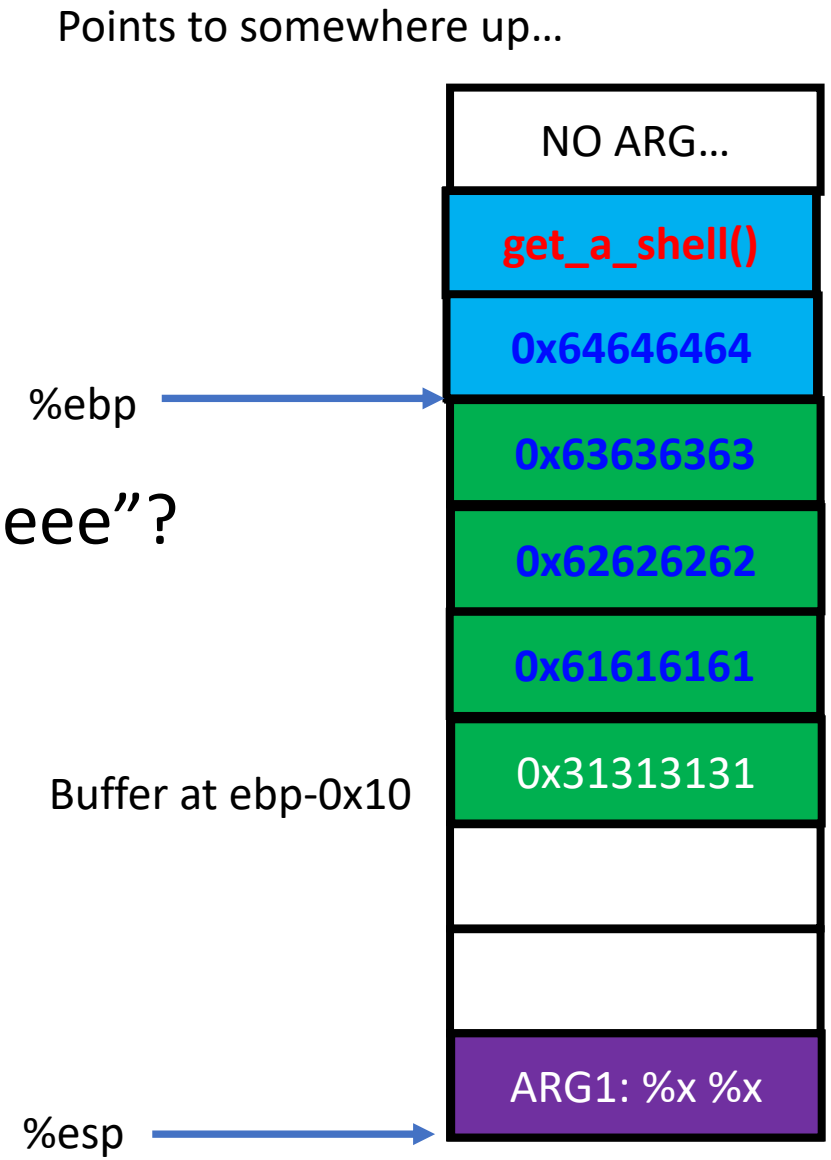


Buffer Overflow

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

- Overwrites the return address!

