

Project 3 - BufferOverflow

Network Security

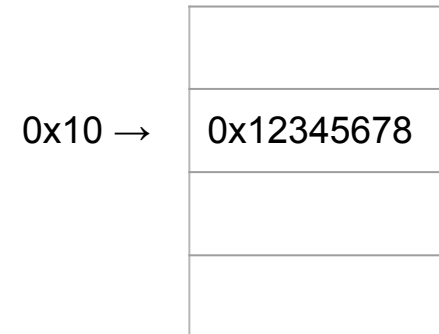
By Wei-Ti Su & Xin-Yu Wang

Outline

- Assembly Instructions
- Stack
- Stack Frame
- Buffer Overflow
- Endianness
- Non-Printable Characters
- Send payload to remote server
- More Info

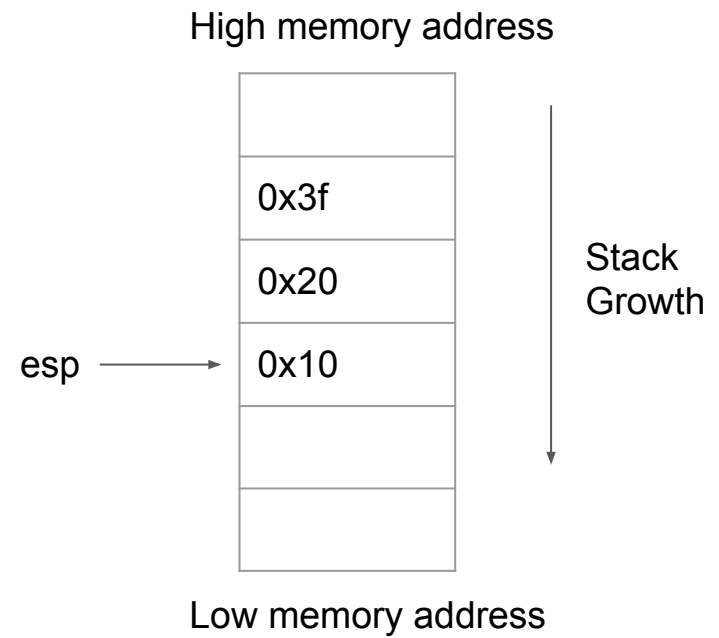
Assembly Instructions (Intel Syntax)

Opcode	Dest	Source	Result
mov	eax,	0x10	# eax: 0x10
sub	eax,	0x01	# eax: 0x0f
add	eax,	0x0f	# eax: 0x1e
lea	ebx,	[eax-0xe]	# ebx: 0x10; eax: 0x1e;
mov	ebx,	[eax-0xe]	# ebx: the value stored at 0x10 in Mem. i.e., 0x12345678



