Project 3 - BufferOverflow

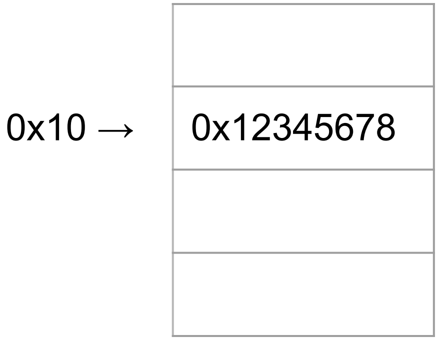
Network Security
By Po-Hsing Wu & Te-Yu Chang

Outline

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

Assembly Instructions (Intel Syntax)

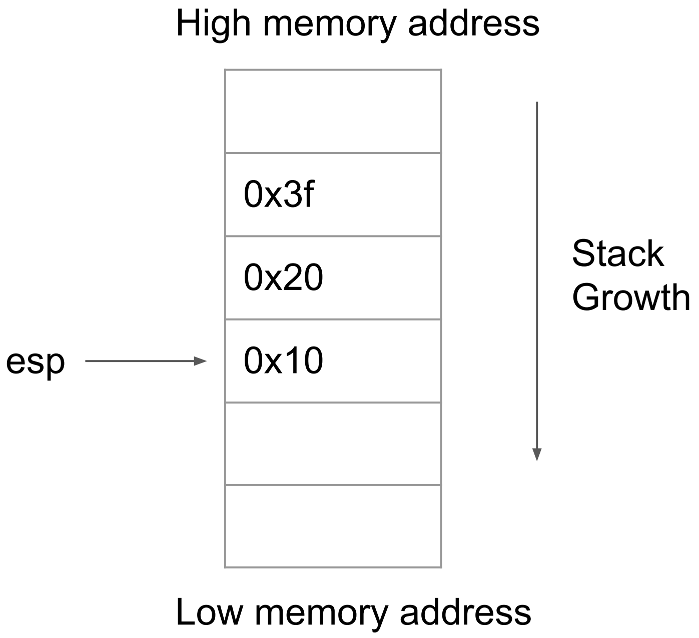
```
Opcode Dest    Source          Result
mov    eax,    0x10            # eax: 0x10
sub    eax,    0x01            # eax: 0x0f
add    eax,    0x0f            # eax: 0x1e

lea    ebx,    [eax-0xe]       # ebx: 0x10; eax: 0x1e;
mov    ebx,    [eax-0xe]       # ebx:  the value stored at 0x10 in Mem.
                                      i.e., 0x12345678
```



0x10 → 0x12345678

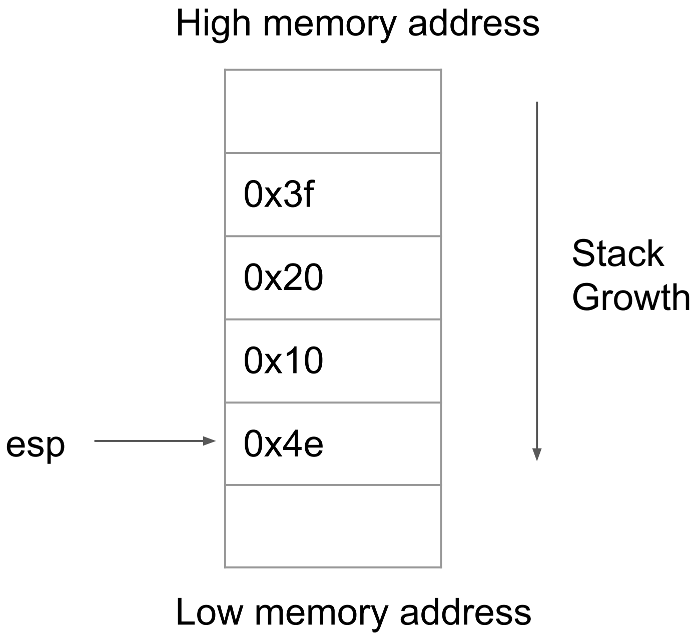
Stack

ESP(stack pointer) points to top of stack



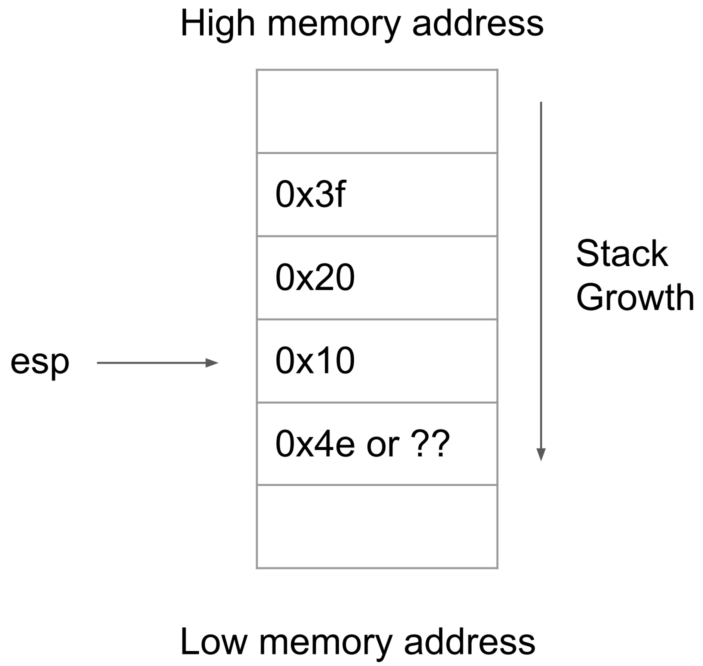
Stack

```
push   0x4e
```



Stack

```
push   0x4e

pop    eax    #now eax equals 0x4e
```



High memory address

0x3f

0x20

esp

0x10

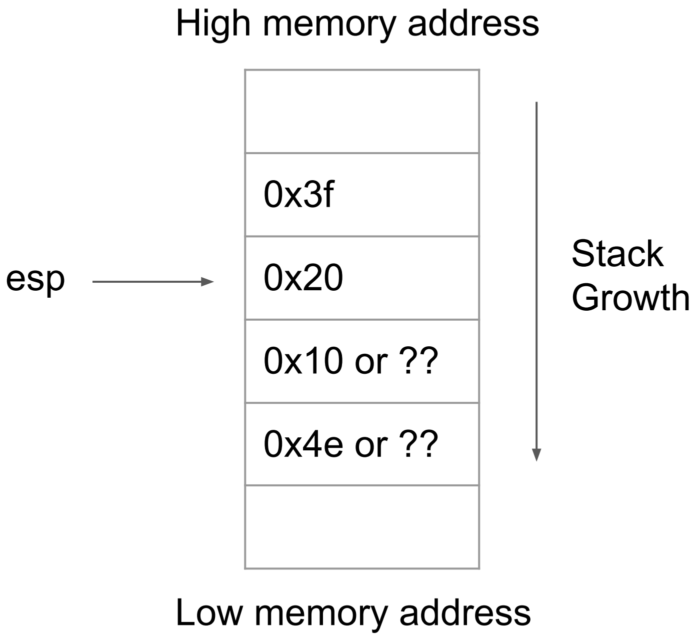
0x4e or ??

