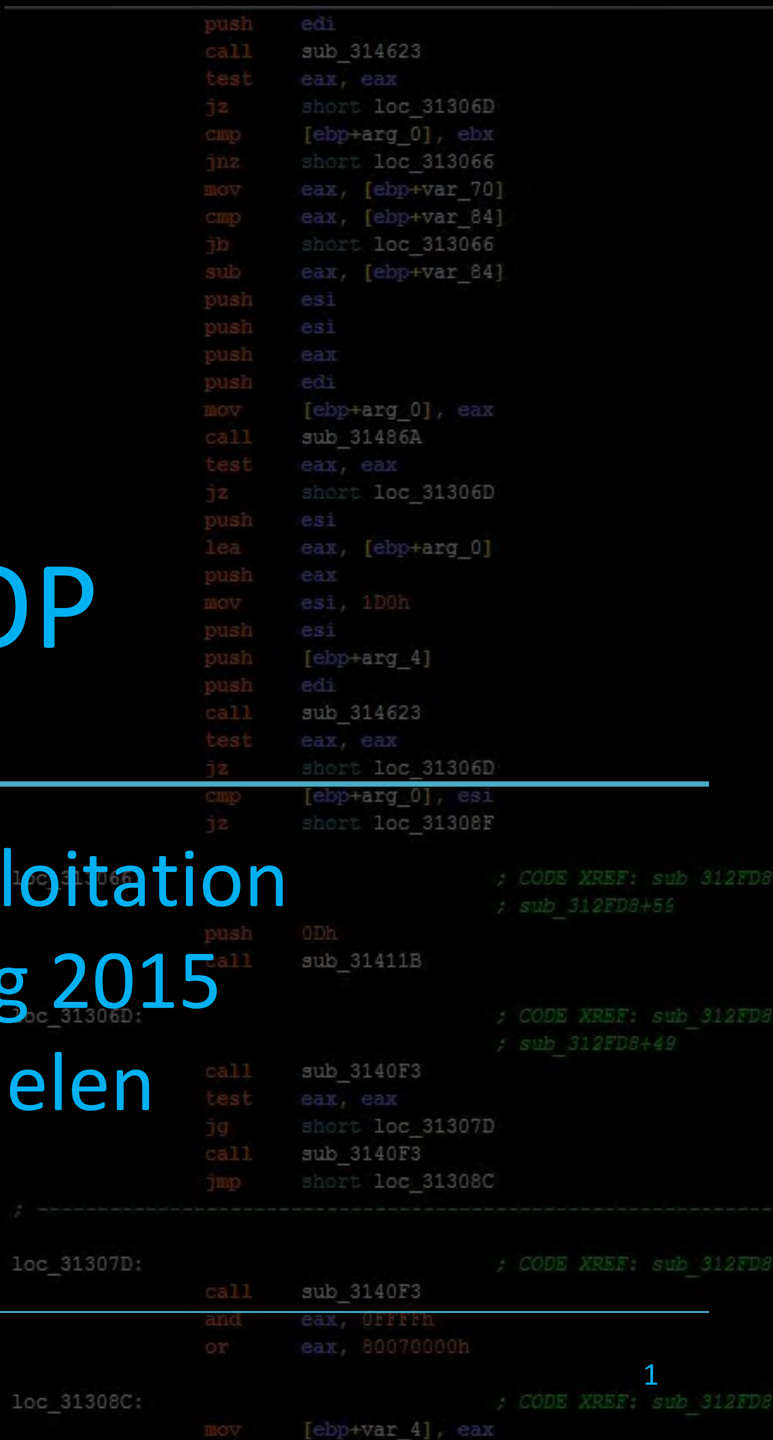
DEP & ROP

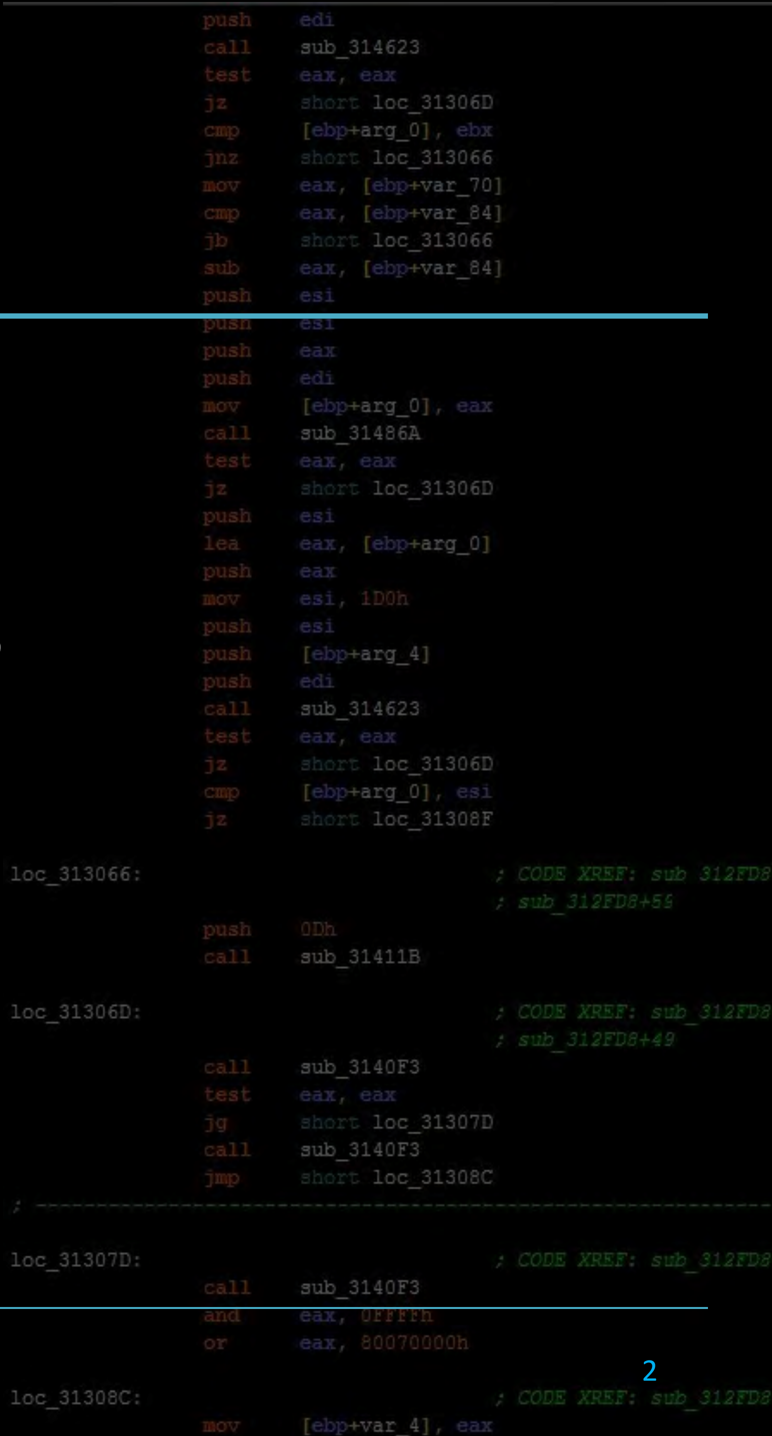
Modern Binary Exploitation
CSCI 4968 - Spring 2015
Markus Gaasedelen