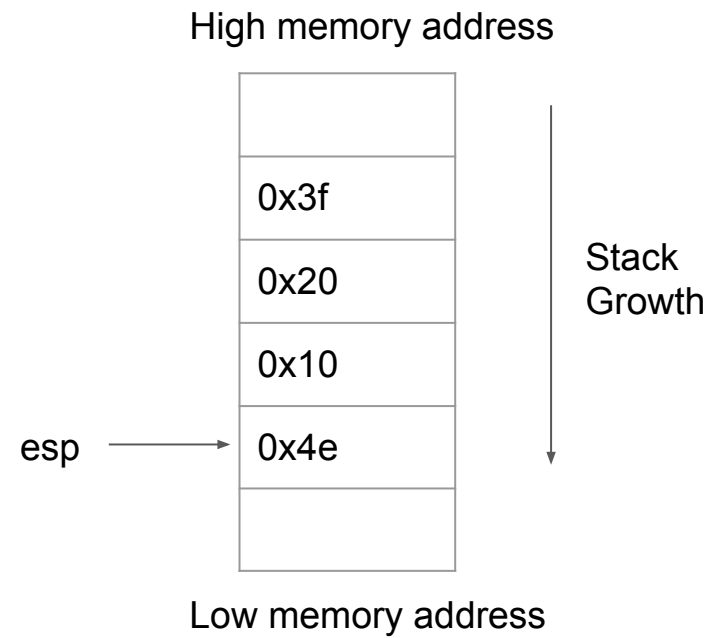
Stack

ESP(stack pointer) points to top of stack



Stack

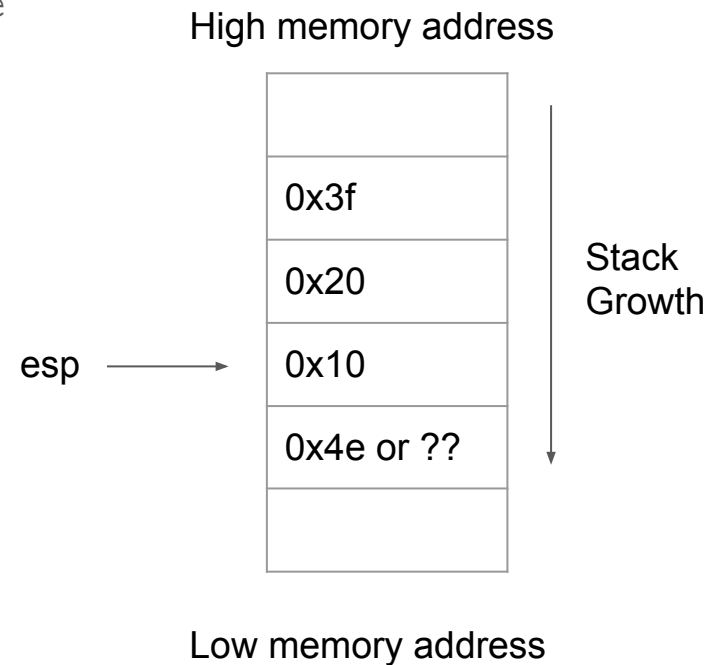
```
push    0x4e
```



Stack

```
push    0x4e
```

```
pop     eax    #now eax equals 0x4e
```

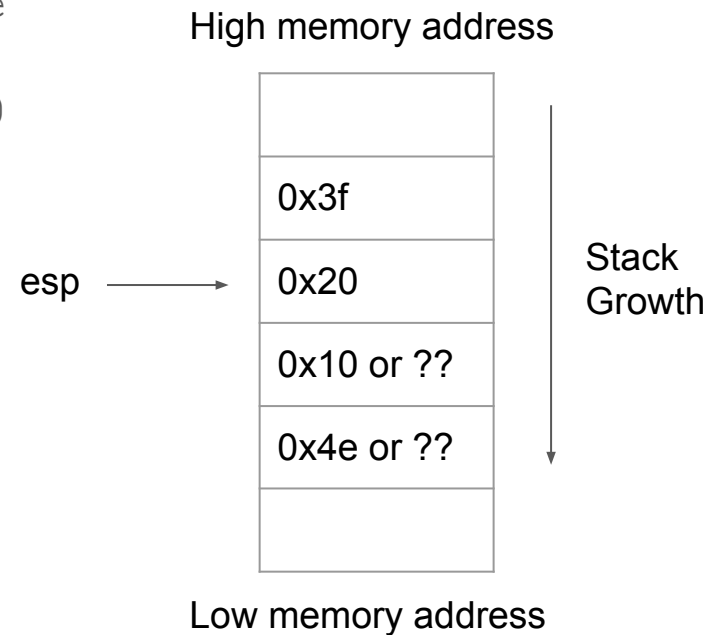


Stack

```
push    0x4e
```

```
pop     eax    #now eax equals 0x4e
```

```
pop     ebx    #now ebx equals 0x10
```



Stack Frame

example: main calls foo

1. Do stuff in main

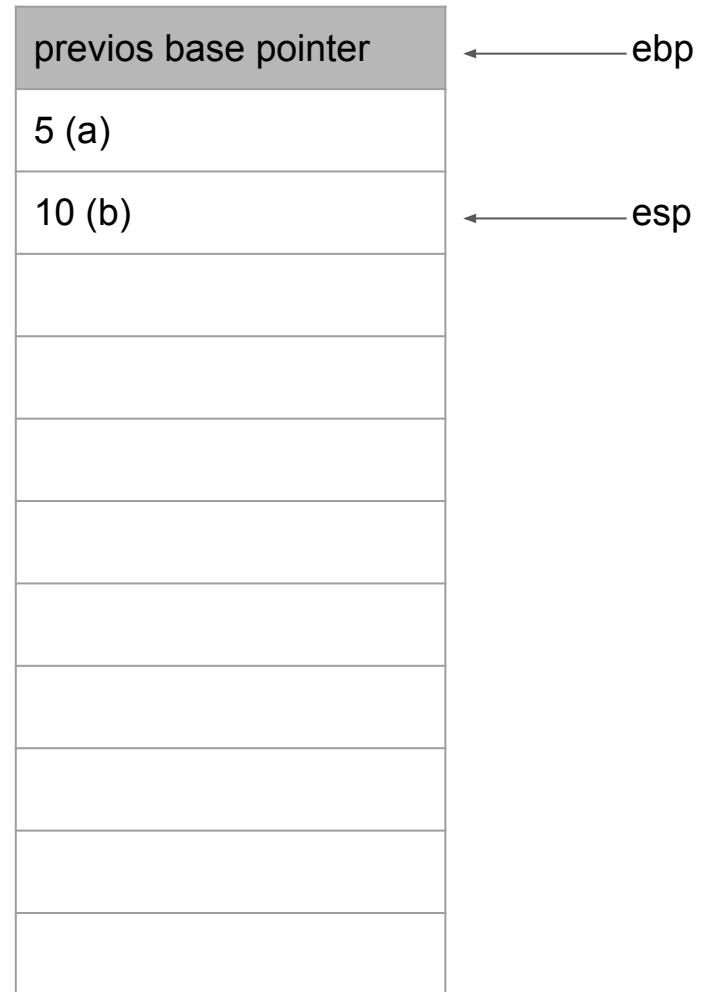
```
1 int main(){  
2     int a = 5;  
3     int b = 10;  
4 }
```

ex:

int a =5; (push 5)

int b = 10; (push 10)