Stack Growth

Low memory address

Stack

```
push   0x4e

pop    eax    #now eax equals 0x4e

pop    ebx    #now ebx equals 0x10
```



High memory address

0x3f

esp

0x20

0x10 or ??

0x4e or ??

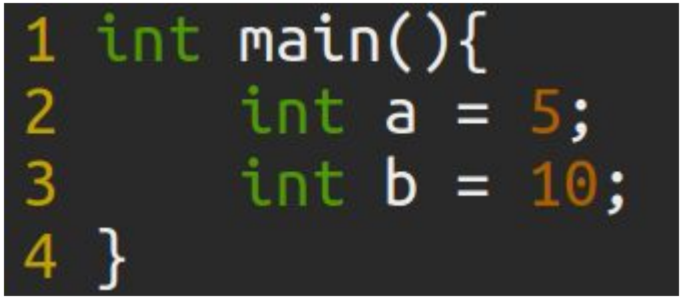
Stack Growth

Low memory address

Stack Frame

example: main calls foo

1. Do stuff in main

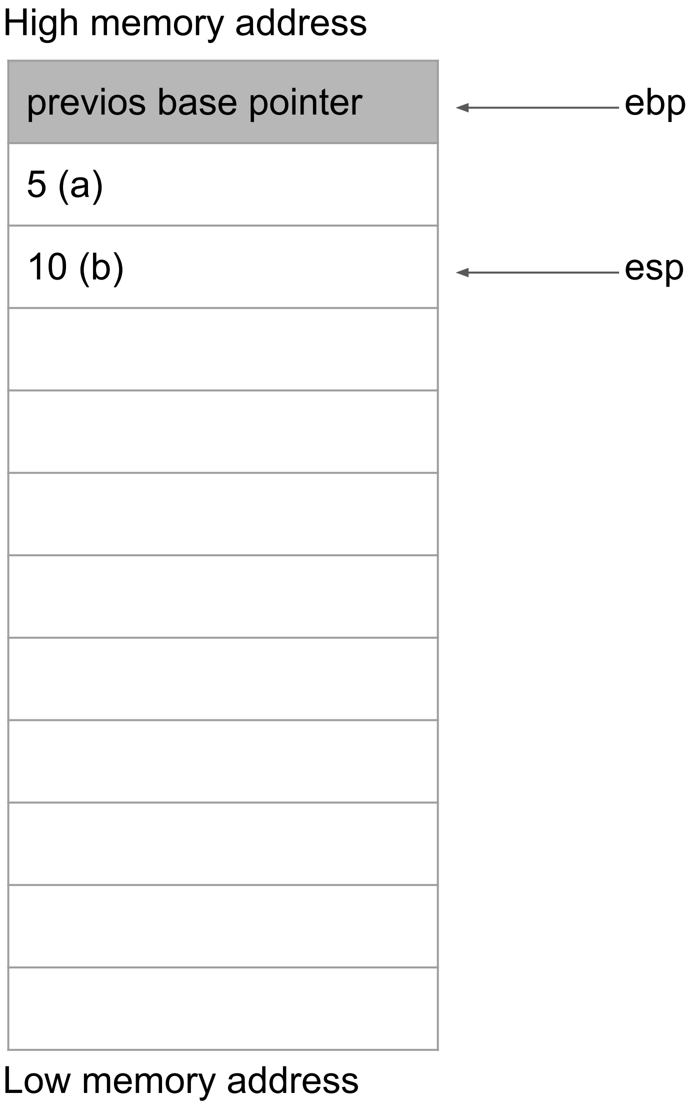


```
int main(){
        int a = 5;
        int b = 10;
}
```

ex:

int a =5; (push 5)

int b = 10; (push 10)



High memory address

previos base pointer ← ebp

5 (a)