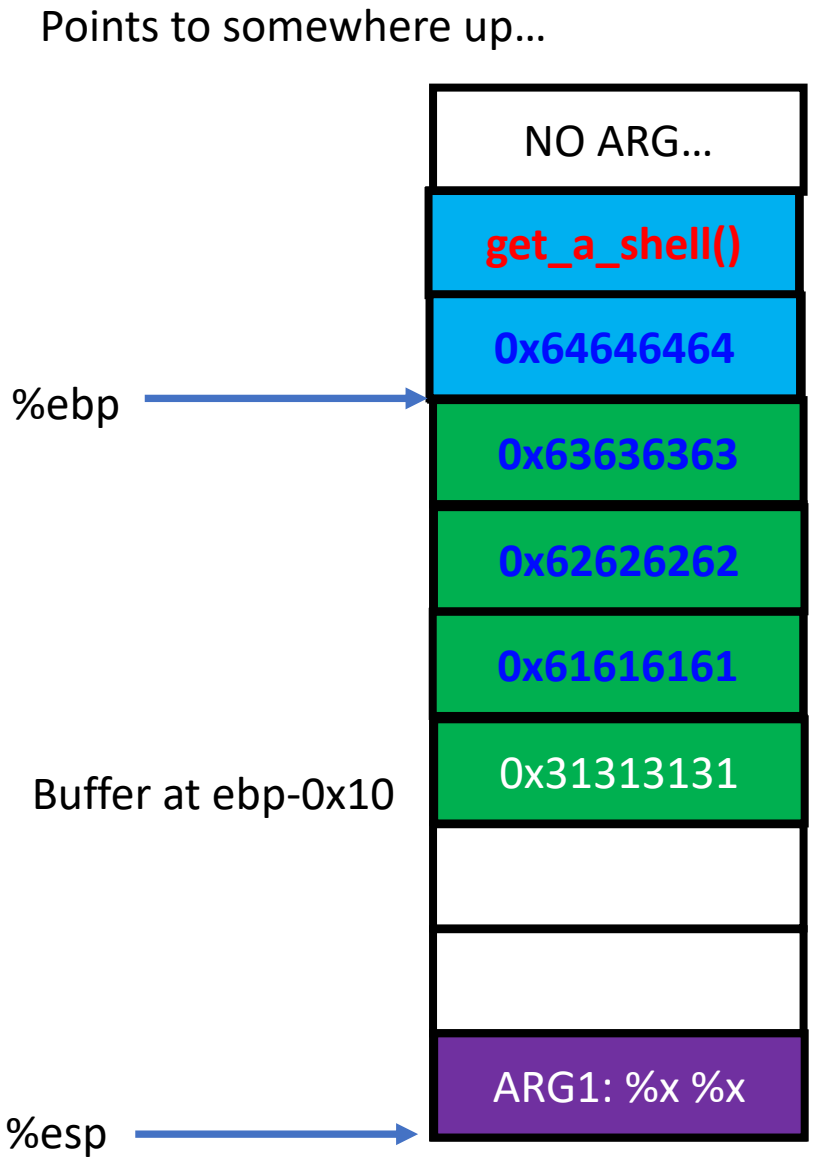
- Can you set that as the address of
  - get_a_shell()?