High memory address



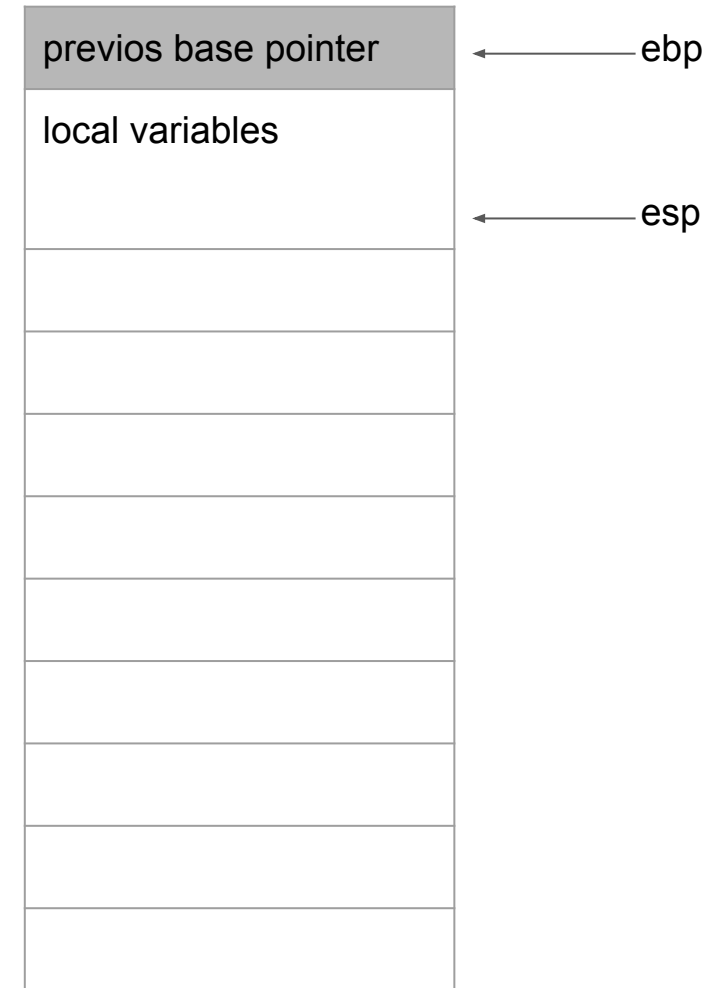
Low memory address

Stack Frame

example: main calls foo

1. Do stuff in main

High memory address



Low memory address

Stack Frame

example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo

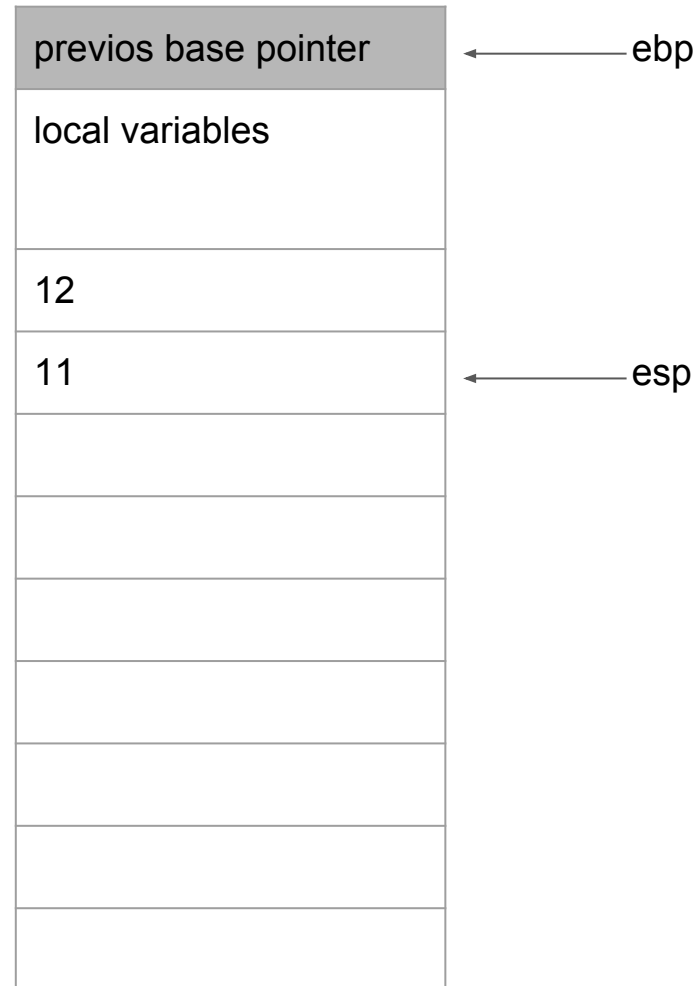
```
1 int main(){  
2     int a = 5;  
3     int b = 10;  
4     foo(11, 12);  
5 }
```

ex:

foo takes two intergers

in main: foo(11, 12); (push 12, push 11)

High memory address



Low memory address

Stack Frame

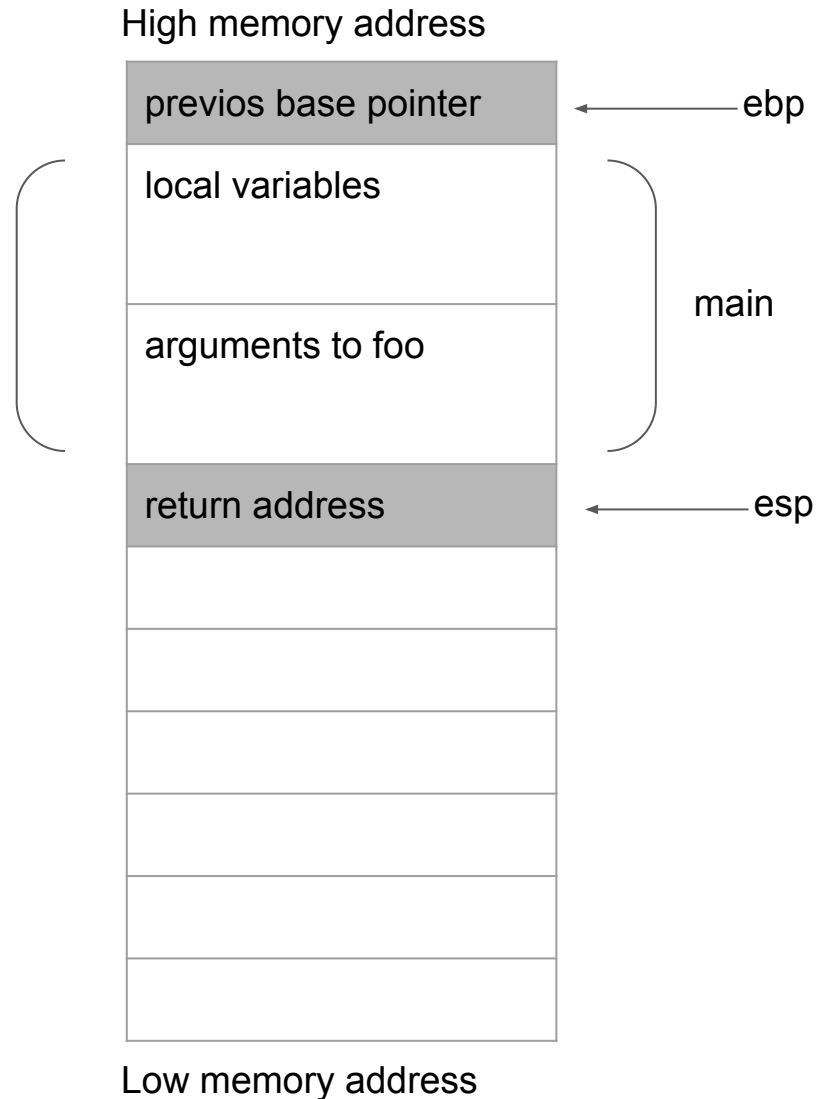
example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo

assembly: `call foo`

is equivalent to

```
push eip; #return address
mov eip,  address of foo()
```



Stack Frame

example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo

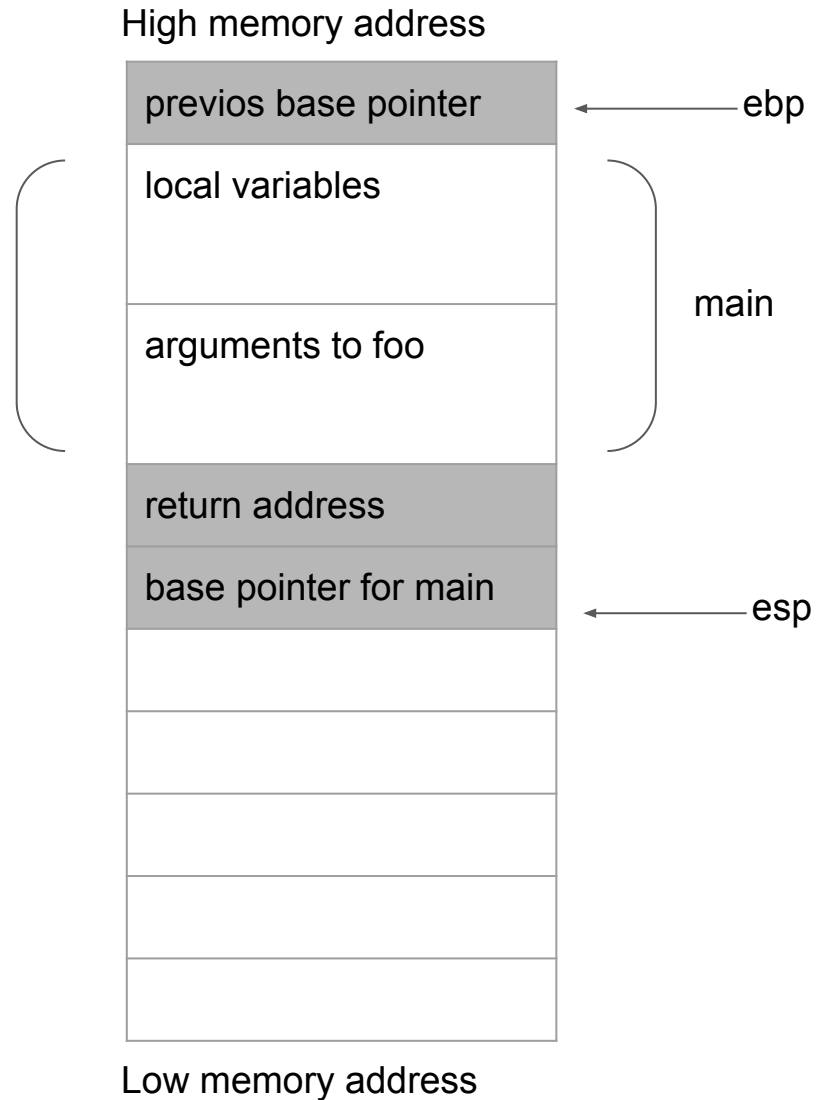
assembly:

```
call foo
```

```
...
```

```
foo:
```

```
push    ebp
```



Stack Frame

example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo

assembly:

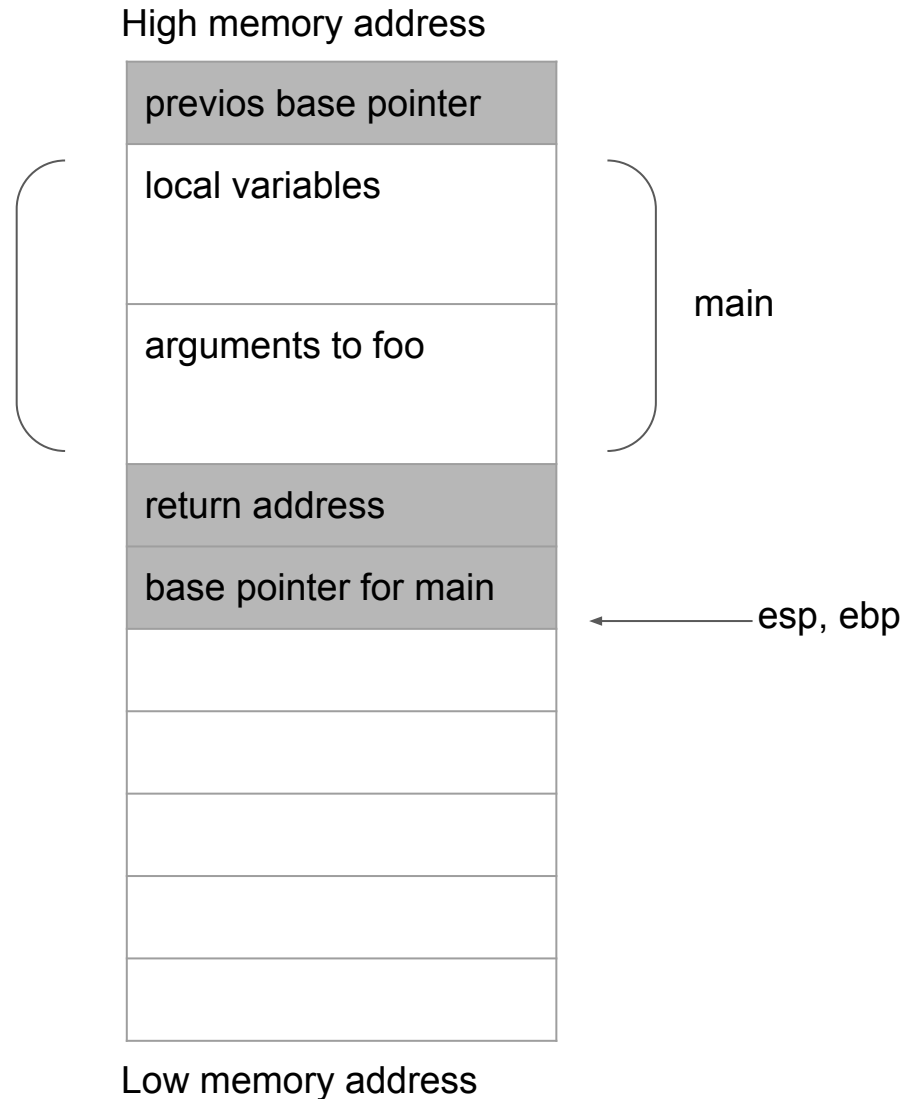
```
call foo
```

```
...
```

```
foo:
```

```
push    ebp
```