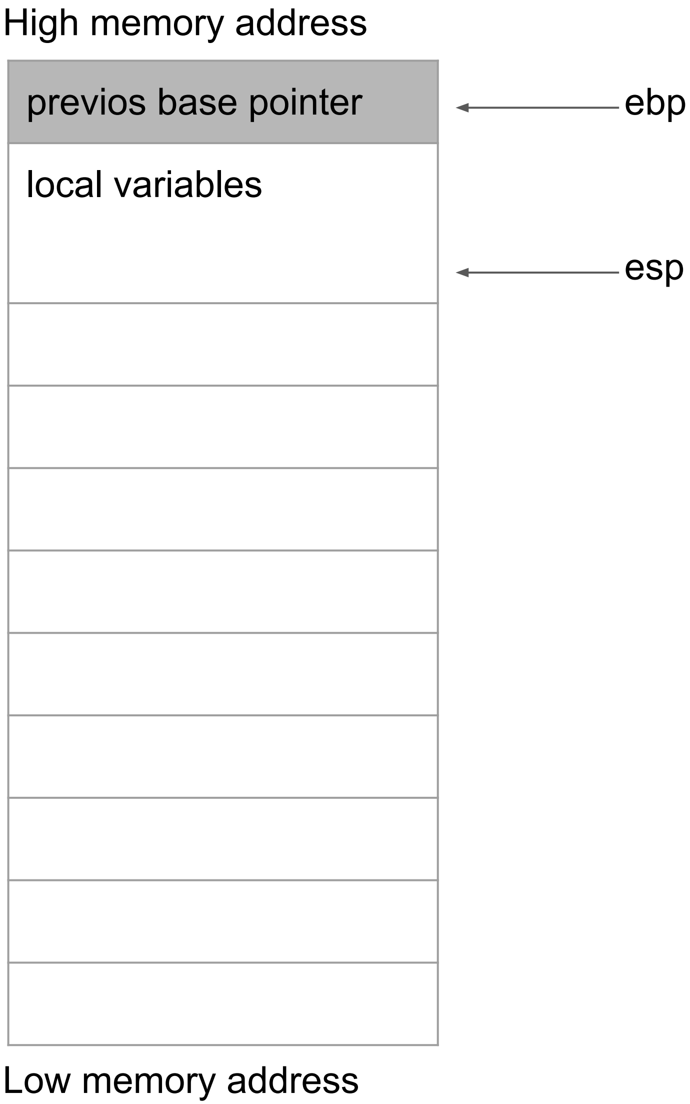
10 (b) ← esp

Low memory address

Stack Frame

example: main calls foo

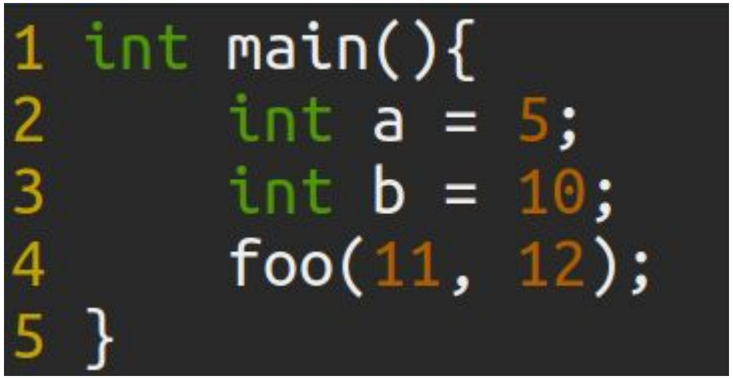
1. Do stuff in main



Stack Frame

example: main calls foo

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

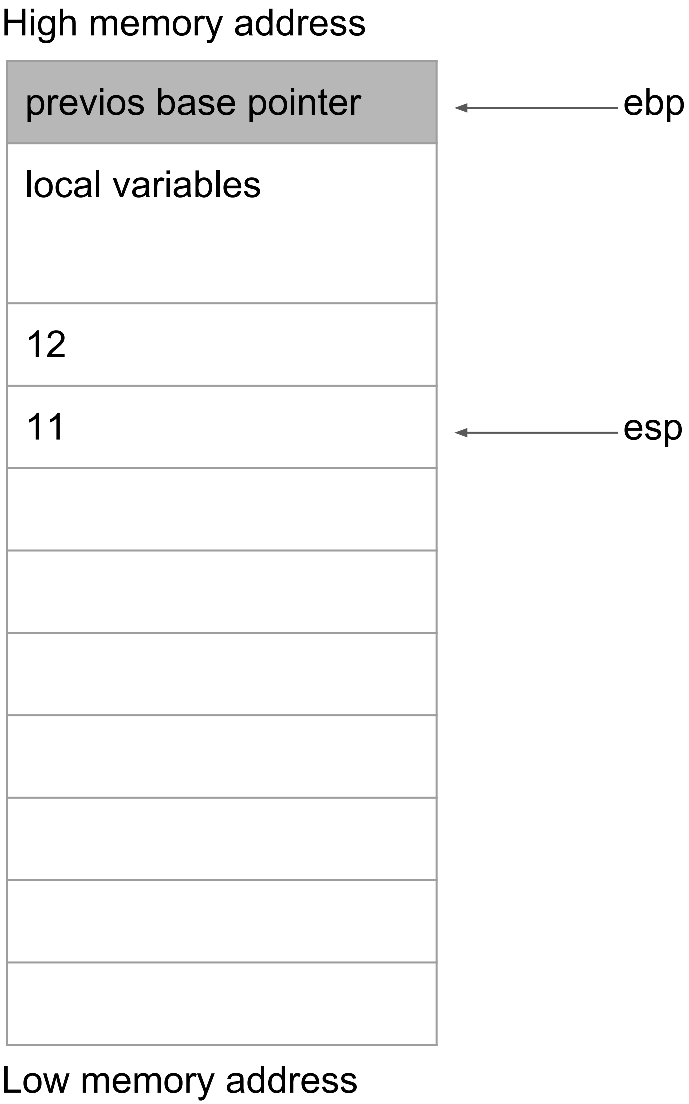


```
int main(){
        int a = 5;
        int b = 10;
        foo(11, 12);
}
```

ex:

foo takes two intergers

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



High memory address

previos base pointer

local variables

12

11

Low memory address

Stack Frame

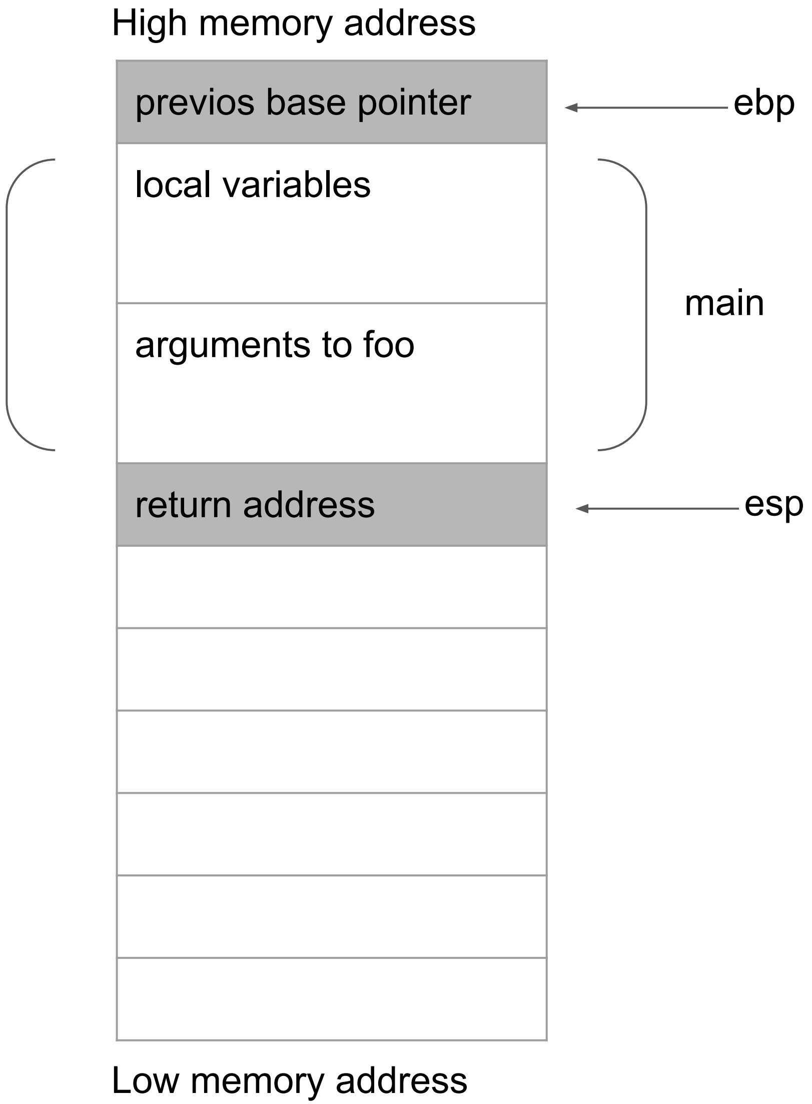
example: main calls foo

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()
```



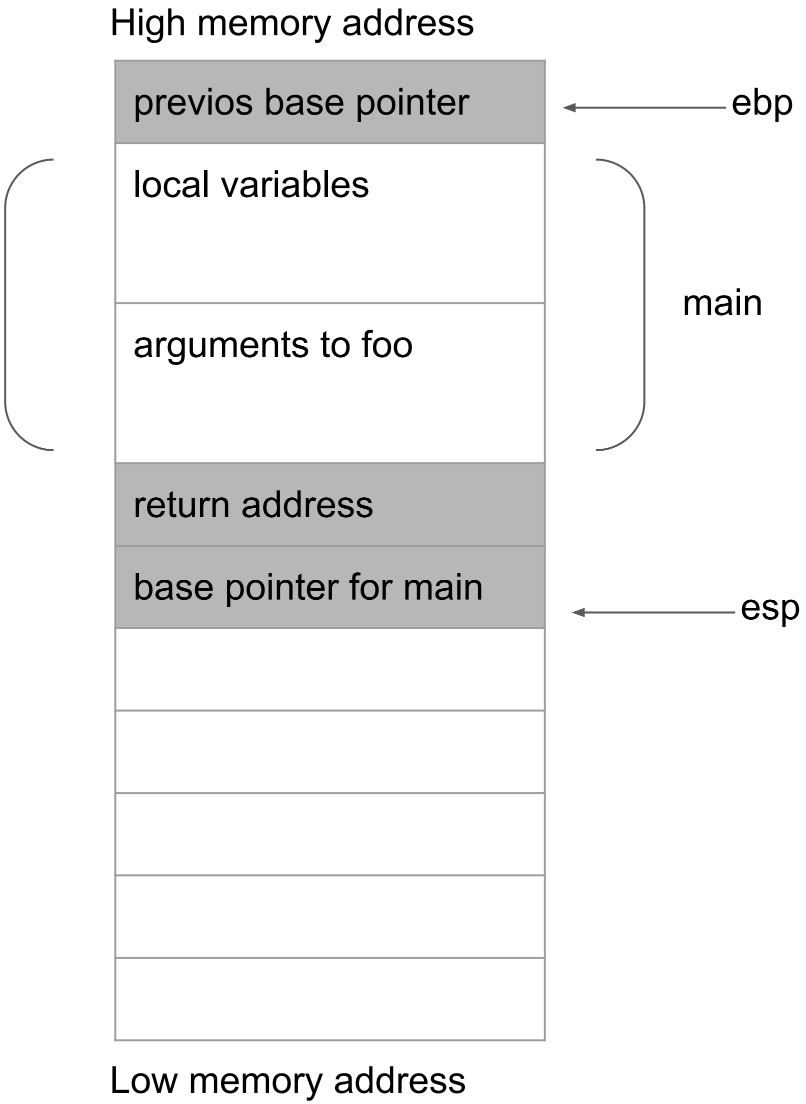
Stack Frame

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
```



High memory address

previos base pointer

ebp

local variables

main

arguments to foo

return address

base pointer for main

esp

Low memory address

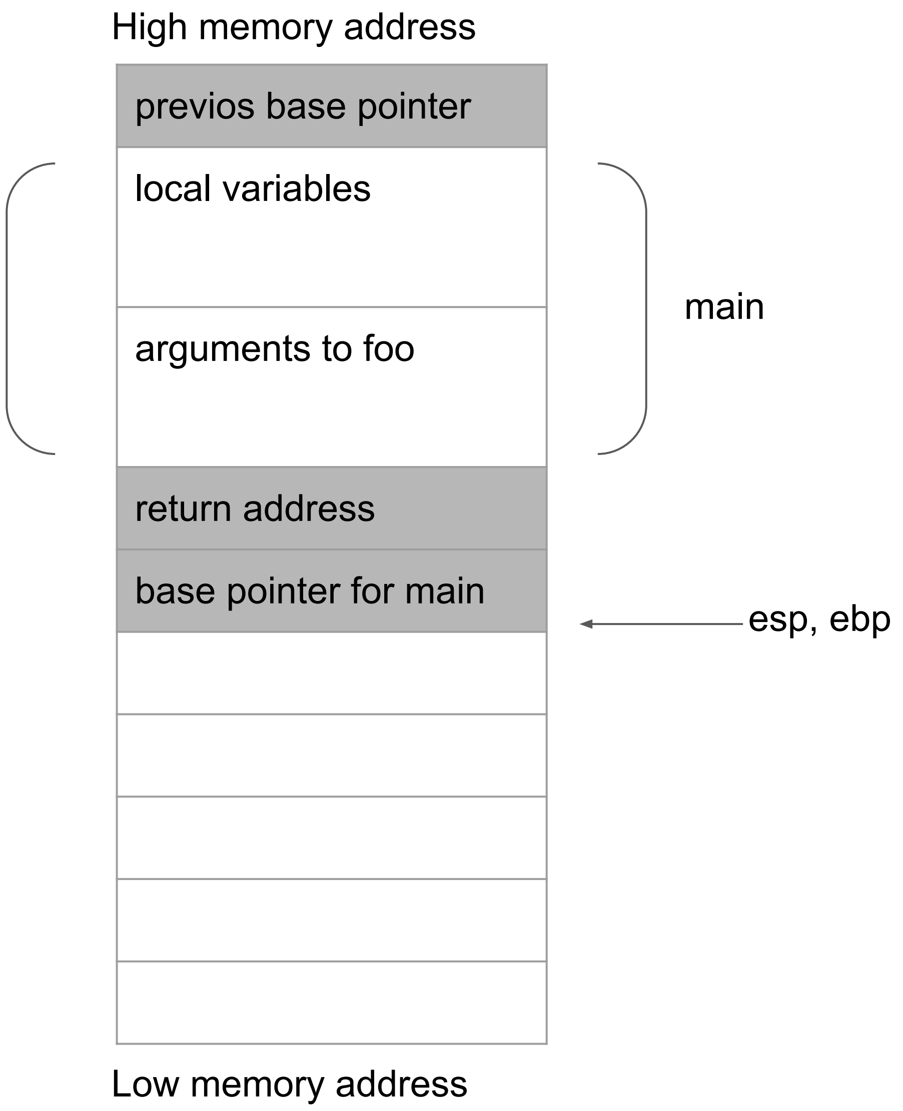
Stack Frame

example: main calls foo

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

assembly:

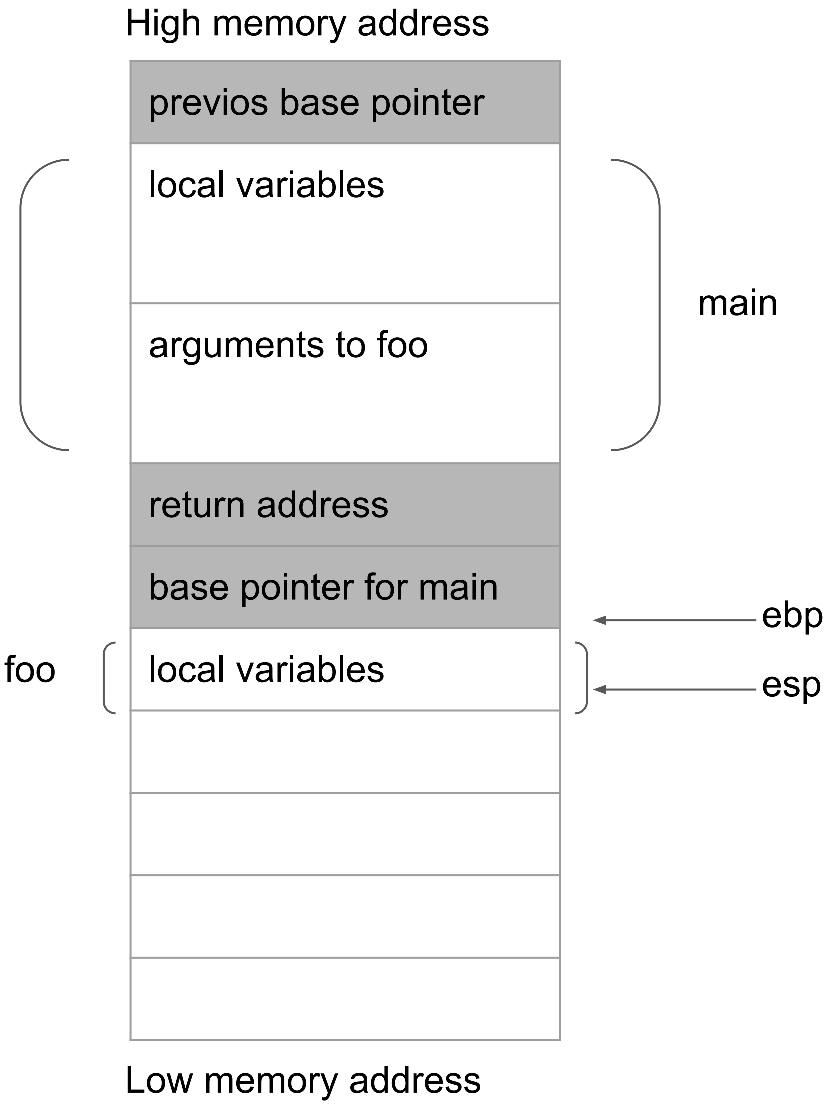
```
call foo
…
foo:
push   ebp
mov    ebp, esp
```



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

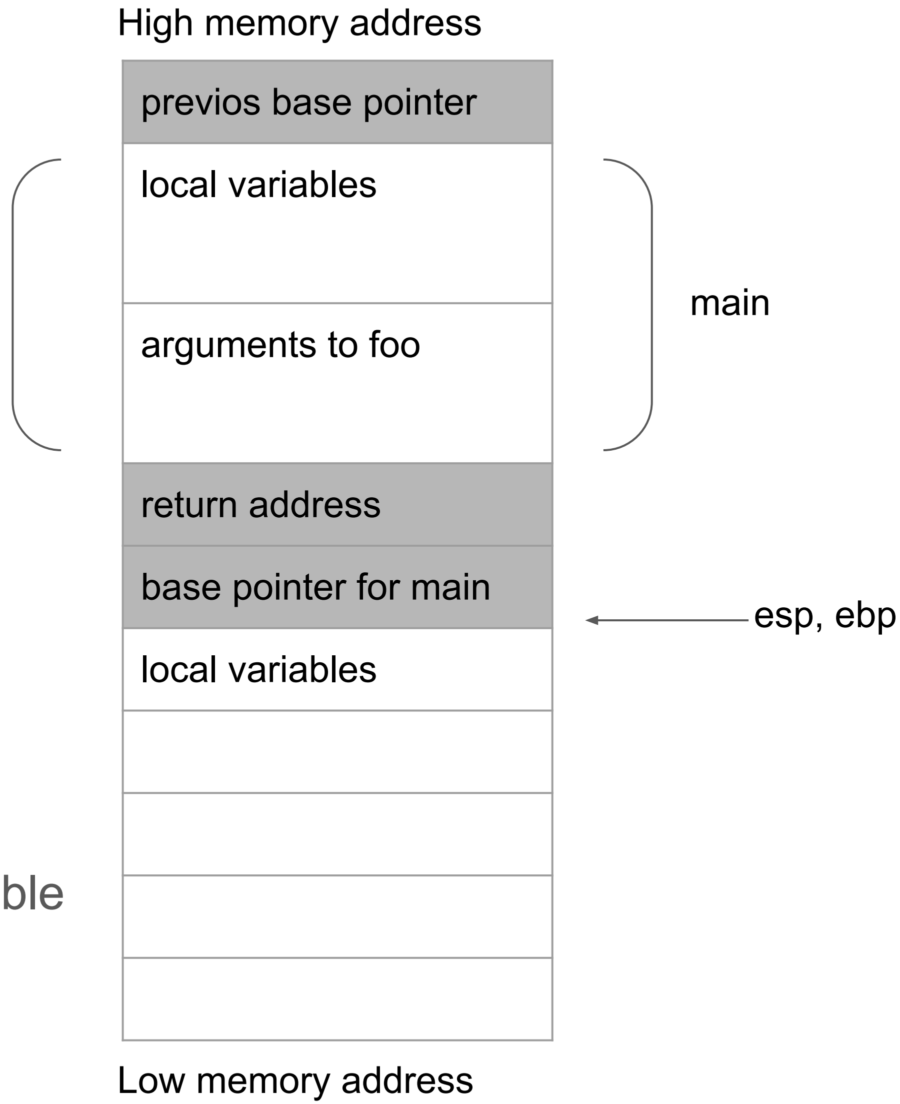
1. Do stuff in main
2. Set up arguments to call foo
3. Set up stack frame for foo
4. Do stuff in foo
5. Return to main