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Lecture Overview

1. Introducing DEP
2. The History of DEP
3. Bypassing DEP with ROP
4. Stack Pivoting



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Class up until Now

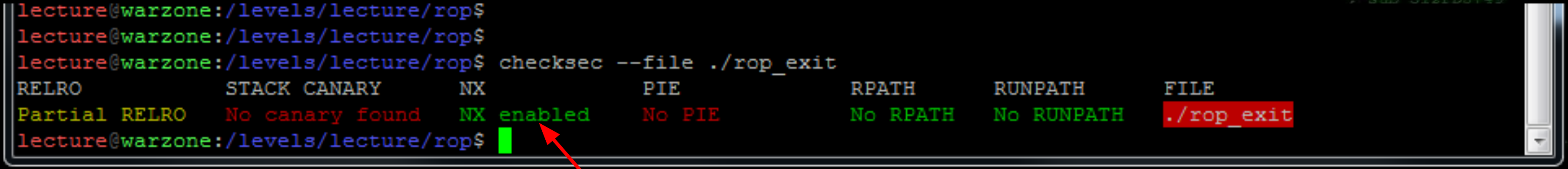
- Reverse Engineering
- Basic memory corruption
- Shellcoding
- Format strings
- Classical exploitation, few protections, pretty eZ
- Time to add some 'modern' to the binary exploitation madness

Modern Exploit Mitigations

- Theres a number of modern exploit mitigations that we've generally been turning off for the labs and exercises
  - DEP
  - ASLR
  - Stack Canaries
  - … ?

Modern Exploit Mitigations

- Theres a number of modern exploit mitigations that we've generally been turning off for the labs and exercises
  - DEP
  - ASLR
  - Stack Canaries
  - … ?
- Today we turn one back on for the remainder of the course
  - no more silly -z execstack in our gcc commands

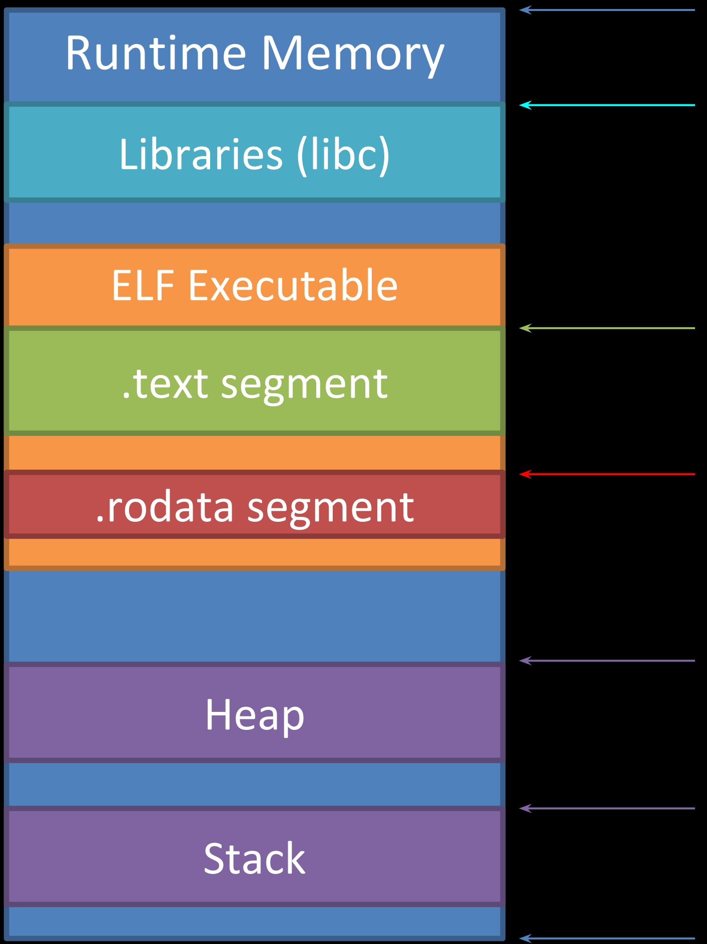


```
lecture@warzone:/levels/lecture/rop$
lecture@warzone:/levels/lecture/rop$
lecture@warzone:/levels/lecture/rop$ checksec --file ./rop_exit
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   ./rop_exit
lecture@warzone:/levels/lecture/rop$
```

Course Terminology

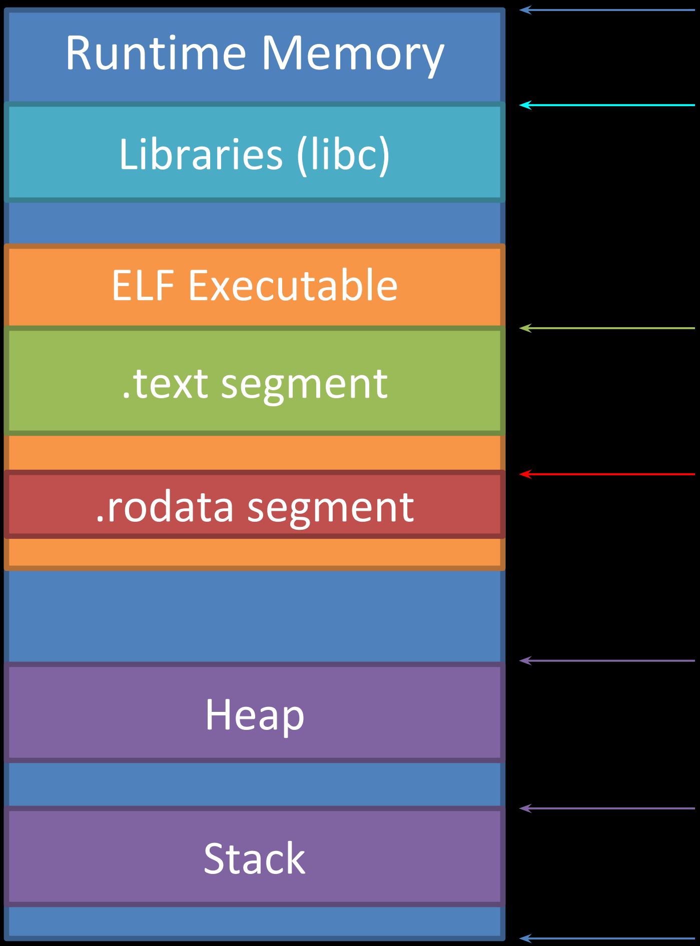
- Data Execution Prevention
  - An exploit mitigation technique used to ensure that only code segments are ever marked as executable
  - Meant to mitigate code injection / shellcode payloads
  - Also known as DEP, NX, XN, XD, W^X

Runtime Process Without DEP



```
0x00000000 – Start of memory

Like an ELF, multiple segments
    R-X
    R-- ...

R-X (Read, Execute)

R-- (Read)

RWX (Read, Write, Execute)

RWX (Read, Write, Execute)

0xFFFFFFFF – End of memory
```

Runtime Process Without DEP



0x00000000 – Start of memory

Like an ELF, multiple segments
    R-X
    R-- ...

