# Buffer Overflows

# Memory Organization Topics

**Kernel organizes memory in pages**

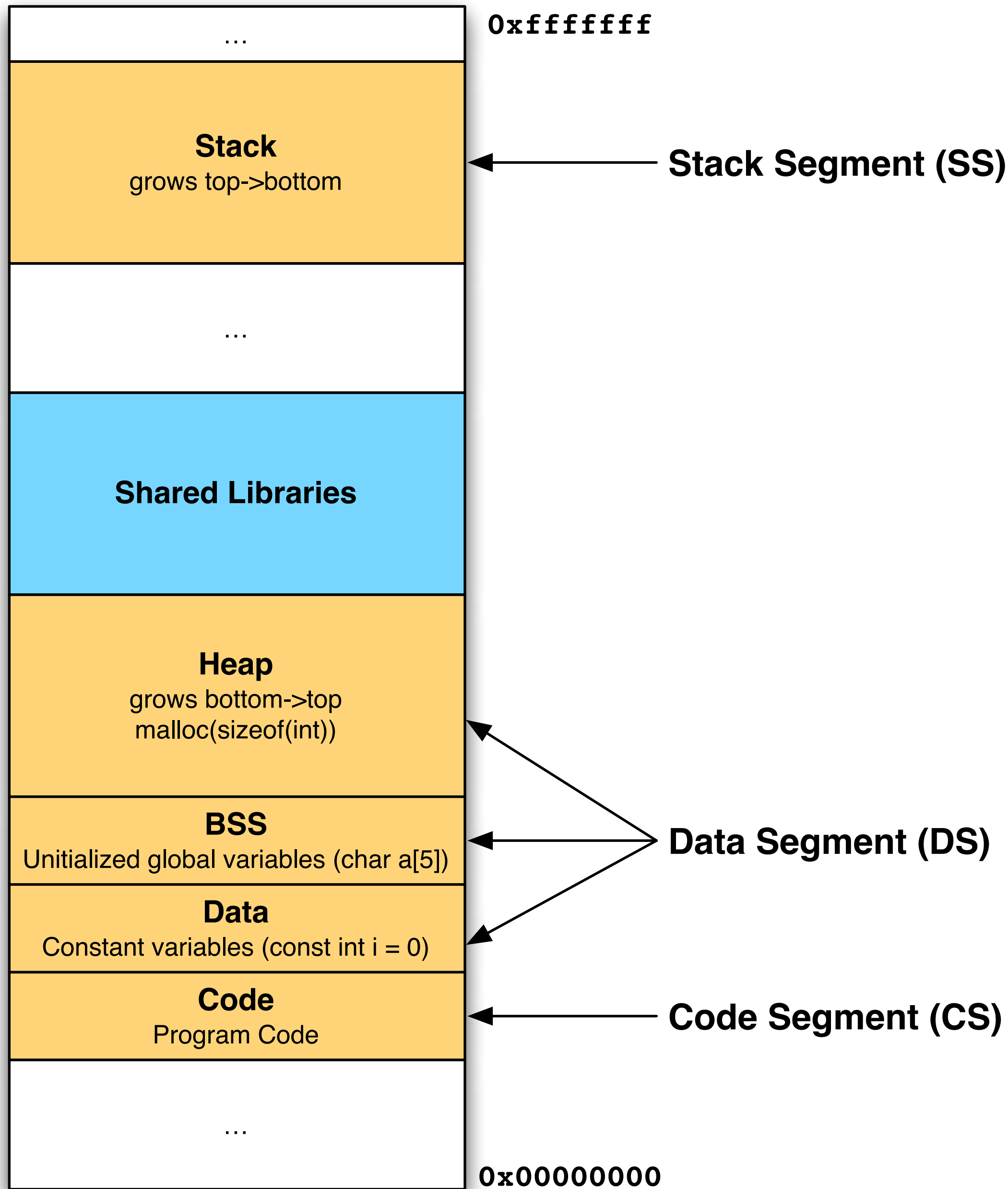
- Typically 4k bytes

**Processes operate in a Virtual Memory Space**

- Mapped to real 4k pages
  - Could live in RAM or be swapped

**Kernel splits program in several segments**

- Increases security
  - segment based permissions
- Increases performance
  - some are dynamic: invalidated when program terminates
  - some are static: can be retained, speed repeated startup