assembly: `leave`

is equivalent to

```
mov esp, ebp
pop ebp
```

←delete all local variable



Stack Frame

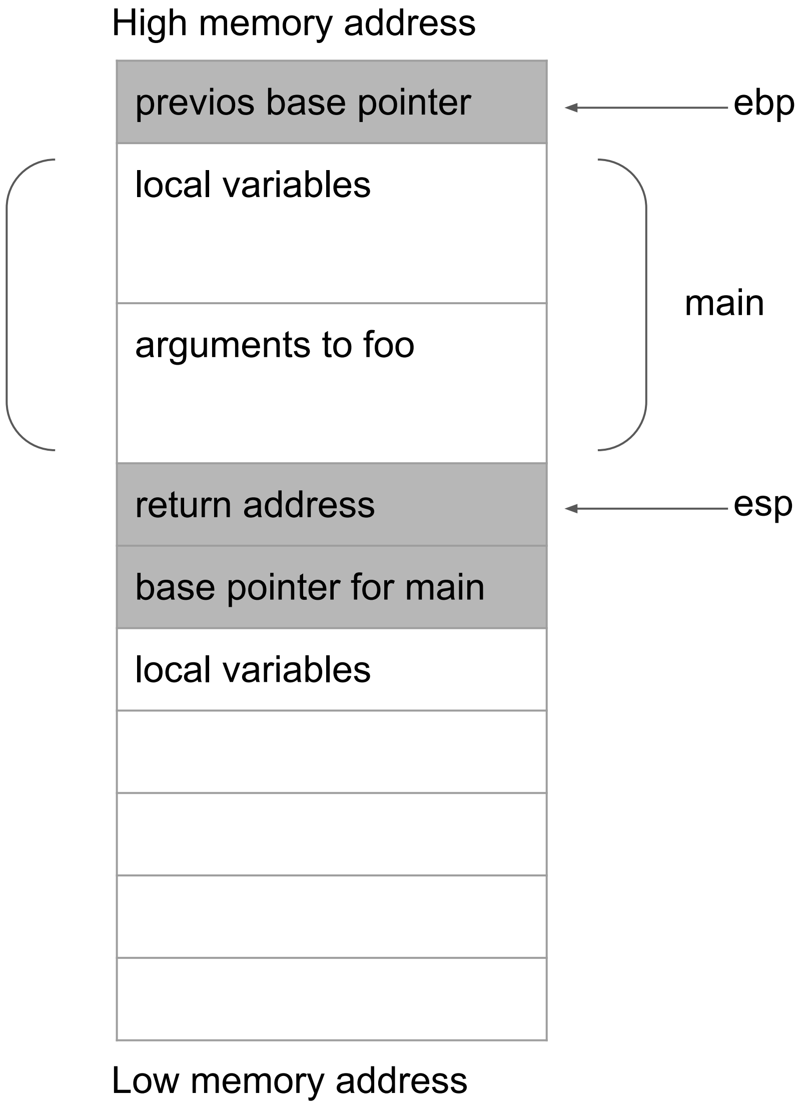
example: main calls foo

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

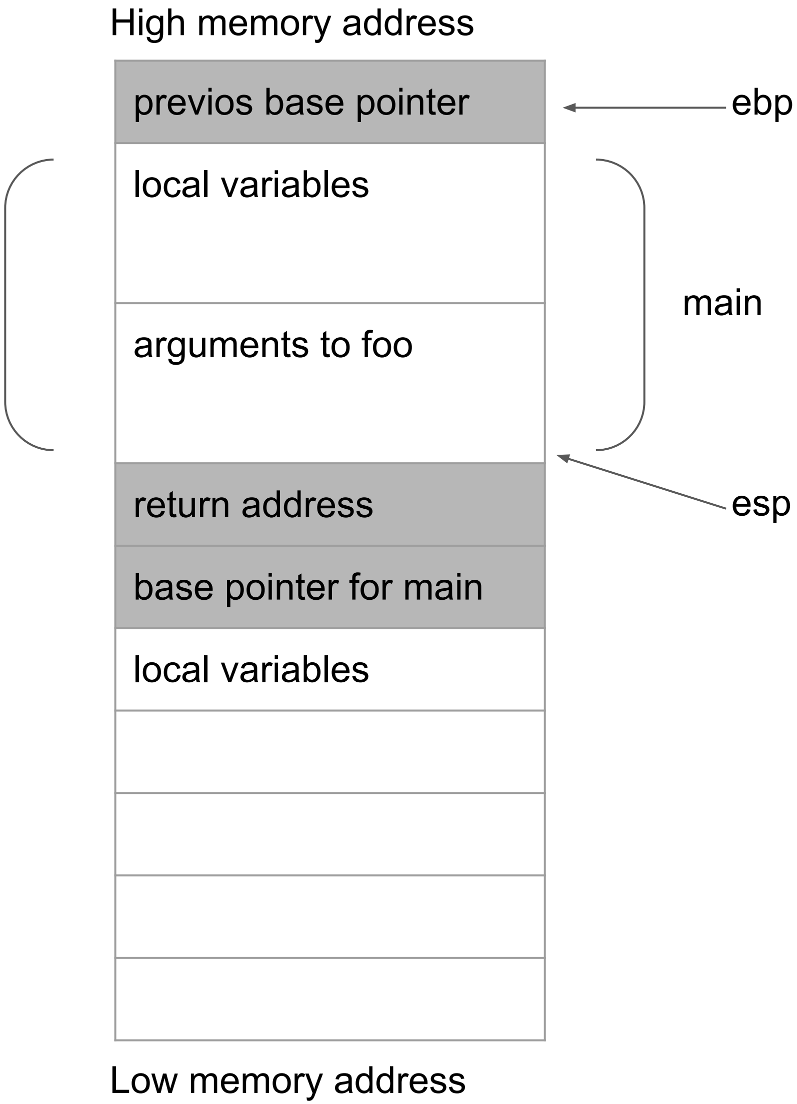
example: main calls foo

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`

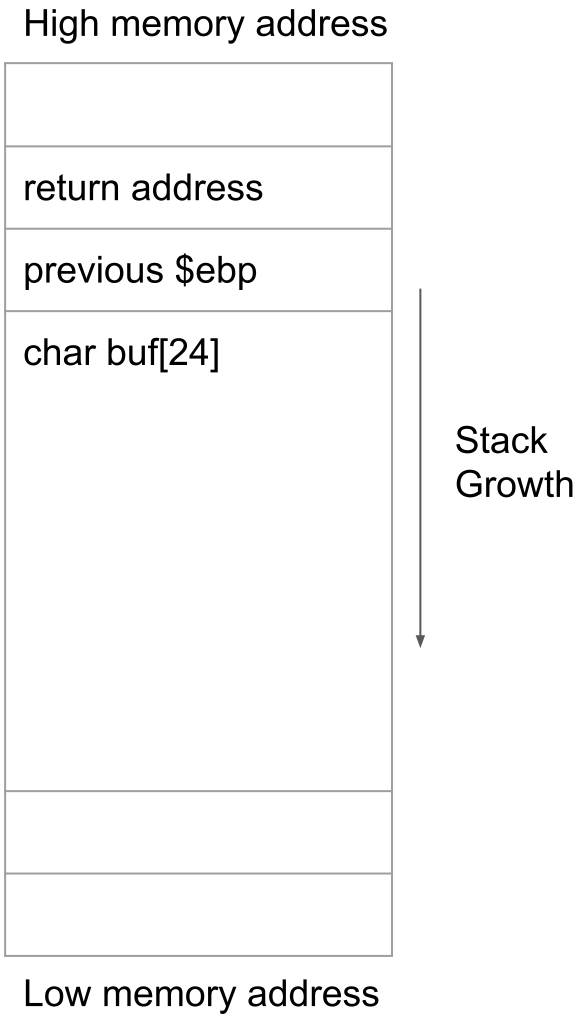