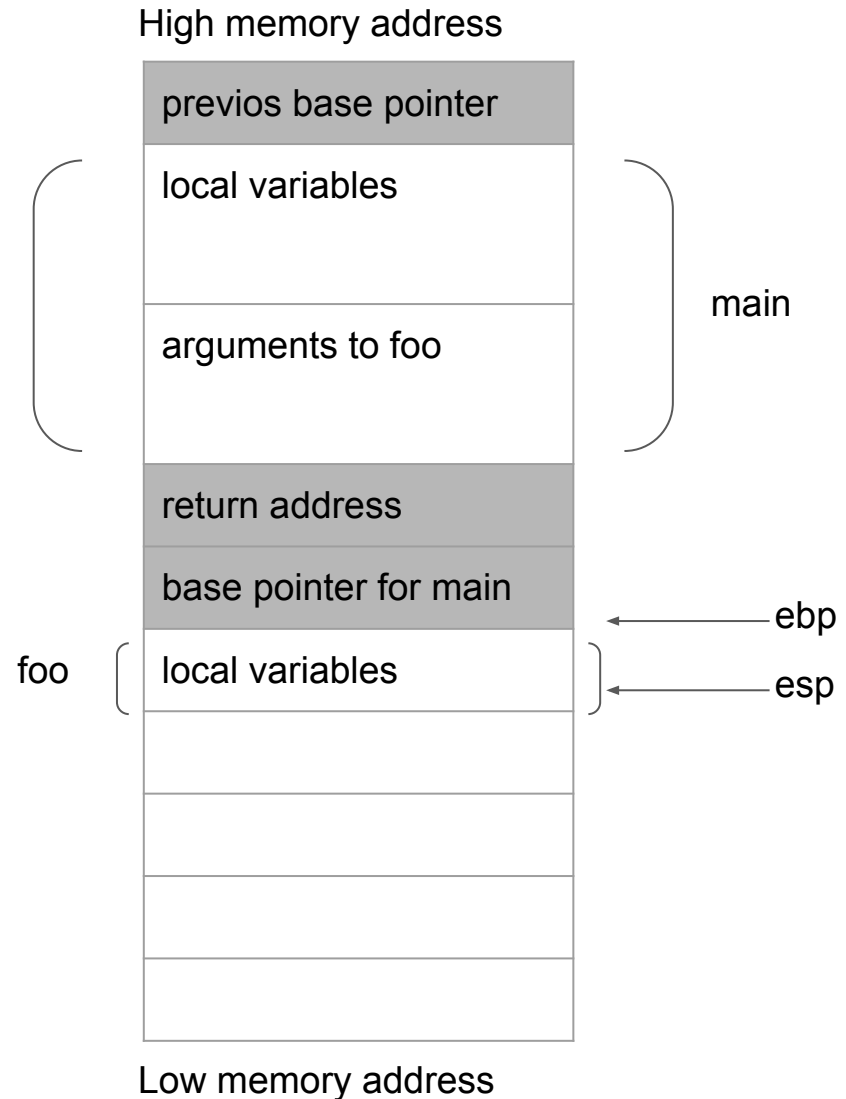
```
mov     ebp, esp
```



Stack Frame

example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo
4. Do stuff in foo



Stack Frame

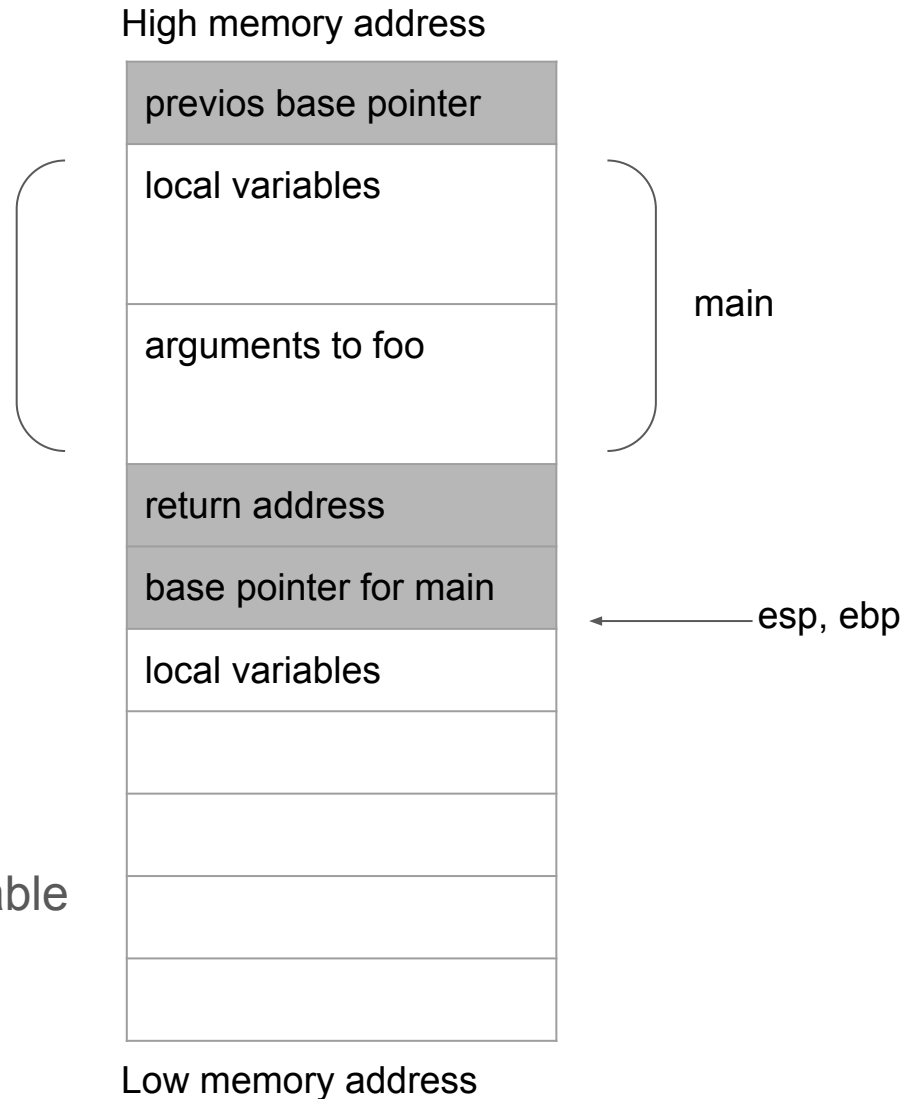
example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo
4. Do stuff in foo
5. Return to main