...
0xffffffff
Stack
grows top->bottom
Stack Segment (SS)
...
Shared Libraries
Heap
grows bottom->top
malloc(sizeof(int))
BSS
Unitialized global variables (char a[5])
Data Segment (DS)
Data
Constant variables (const int i = 0)
Code
Program Code
Code Segment (CS)
...
0x00000000

# mem.c

```
Internal Variables (Page = 4096)
&argc  = bfeb8590 -> stack = bfeb8000
malloc = 08435008 -> heap  = 08435000
bssvar = 0804a034 -> bss   = 0804a000
cntvar = 08048920 -> const = 08048000
&main  = 0804865c -> text  = 08048000
```

# mem.c

```
Content of /proc/self/maps
08048000-08049000  r-xp 00000000 08:01 26845750   /home/s/seguranca/mem
08049000-0804a000  r--p 00000000 08:01 26845750   /home/s/seguranca/mem
0804a000-0804b000  rw-p 00001000 08:01 26845750   /home/s/mem
08435000-08456000  rw-p 00000000 00:00 0          [heap]
b7616000-b7617000  rw-p 00000000 00:00 0
b7617000-b776a000  r-xp 00000000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776a000-b776b000  ---p 00153000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776b000-b776d000  r--p 00153000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776d000-b776e000  rw-p 00155000 08:01 1574823    /lib/tls/i686/cmov/libc-2.11.1.so
b776e000-b7771000  rw-p 00000000 00:00 0
b777e000-b7782000  rw-p 00000000 00:00 0
b7782000-b7783000  r-xp 00000000 00:00 0          [vdso]
b7783000-b779e000  r-xp 00000000 08:01 1565567    /lib/ld-2.11.1.so
b779e000-b779f000  r--p 0001a000 08:01 1565567    /lib/ld-2.11.1.so
b779f000-b77a0000  rw-p 0001b000 08:01 1565567    /lib/ld-2.11.1.so
bfe99000-bfeba000  rw-p 00000000 00:00 0          [stack]
```

# mem.c

```
Stack evolution:
foo [000]: &argc  = bfeb8140 -> stack = bfeb8000
foo [001]: &argc  = bfdb8110 -> stack = bfdb8000
foo [002]: &argc  = bfcb80e0 -> stack = bfcb8000
foo [003]: &argc  = bfbb80b0 -> stack = bfbb8000
foo [004]: &argc  = bfab8080 -> stack = bfab8000
foo [005]: &argc  = bf9b8050 -> stack = bf9b8000
foo [006]: &argc  = bf8b8020 -> stack = bf8b8000
foo [007]: &argc  = bf7b7ff0 -> stack = bf7b7000
foo [008]: &argc  = bf6b7fc0 -> stack = bf6b7000
Segmentation fault
```

# CPU Registers (x86)

**General Purpose: EAX, EBX, ECX, EDX**
- A: 8bits, AX: 16bits, EAX: 32bits, RAX: 64bits

**EBP: Base Pointer**
- Points to Start of Stack

**ESP: Stack Pointer**
- Points to End of Stack

**EIP: Instruction Pointer**
- Points to current instruction

**ESI: Stack Index**
- Points to an address in Stack Segement

**EDI: Data Index**
- Points to an address in Data Segment

# Stack Segment

**Stack is used to pass parameters to functions**

- Ex: foo(int a)

**Stack is used to store local variables**

- Ex: int a;

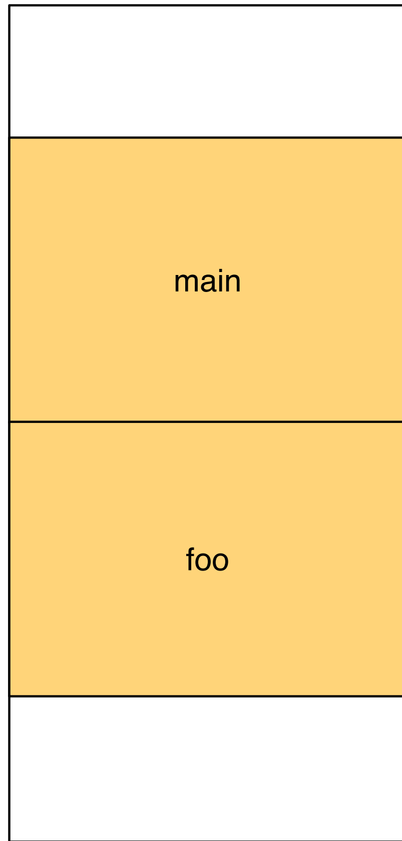
**Values are PUSHed or POPed from stack**