Buffer Overflow

- Function return of receive_input

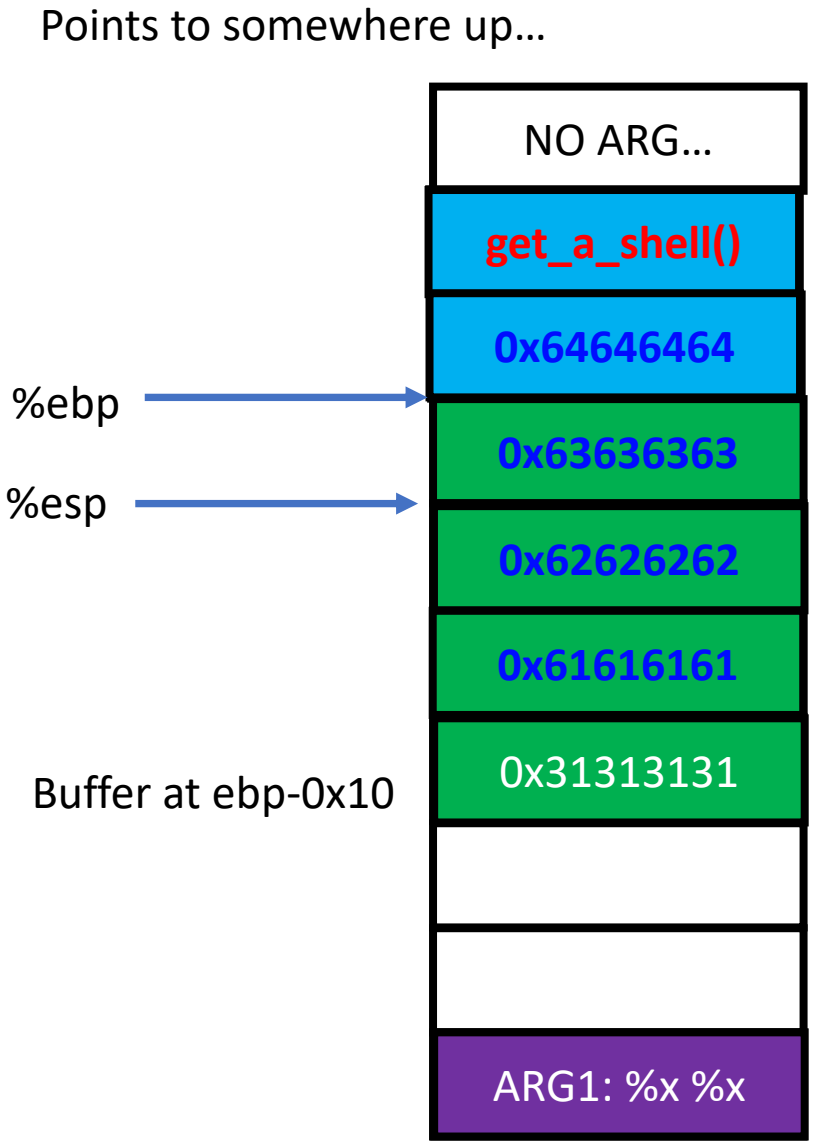
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Buffer Overflow

- Function return of receive_input

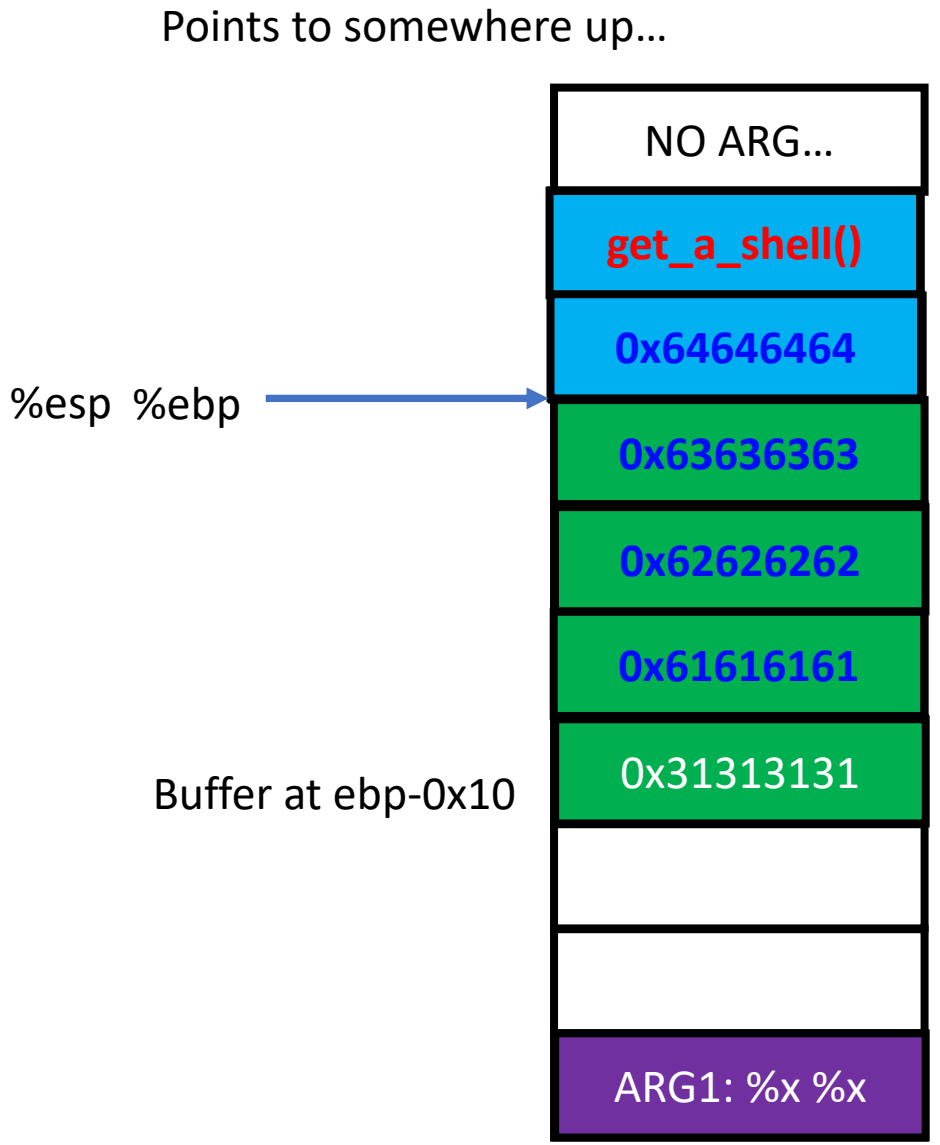
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Buffer Overflow