R-X (Read, Execute)

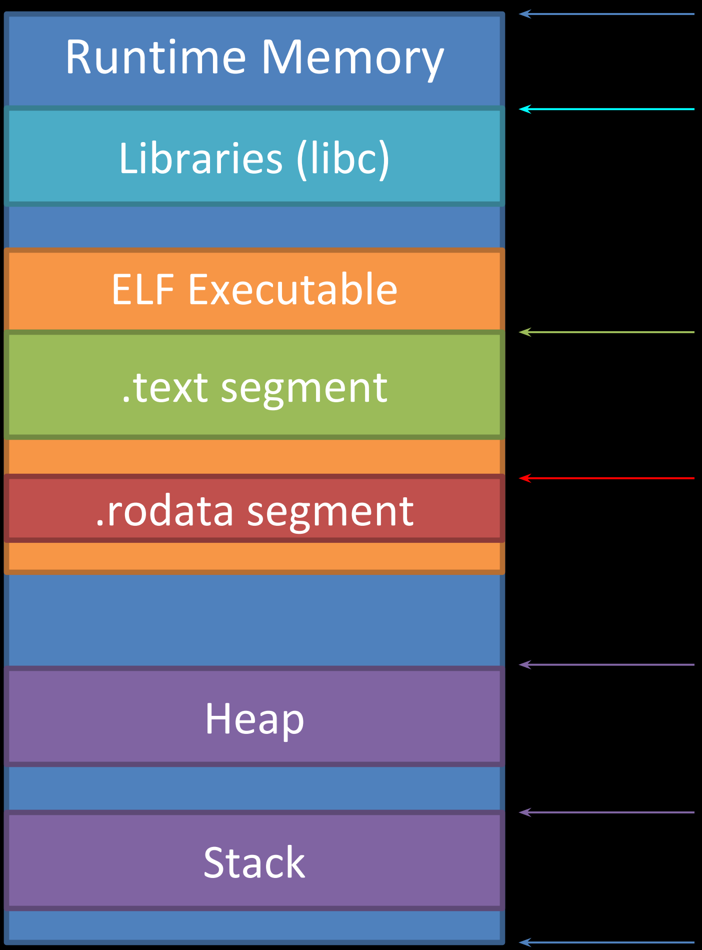
R-- (Read)

RWX (Read, Write, Execute)

RWX (Read, Write, Execute)

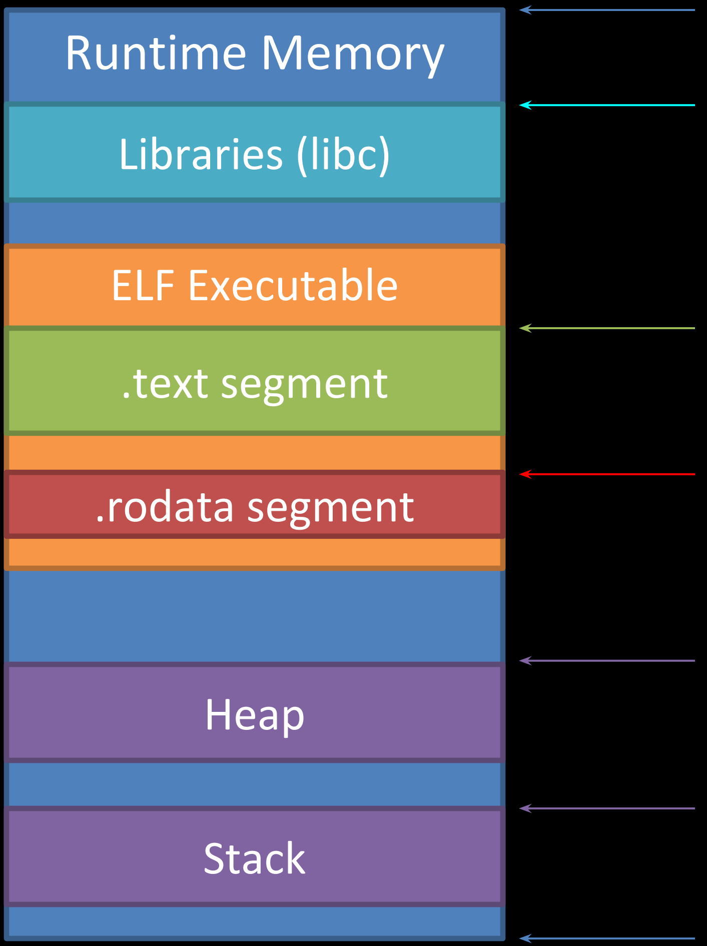
0xFFFFFFFF – End of memory

Runtime Process With~~out~~ DEP



```
0x00000000 – Start of memory

Like an ELF, multiple segments
    R-X
    R-- ...

R-X (Read, Execute)

R-- (Read)

RW- (Read, Write, ~~Execute~~)

RW- (Read, Write, ~~Execute~~)

0xFFFFFFFF – End of memory
```

Runtime Process With DEP



0x00000000 – Start of memory

Like an ELF, multiple segments
R-X
R-- ...

R-X (Read, Execute)

R-- (Read)

RW- (Read, Write)

RW- (Read, Write)

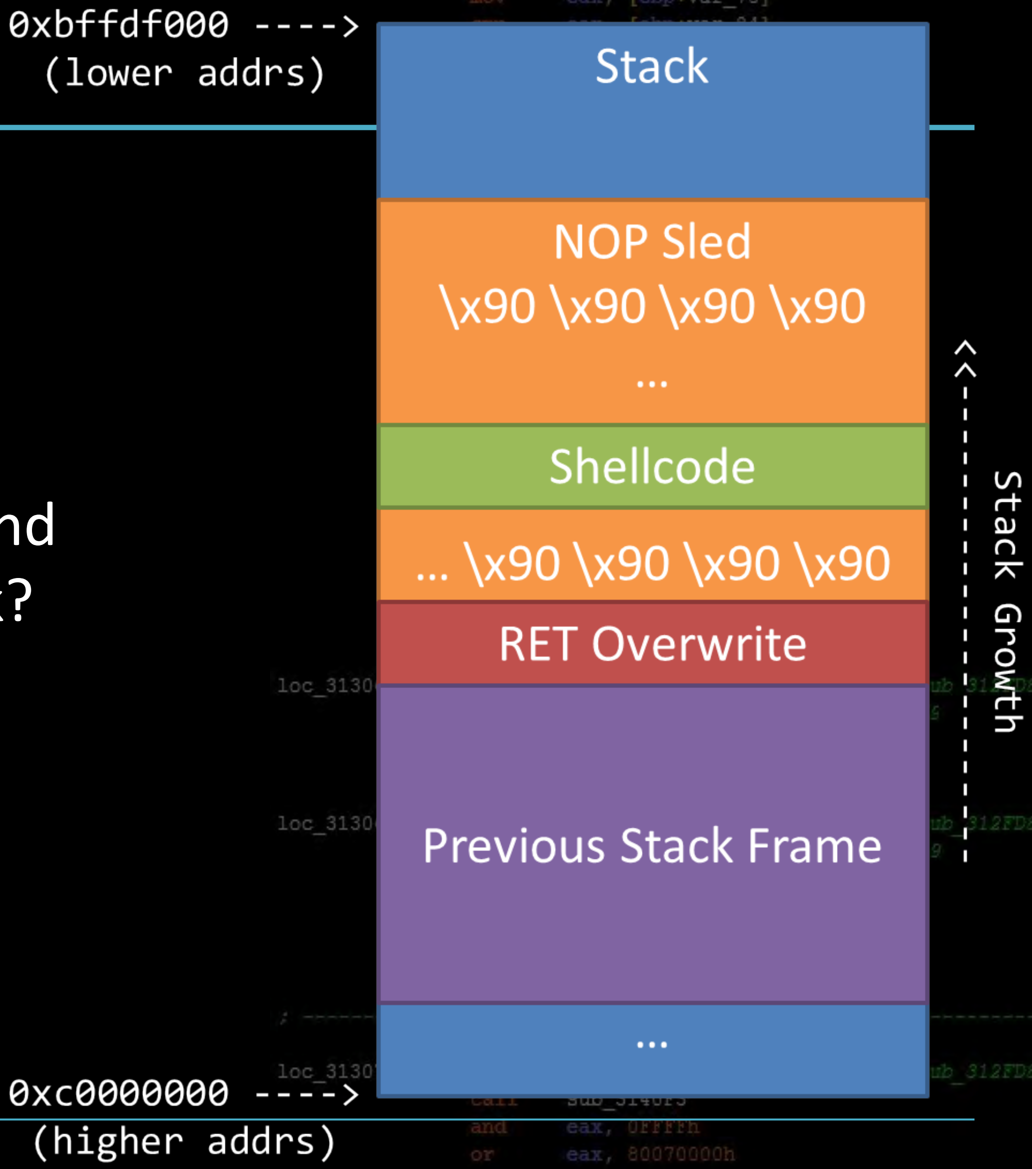
0xFFFFFFFF – End of memory

DEP Basics

- No segment of memory should ever be Writable and Executable at the same time, ‘W^X’
- Common data segments
  - Stack, Heap
  - .bss
  - .ro
  - .data
- Common code segments
  - .text
  - .plt