is equivalent to

`pop eip` ←



Buffer Overflow

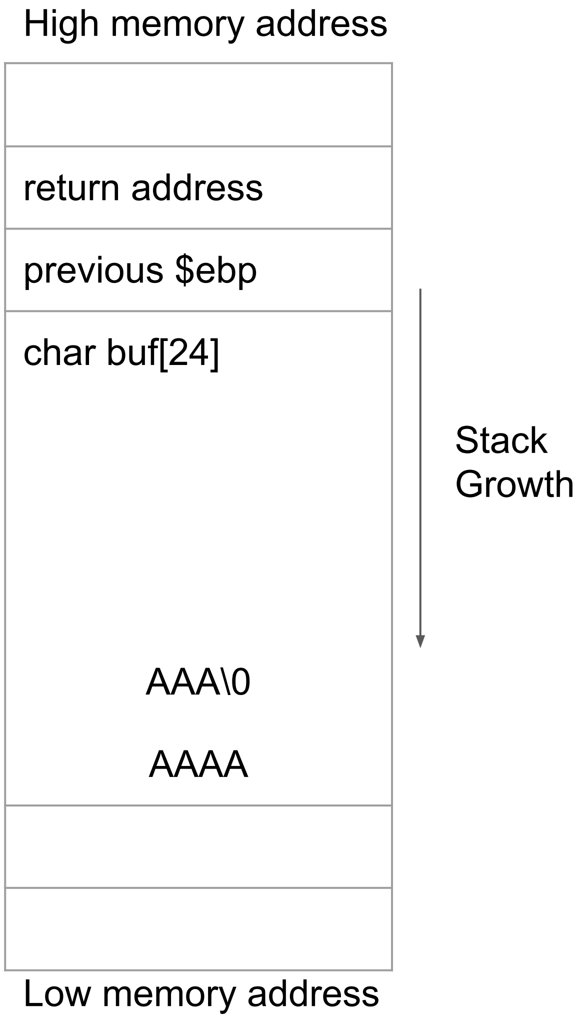
Overwrite the return address on the stack frame, to hijack the program control flow



Buffer Overflow

gets(buf);

input: AAAAAAA

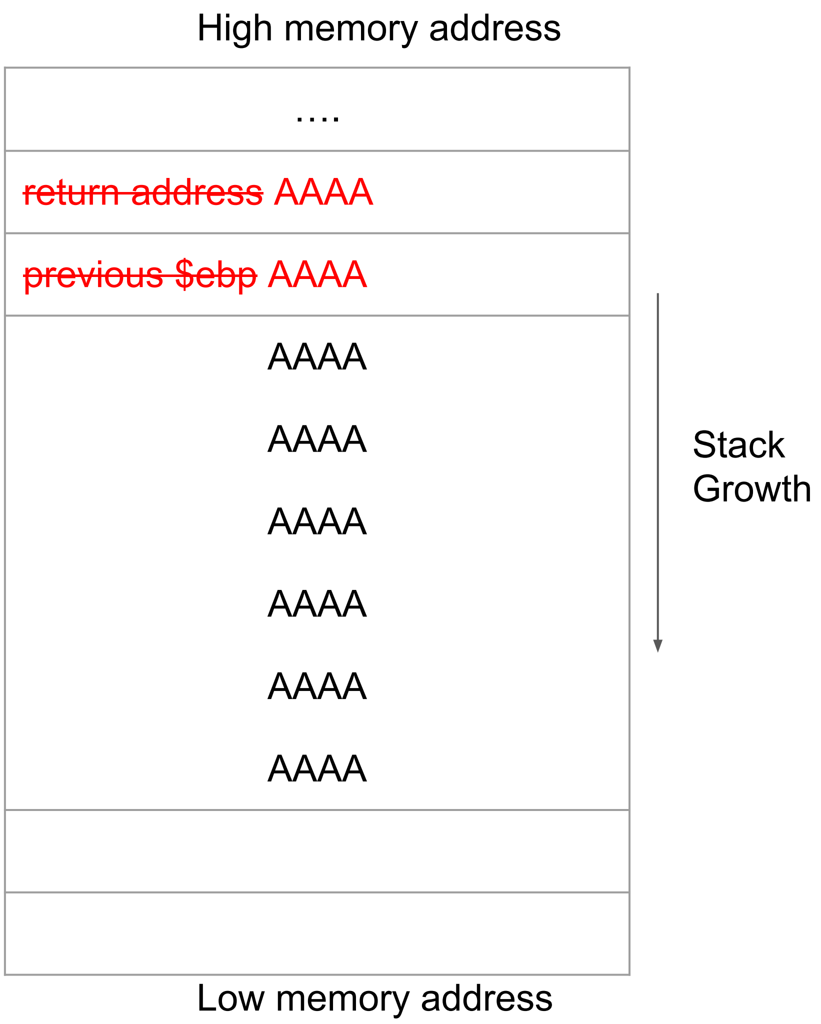


Buffer Overflow

gets(buf);

input: AAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA….