assembly: `leave`

is equivalent to

```
mov esp, ebp ← delete all local variable
pop ebp
```



Stack Frame

example: main calls foo

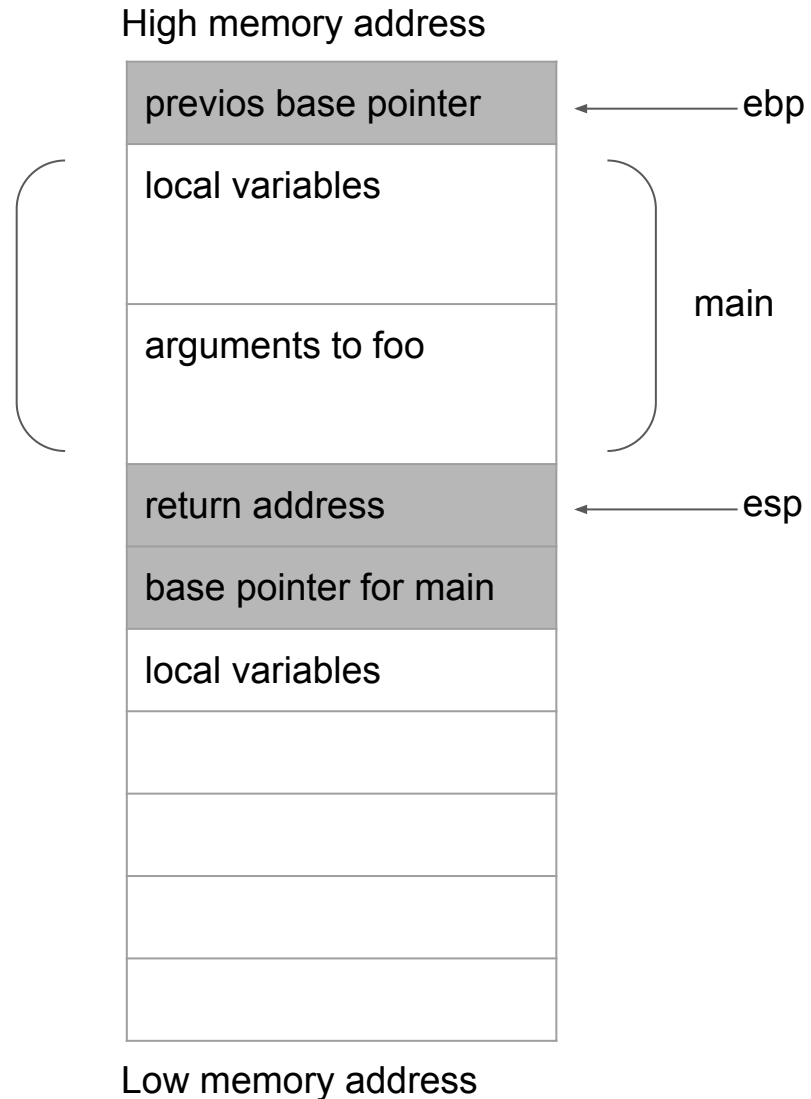
1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo
4. Do stuff in foo
5. Return to main

assembly: `leave`

is equivalent to

```
mov esp, ebp
```

```
pop ebp ←
```



Stack Frame

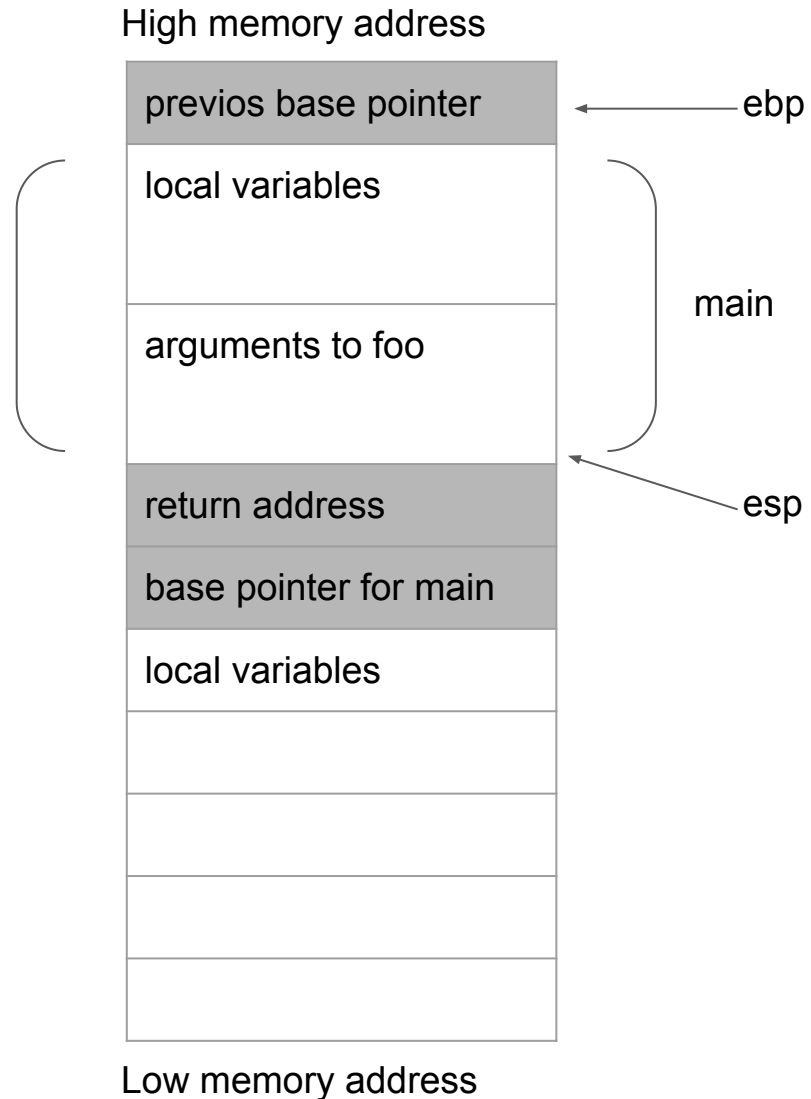
example: main calls foo

1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo
4. Do stuff in foo
5. Return to main