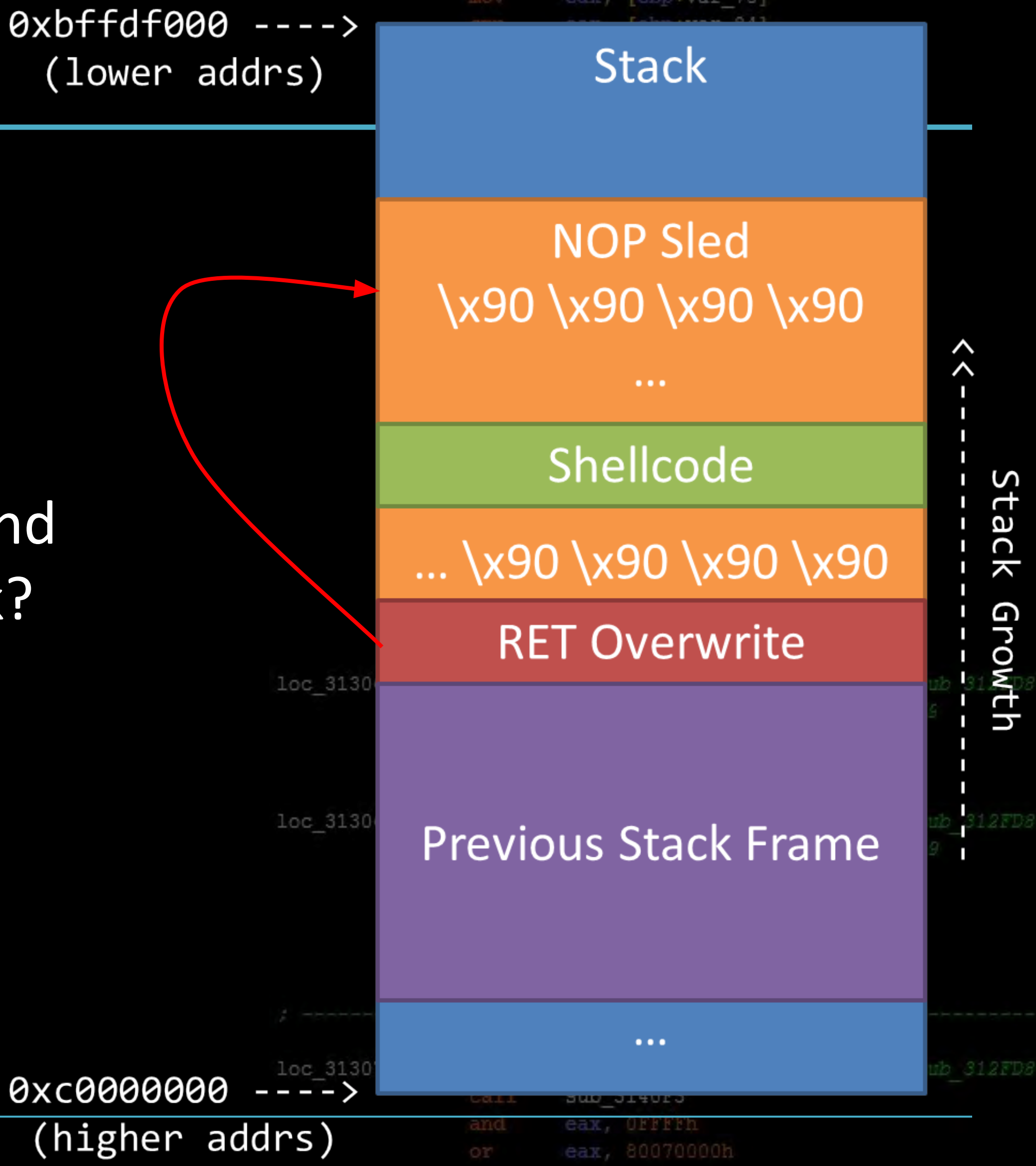
DEP in Action

- Data should never be executable, only code
- What happens if we stack smash, inject shellcode, and try to jump onto the stack?



DEP in Action

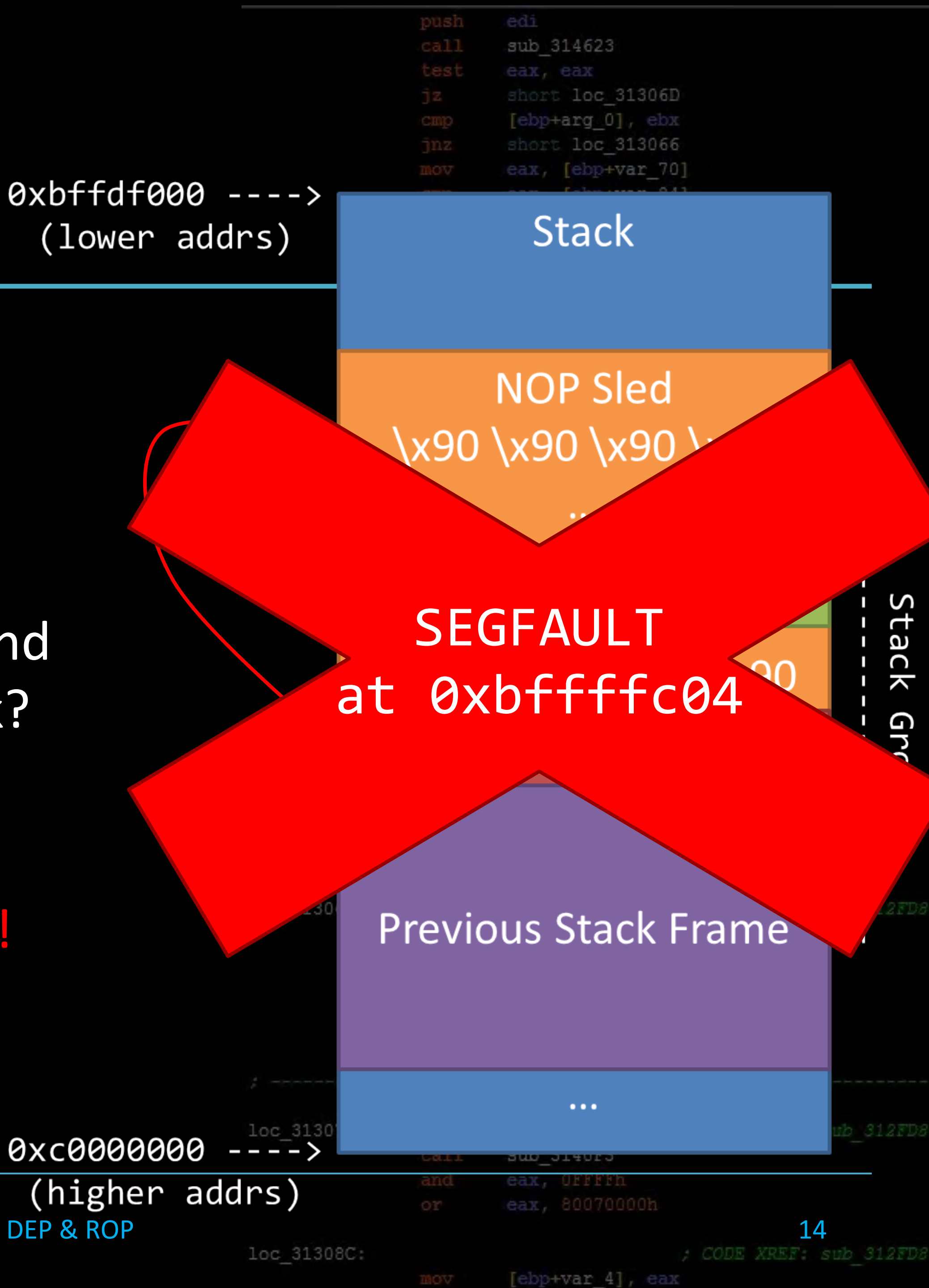
- Data should never be executable, only code
- What happens if we stack smash, inject shellcode, and try to jump onto the stack?