- Ex: push ebp, pop ebp

**Ex: Accessing a variable: ebp+4**

**allocating 4 bytes in stack: sub esp,4**

# stack.c



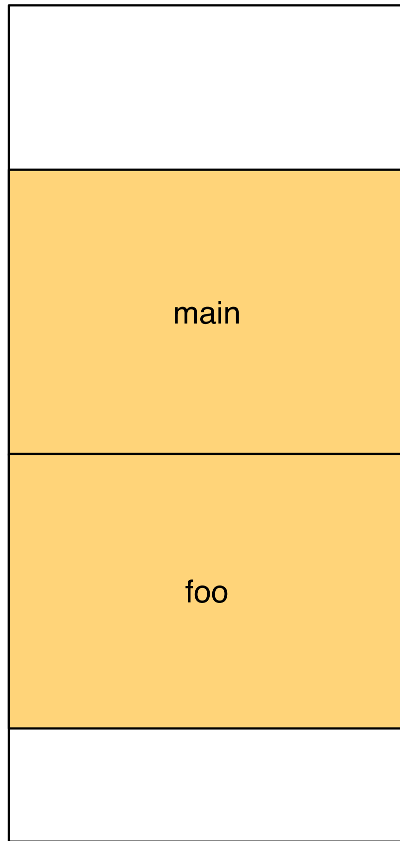

0xfffffff

```
int foo(int bar)
{
    return 3;
}

int main(int argc, char** argv)
{
    foo(argc);

    return 0;
}
```

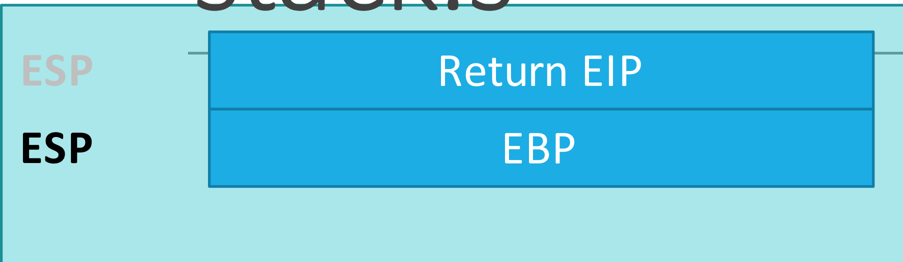
# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

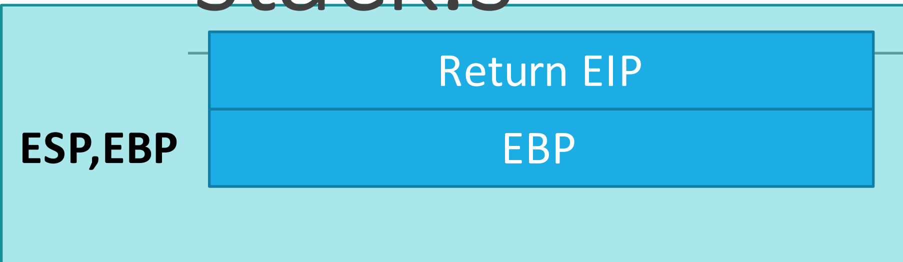
gcc -S –masm=intel –fno-stack-protector stack.c

# stack.s



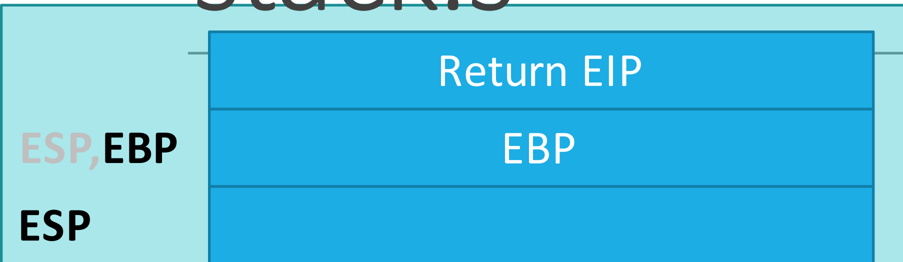

```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

# stack.s



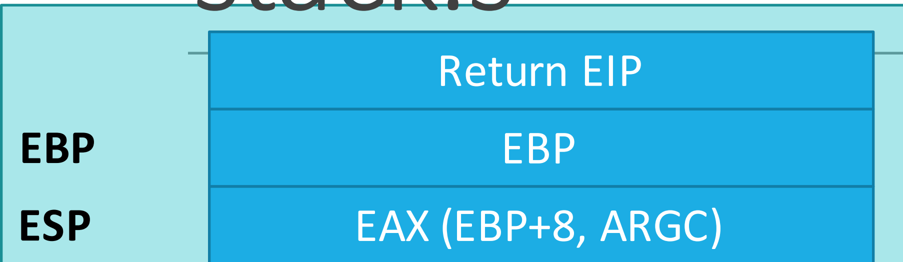