assembly: `ret`

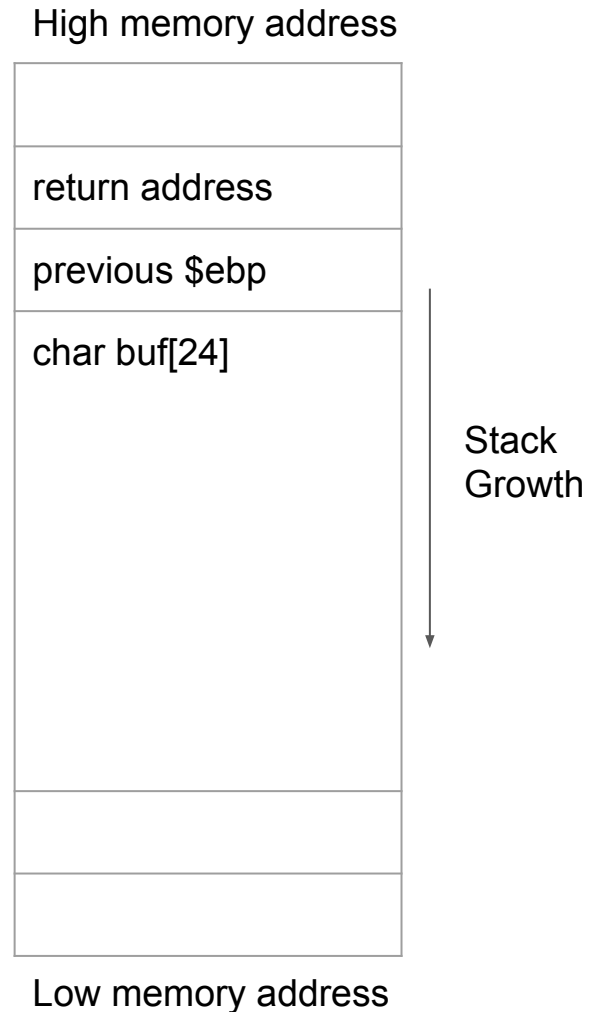
is equivalent to

`pop eip ←`



Buffer Overflow

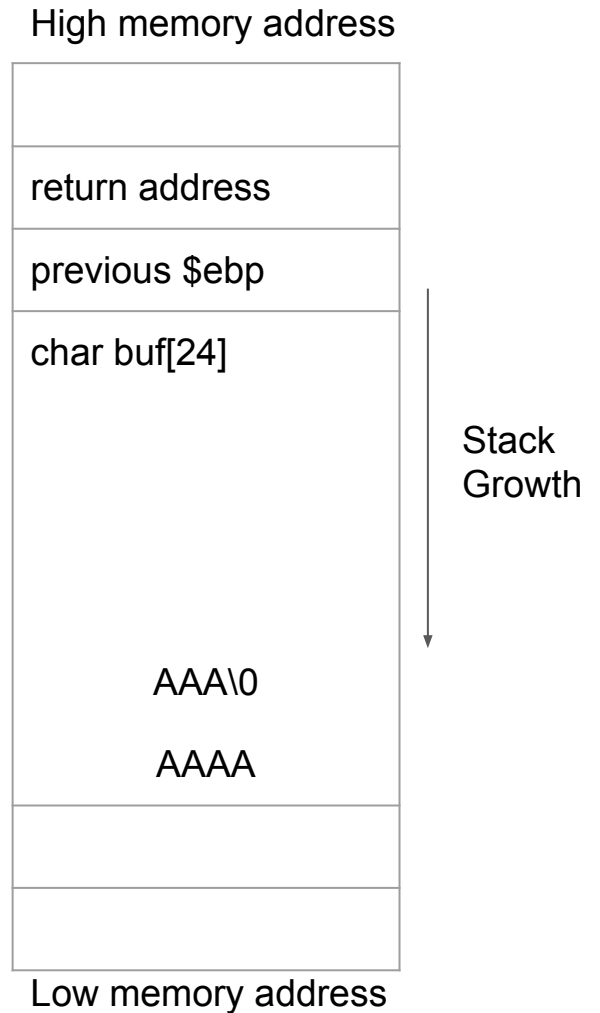
Overwrite the return address on the stack frame,
to hijack the program control flow



Buffer Overflow

gets(buf);