DEP in Action

- Data should never be executable, only code
- What happens if we stack smash, inject shellcode, and try to jump onto the stack?

yay mitigation technologies!



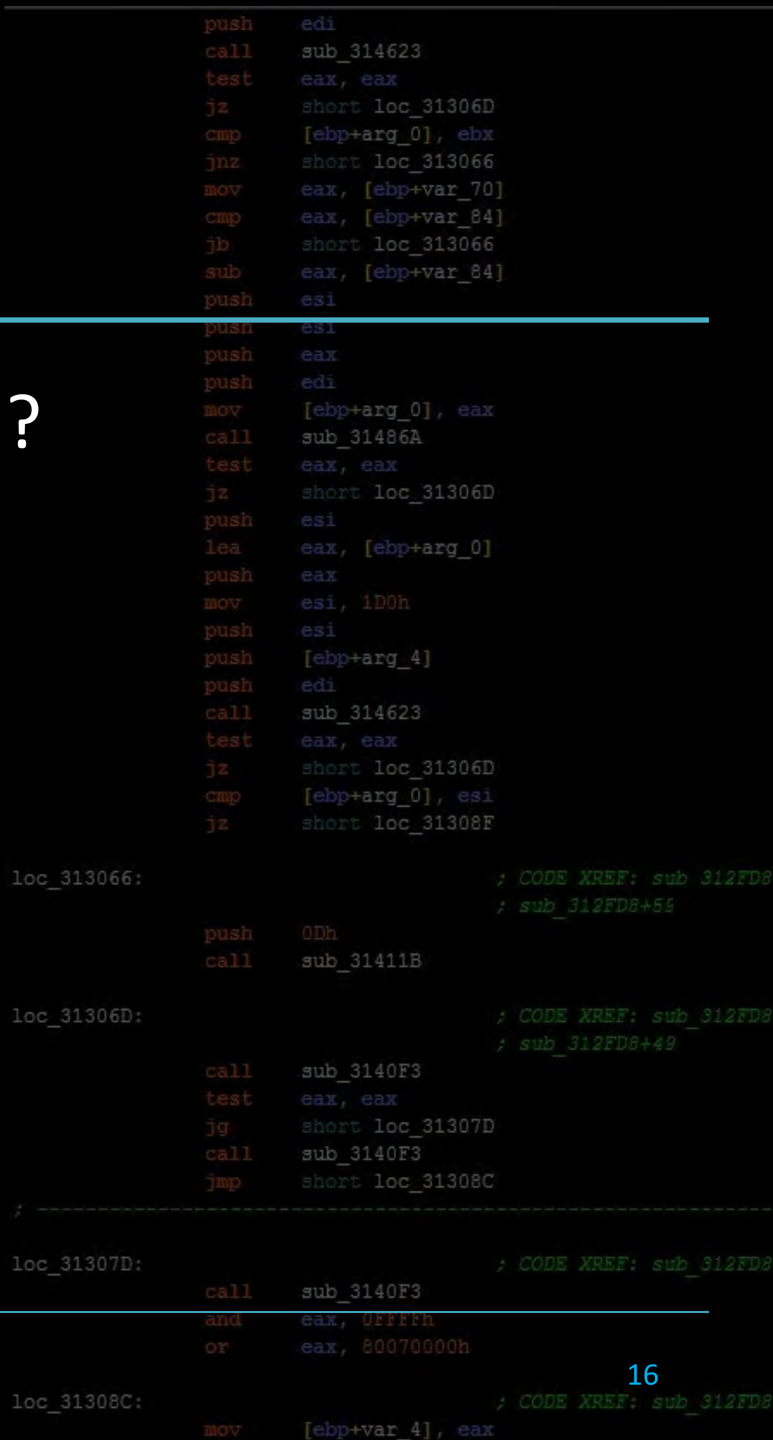
Lecture Overview

1. Introducing DEP
2. The History of DEP
3. Bypassing DEP with ROP
4. Stack Pivoting

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

History of DEP

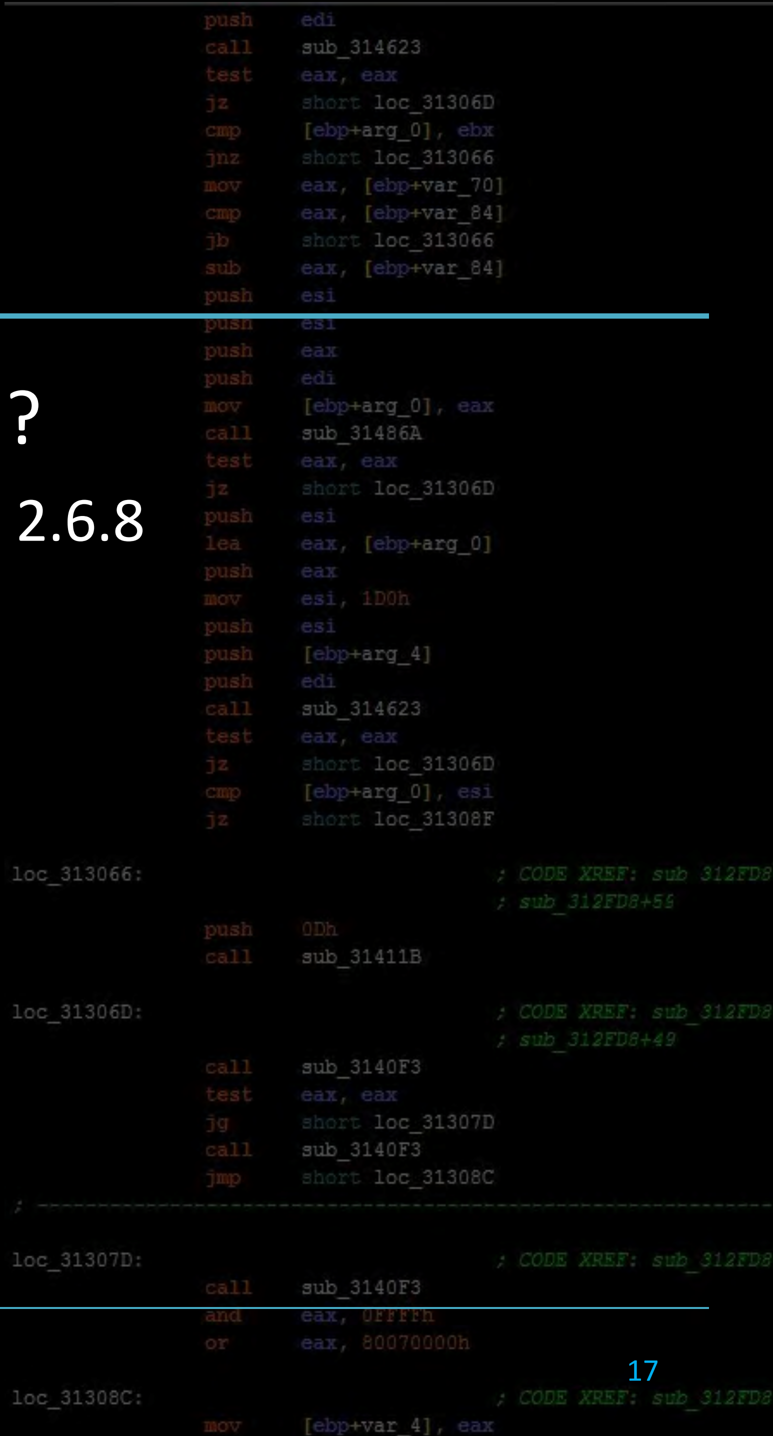
- When was DEP implemented?