```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

# stack.s



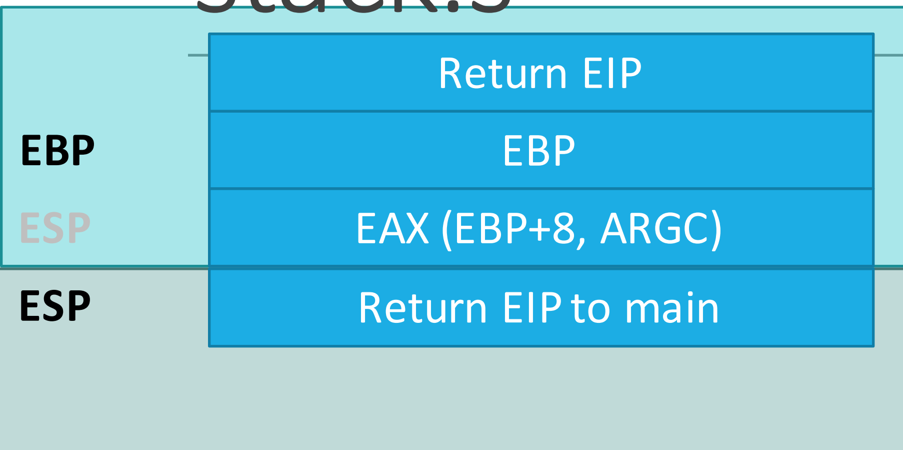

```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
→ mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

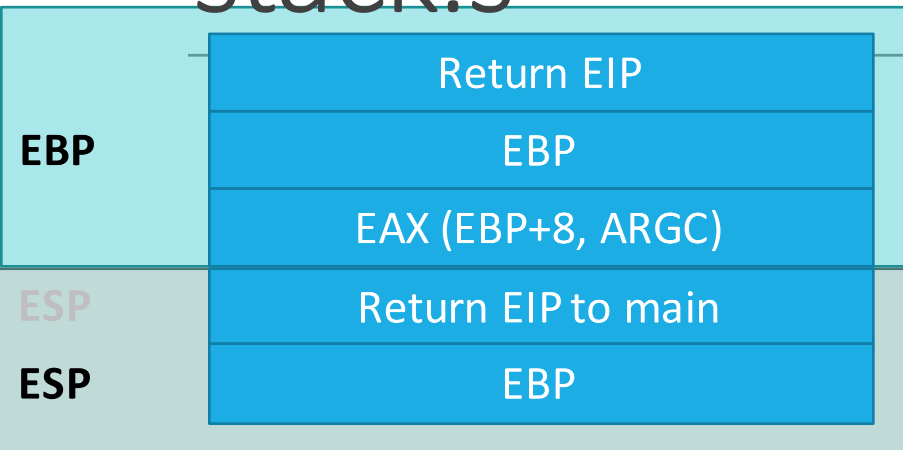
# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

Return EIP to main