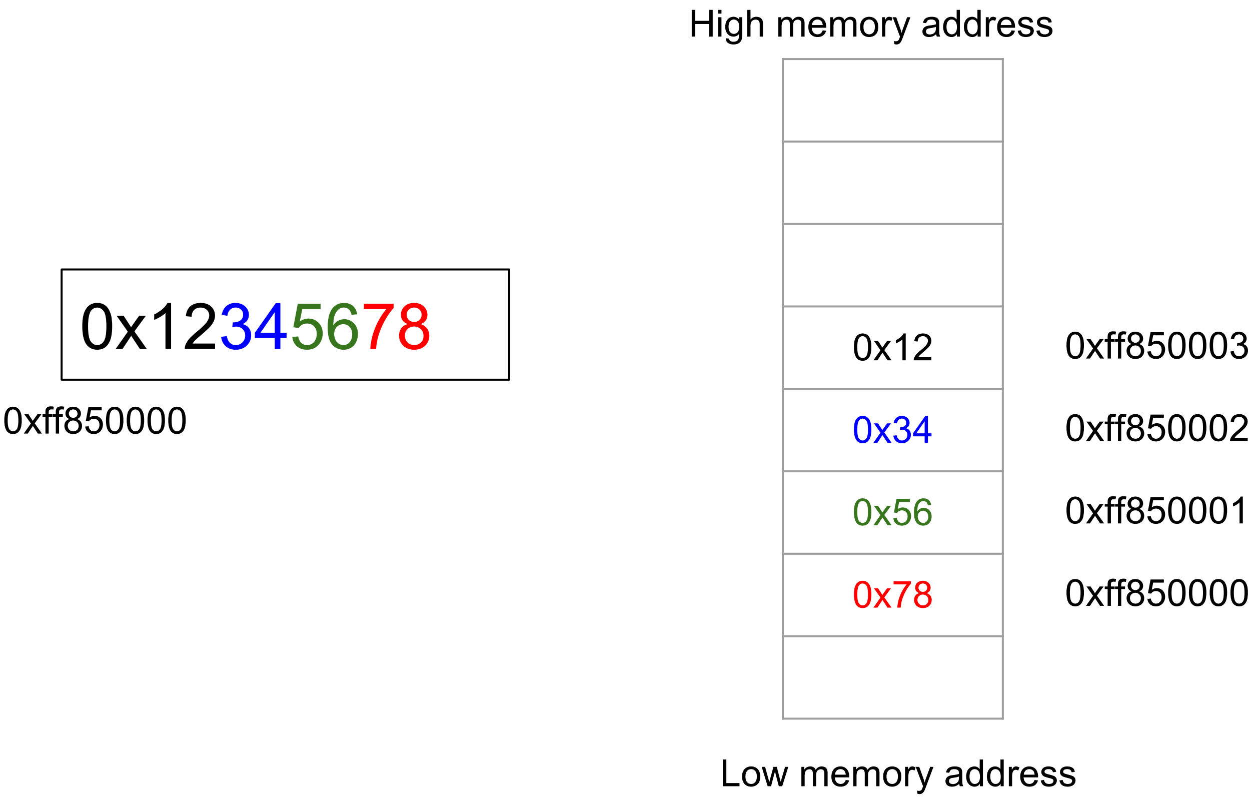
If you overwrite the original return address with the address of `magic()`, it will transfer control to `magic()` after this function return.



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.



0x12345678

0xff850000

High memory address

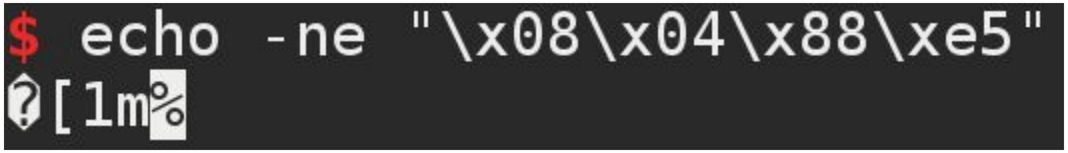
0x12	0xff850003
0x34	0xff850002
0x56	0xff850001
0x78	0xff850000

Low memory address

Non-Printable Characters

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

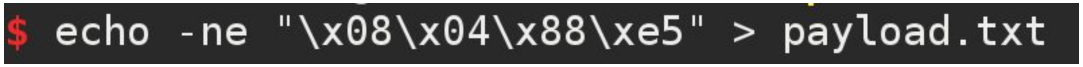
```
$ echo -ne [your malicious string]
```



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

Save them into a file

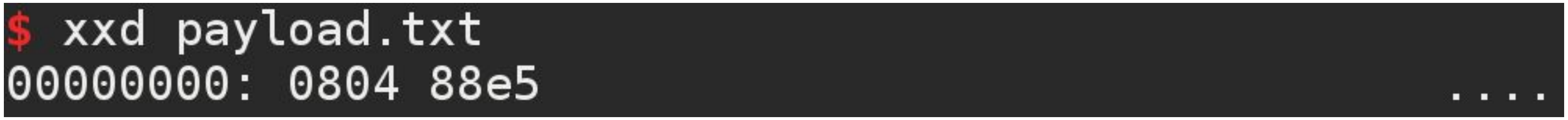
```
$ echo -ne [your malicious string] > [filename]
```



```
$ 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 -2 140.113.194.78 [your port nuber]
```

e.g.

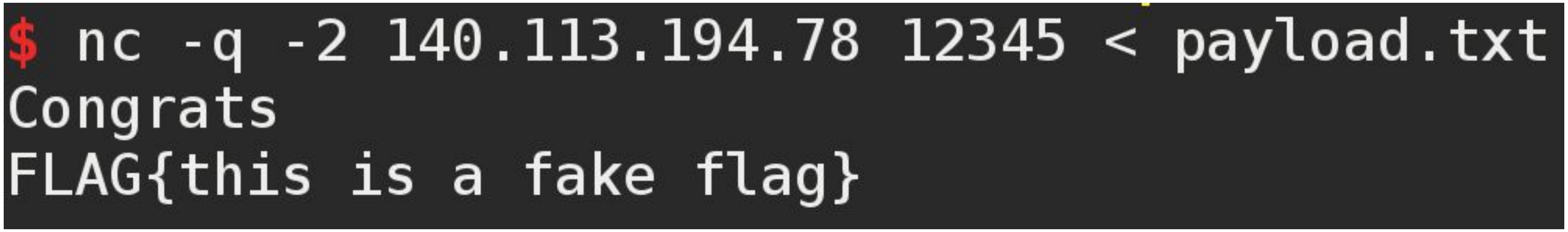
```
$ nc -q -2 140.113.194.78 12345
```

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

```
$ nc -q -2 140.113.194.78 [your port nuber] < [filename]
```

e.g.

```
$ nc -q -2 140.113.194.78 12345 < payload.txt
```



```
$ nc -q -2 140.113.194.78 12345 < payload.txt
Congrats
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!

32-bit x86 ISA

char	1 byte
int	4 bytes
Memory address	4 bytes
Each Memory location	1 bytes

0xbffe1236	0x10
0xbffe1235	0x20
0xbffe1234	0x3f