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

History of DEP

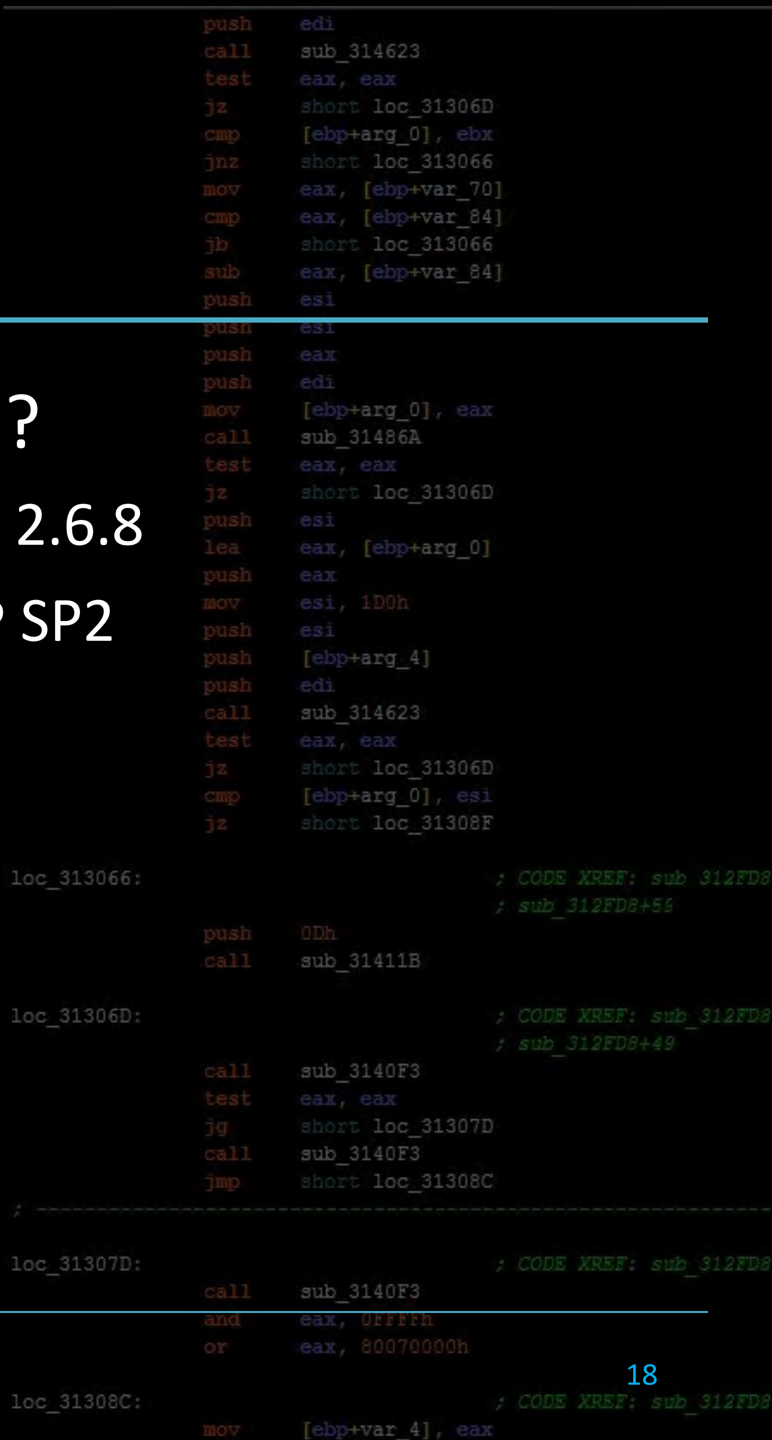
- When was DEP implemented?
  - August 14th, 2004 - Linux Kernel 2.6.8



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

History of DEP

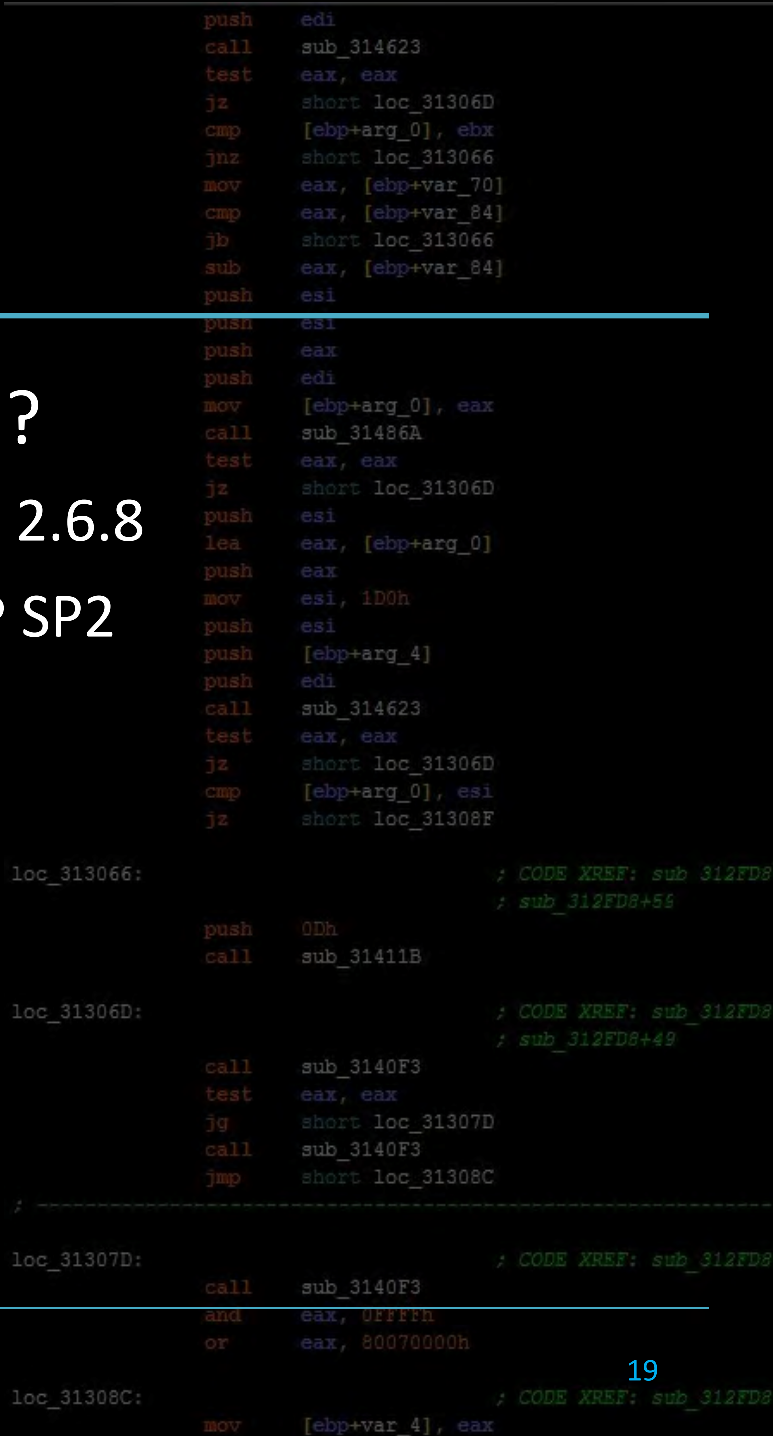
- When was DEP implemented?
  - August 14th, 2004 - Linux Kernel 2.6.8
  - August 25th, 2004 - Windows XP SP2