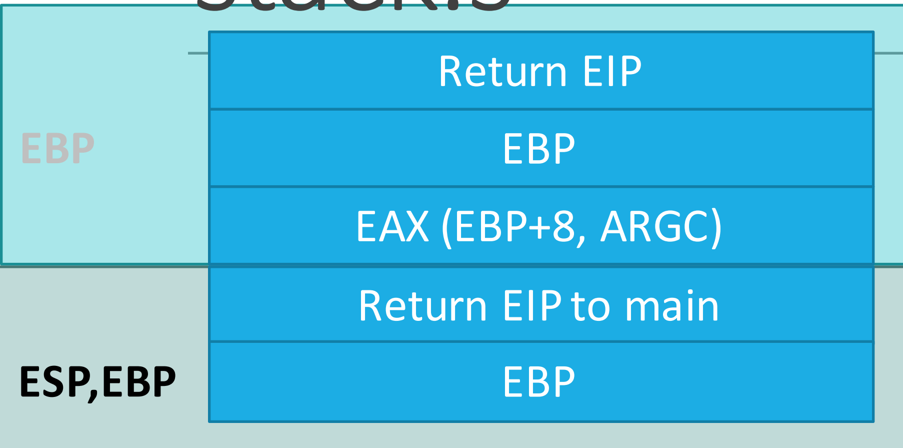
# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

Return EIP to main

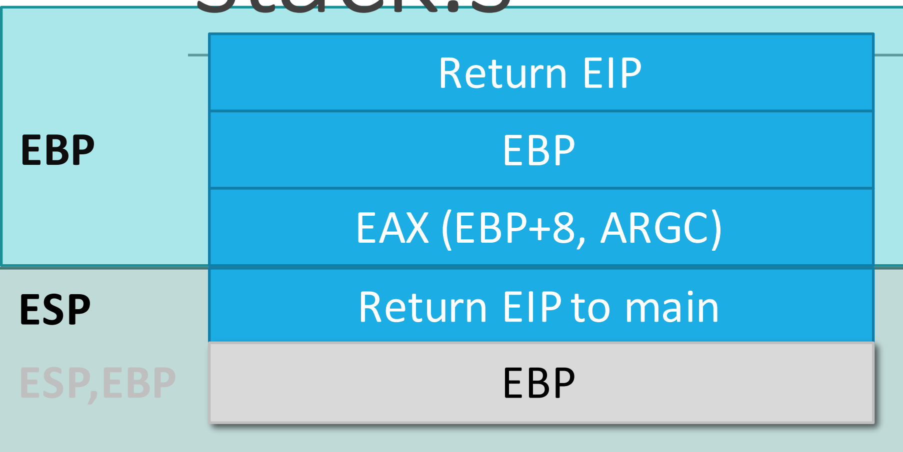
# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

Return EIP to main

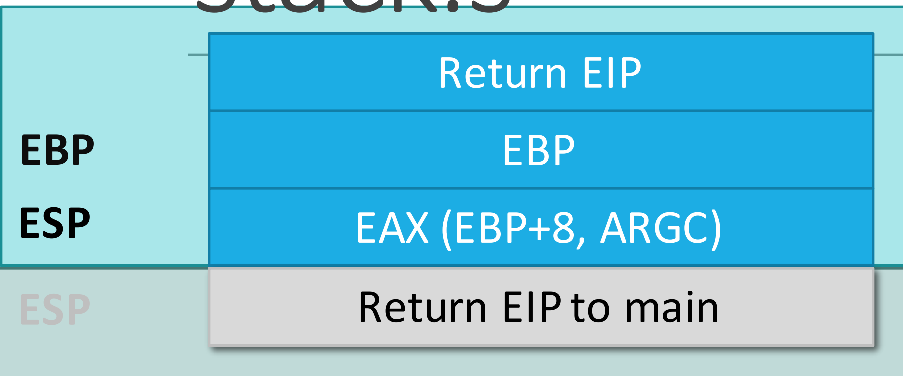
# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

Return EIP to main

# stack.s