- Function return of receive_input

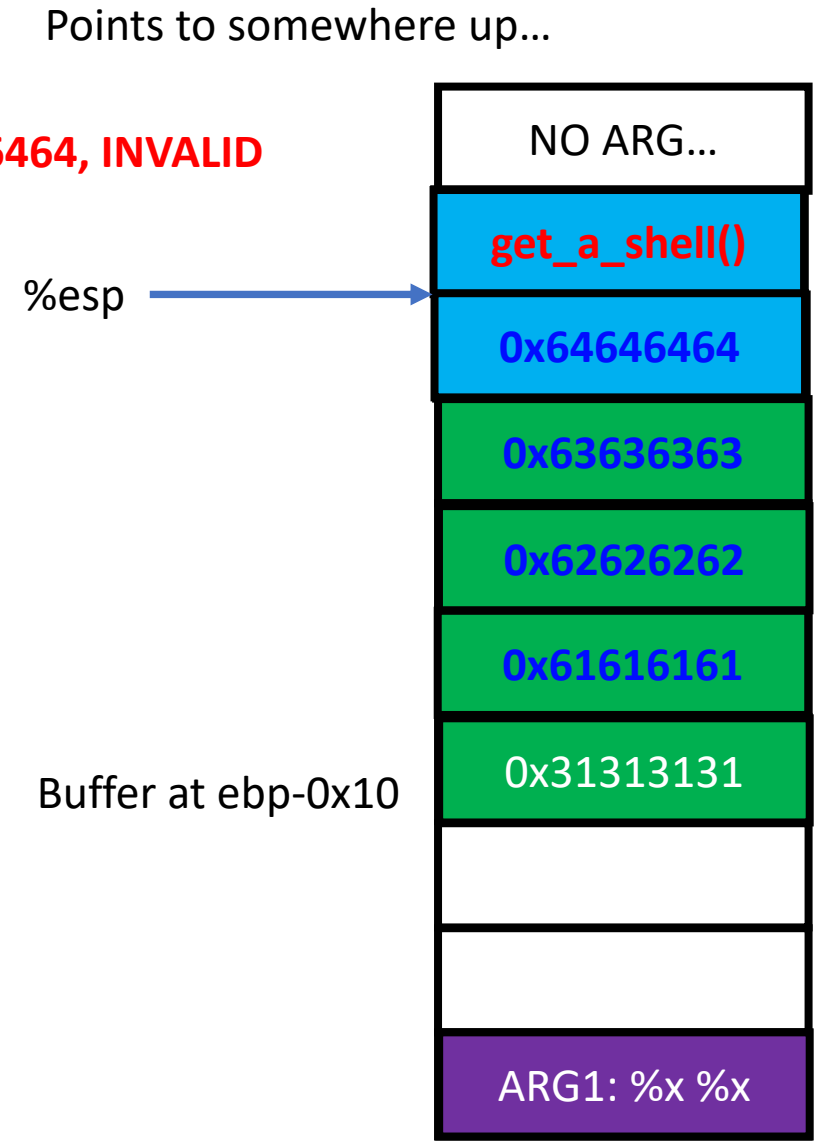
```
add     $0x54,%esp
pop     %esi
pop     %ebp
ret
```