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

History of DEP

- When was DEP implemented?
  - August 14th, 2004 - Linux Kernel 2.6.8
  - August 25th, 2004 - Windows XP SP2
  - June 26th, 2006 - Mac OSX 10.5



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

History of DEP

- When was DEP implemented?
  - August 14th, 2004 - Linux Kernel 2.6.8
  - August 25th, 2004 - Windows XP SP2
  - June 26th, 2006 - Mac OSX 10.5

about 10 years ago

2004 in Perspective

- Facebook is created
- G-Mail launches as beta
- Ken Jennings begins his 74 win streak on Jeopardy
- Halo 2 is released, as is Half Life 2
- LOST airs its first episode

Security is Young

- Technologies in modern exploit mitigations are incredibly young, and the field of computer security is rapidly evolving

- DEP is one of the of the main mitigation technologies you must bypass in modern exploitation

Lecture Overview

1. Introducing DEP
2. The History of DEP
3. Bypassing DEP with ROP
4. Stack Pivoting

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Bypassing DEP

- DEP stops an attacker from easily executing injected shellcode assuming they gain control of EIP
    - shellcode almost always ends up in a RW- region

- If you can't inject (shell)code to do your bidding, you must re-use the existing code!
    - This is technique is usually some form of ROP

Course Terminology

- Return Oriented Programming
  - A technique in exploitation to reuse existing code gadgets in a target binary as a method to bypass DEP
  - Also known as ROP

- Gadget
  - A sequence of meaningful instructions typically followed by a return instruction
  - Usually multiple gadgets are chained together to compute malicious actions like shellcode does
  - These chains are called ROP Chains

Relevant Quotes

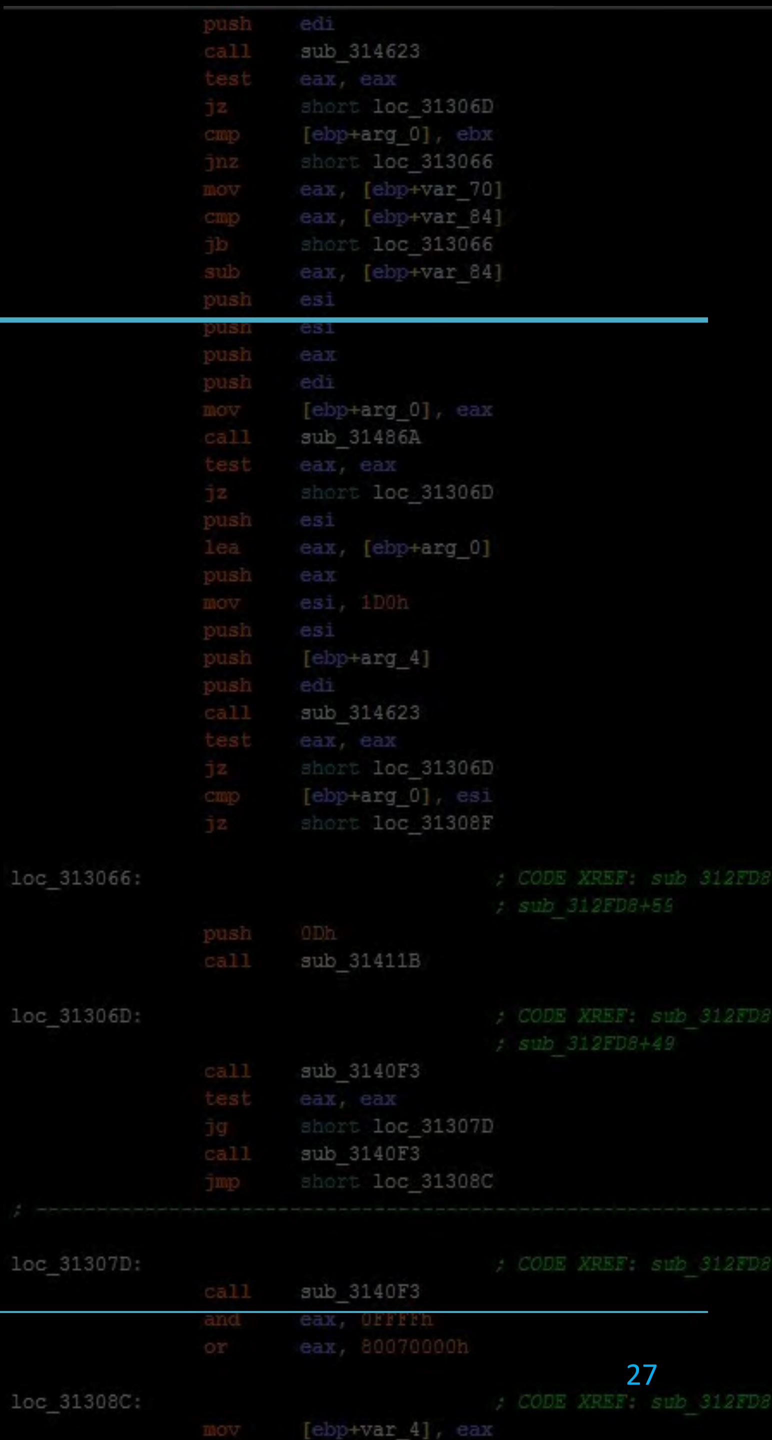
"Preventing the introduction of malicious code is not enough to prevent the execution of malicious computations"