```
foo:
  push  ebp
  mov ebp, esp
  mov eax, 3
  pop ebp
  ret
main:
  push  ebp
  mov ebp, esp
  sub esp, 4
  mov eax, DWORD PTR [ebp+8]
  mov DWORD PTR [esp], eax
  call  foo
  mov eax, 0
  leave
  ret
```

Return EIP to main

# Buffer Overflow

**Write over the boundaries of a buffer**

**Consequences**

- Write over other variables in local function
- Write over Return EIP
  - Jump to any address on return!
- Put code in stack and jump to stack
  - Execute injected code

# bo.c

```
.LC0:
  .string "%s"
  .text

foo:
push  ebp
mov ebp, esp
sub esp, 40
mov eax, OFFSET FLAT:.LC0
lea edx, [ebp-12]
mov DWORD PTR [esp+4], edx
mov DWORD PTR [esp], eax
call  __isoc99_scanf
Leave
ret
```

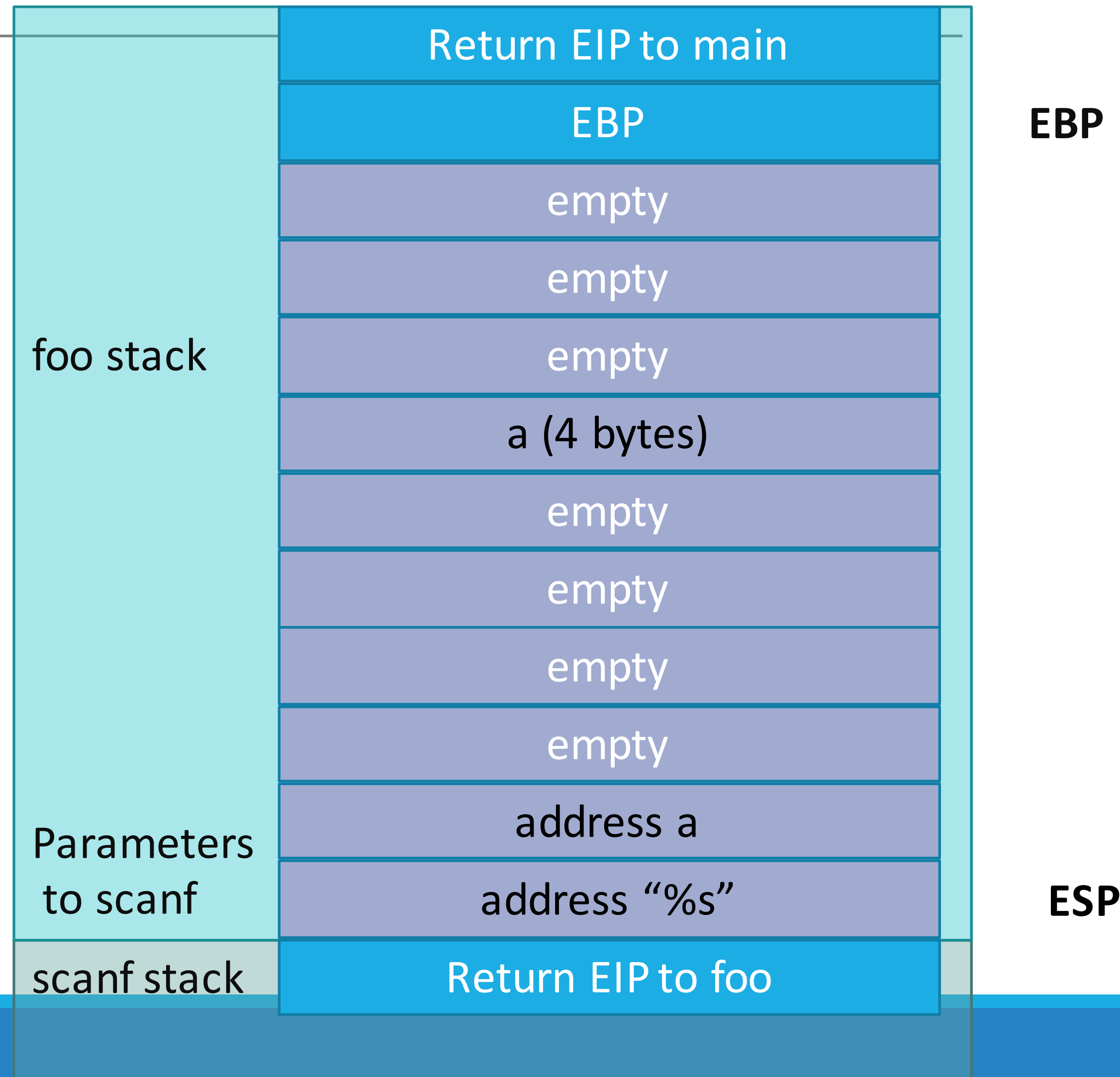
```
int foo(int bar)
{
  char a[4];
  scanf("%s",a);
}
```

gcc -S –masm=intel –fno-stack-protector bo.c

# bo.s

```
.LC0:
  .string "%s"
  .text

foo:
push   ebp
mov  ebp, esp
sub  esp, 40
mov  eax, OFFSET FLAT:.LC0
lea  edx, [ebp-12]
mov  DWORD PTR [esp+4], edx
mov  DWORD PTR [esp], eax
call   __isoc99_scanf
Leave
ret
```



# Buffer Overflow

```
[jpbarraca@atnog: seguranca]$ ./bo
a
[jpbarraca@atnog: seguranca]$ ./bo
aa
[jpbarraca@atnog: seguranca]$ ./bo
aaaaaaaaaaaa
[jpbarraca@atnog: seguranca]$ ./bo
aaaaaaaaaaaaa
Segmentation fault
```

Write inside a

Write inside a

Write outside a

Write over stored EBP