Buffer Overflow

- Function return of receive_input

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

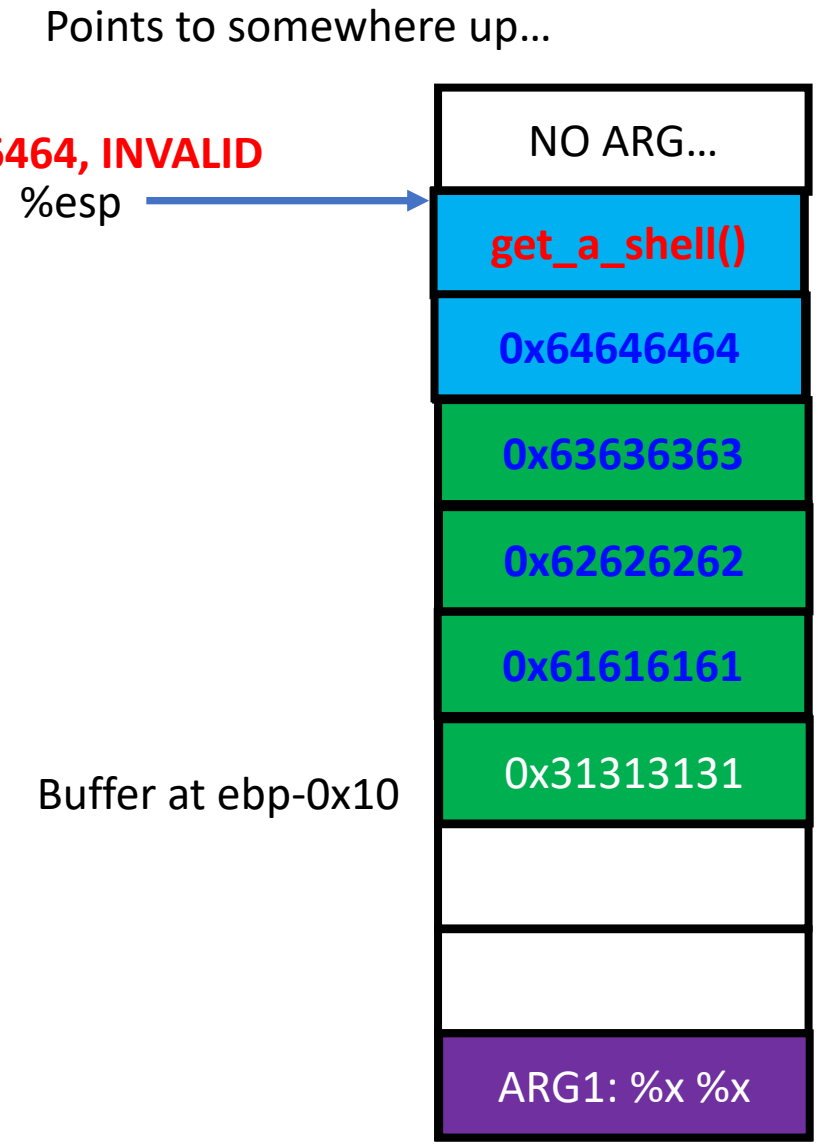


Buffer Overflow

- 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: 1/30 10:00am**