input: AAAAAAA

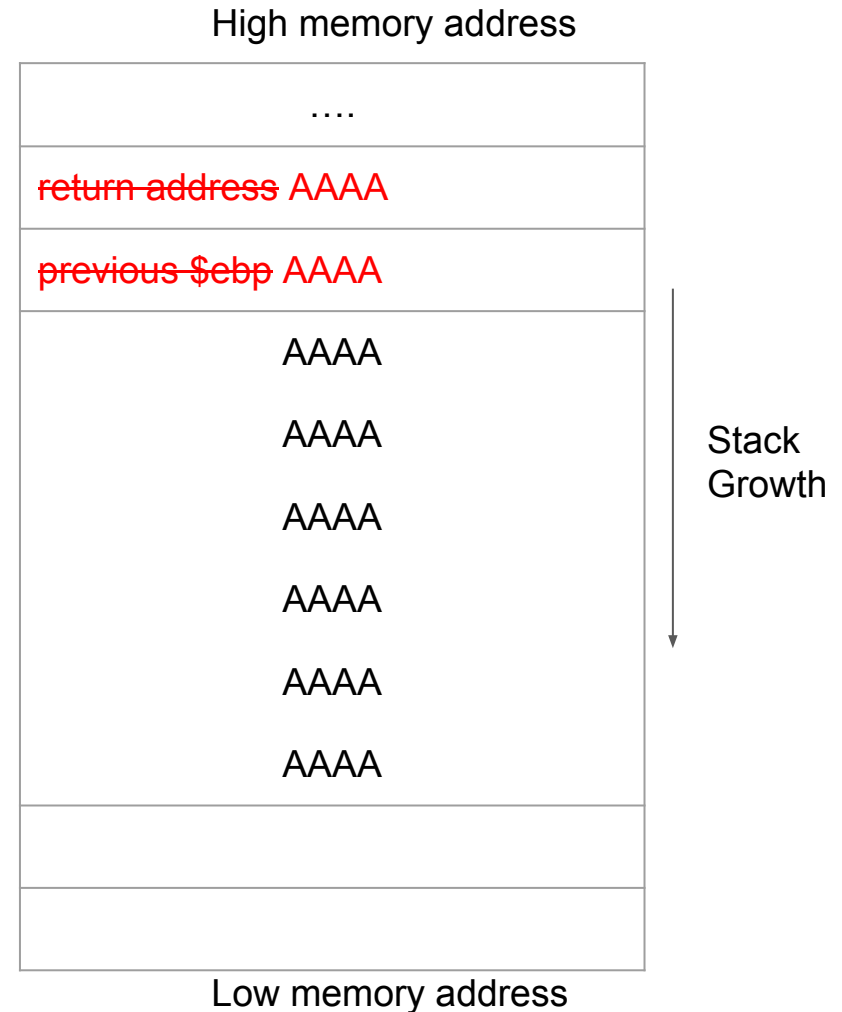


Buffer Overflow

```
gets(buf);
```

```
input: AAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAA....
```

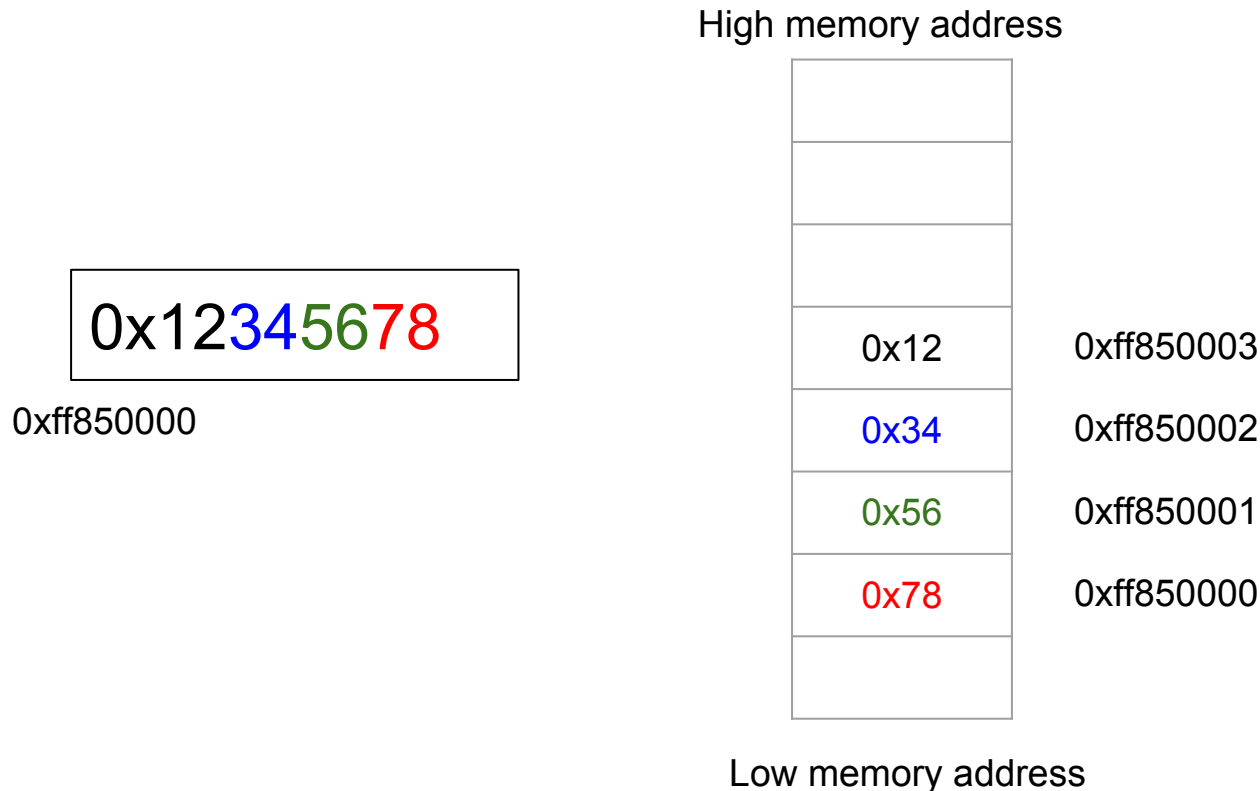
If you overwrite the original return address with the address of `magic()`, it will transfer control to `magic()` after this function return.



Endianness

Byte order for x86 is little endian

The least significant byte (LSB) value is at the lowest address. The other bytes follow in increasing order of significance.



Non-Printable Characters

Create non-printable string 0x08 0x04 0x88 0xe5

```
$ echo -ne [your malicious string]
```

```
$ echo -ne "\x08\x04\x88\xe5"  
?[1m%
```

Save them into a file

```
$ echo -ne [your malicious string] > [filename]
```

```
$ echo -ne "\x08\x04\x88\xe5" > payload.txt
```

Check the content in the file

```
$ xxd [filename]
```

```
$ xxd payload.txt  
00000000: 0804 88e5
```

```
....
```

Send payload to remote server

Connect to your target machine

```
$ nc -q -2 140.113.194.66 [your port number]
```

e.g.

```
$ nc -q -2 140.113.194.66 12345
```

Connect and send your payload in a file to the target machine

```
$ nc -q -2 140.113.194.66 [your port number] < [filename]
```

e.g.

```
$ nc -q -2 140.113.194.66 12345 < payload.txt
```

```
$ nc -q -2 140.113.194.66 12345 < payload.txt  
FLAG{this is a fake flag}
```

More Info

X86 Assembly

<http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html#s3>

https://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax

GDB

<http://csapp.cs.cmu.edu/2e/docs/gdbnotes-ia32.pdf>

<https://beej.us/guide/bggdb/>

You are free to use any other tools.

GET STARTED EARLY!