-Dino Dai Zovi

Gadgets

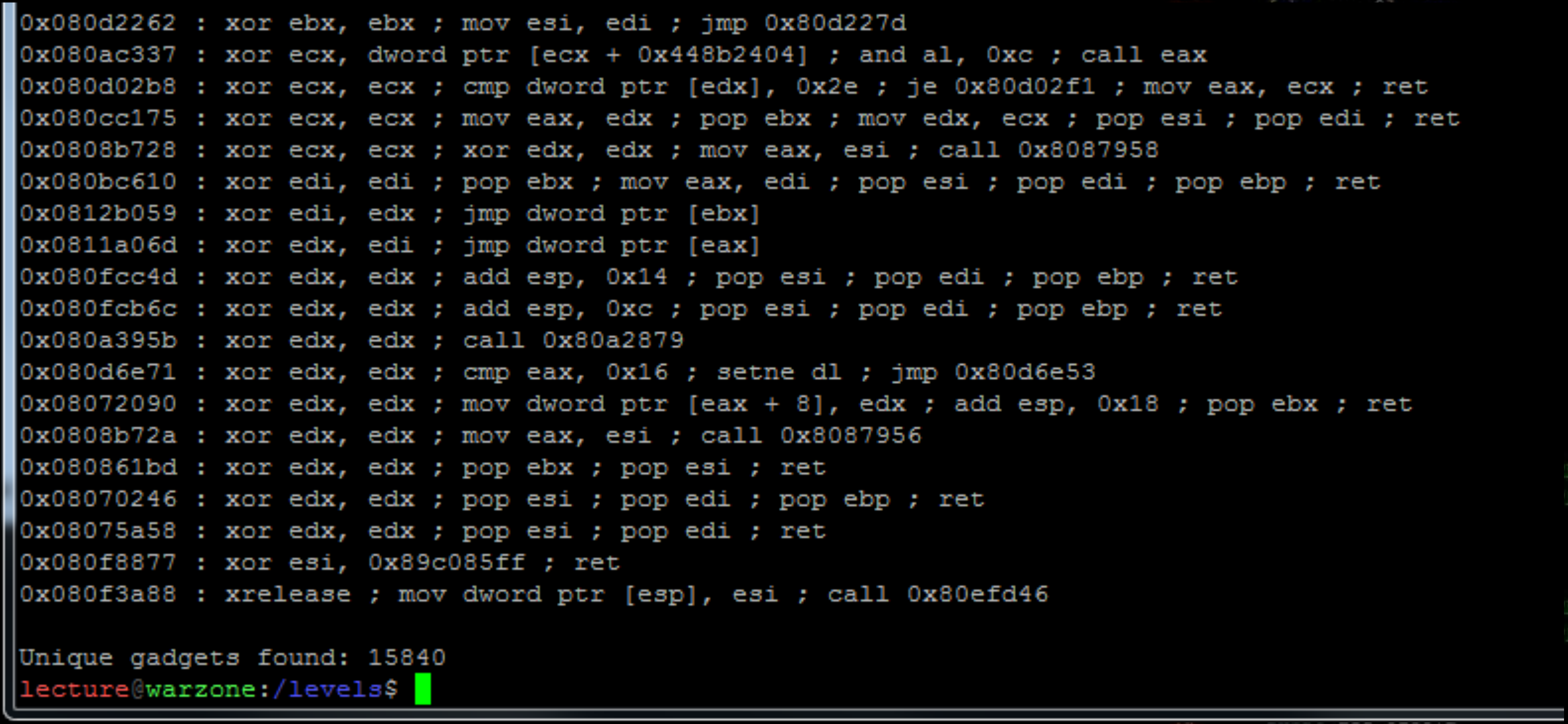
- ROP Chains are made up of gadgets
- Example gadgets -

```
xor    eax, eax
ret

pop    ebx
pop    eax
ret

add   eax, ebx
ret
```



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

$ ropgadget --binary /bin/bash



```
0x080d2262 : xor ebx, ebx ; mov esi, edi ; jmp 0x80d227d
0x080ac337 : xor ecx, dword ptr [ecx + 0x448b2404] ; and al, 0xc ; call eax
0x080d02b8 : xor ecx, ecx ; cmp dword ptr [edx], 0x2e ; je 0x80d02f1 ; mov eax, ecx ; ret
0x080cc175 : xor ecx, ecx ; mov eax, edx ; pop ebx ; mov edx, ecx ; pop esi ; pop edi ; ret
0x0808b728 : xor ecx, ecx ; xor edx, edx ; mov eax, esi ; call 0x8087958
0x080bc610 : xor edi, edi ; pop ebx ; mov eax, edi ; pop esi ; pop edi ; pop ebp ; ret
0x0812b059 : xor edi, edx ; jmp dword ptr [ebx]
0x0811a06d : xor edx, edi ; jmp dword ptr [eax]
0x080fcc4d : xor edx, edx ; add esp, 0x14 ; pop esi ; pop edi ; pop ebp ; ret
0x080fcb6c : xor edx, edx ; add esp, 0xc ; pop esi ; pop edi ; pop ebp ; ret
0x080a395b : xor edx, edx ; call 0x80a2879
0x080d6e71 : xor edx, edx ; cmp eax, 0x16 ; setne dl ; jmp 0x80d6e53
0x08072090 : xor edx, edx ; mov dword ptr [eax + 8], edx ; add esp, 0x18 ; pop ebx ; ret
0x0808b72a : xor edx, edx ; mov eax, esi ; call 0x8087956
0x080861bd : xor edx, edx ; pop ebx ; pop esi ; ret
0x08070246 : xor edx, edx ; pop esi ; pop edi ; pop ebp ; ret
0x08075a58 : xor edx, edx ; pop esi ; pop edi ; ret
0x080f8877 : xor esi, 0x89c085ff ; ret
0x080f3a88 : xrelease ; mov dword ptr [esp], esi ; call 0x80efd46

Unique gadgets found: 15840
lecture@warzone:/levels$
```

Understanding ROP

- It is almost always possible to create a logically equivalent ROP chain for a given piece of shellcode

exit(0) - shellcode

```
xor     eax, eax
xor     ebx, ebx
inc     eax
int     0x80
```

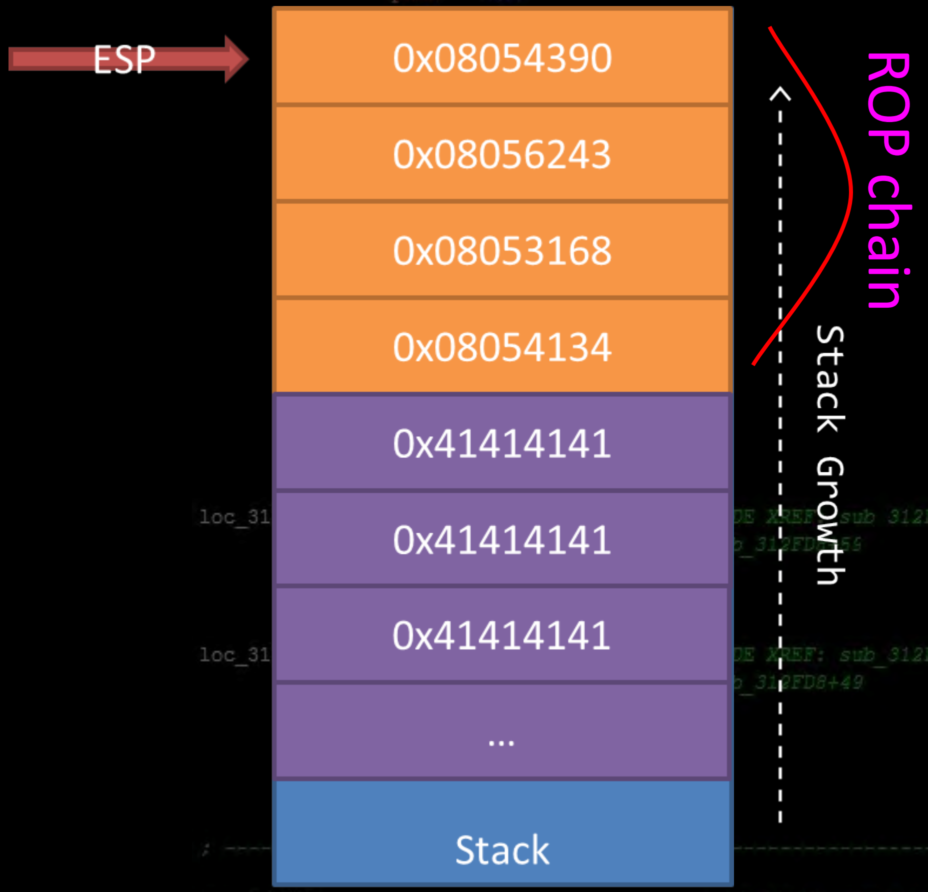
exit(0) - ROP chain

```
xor     eax, eax
ret
.......................
xor     ebx, ebx
ret
.......................
inc     eax
ret
.......................
int     0x80
```

Understanding ROP

exit(0) - ROP chain

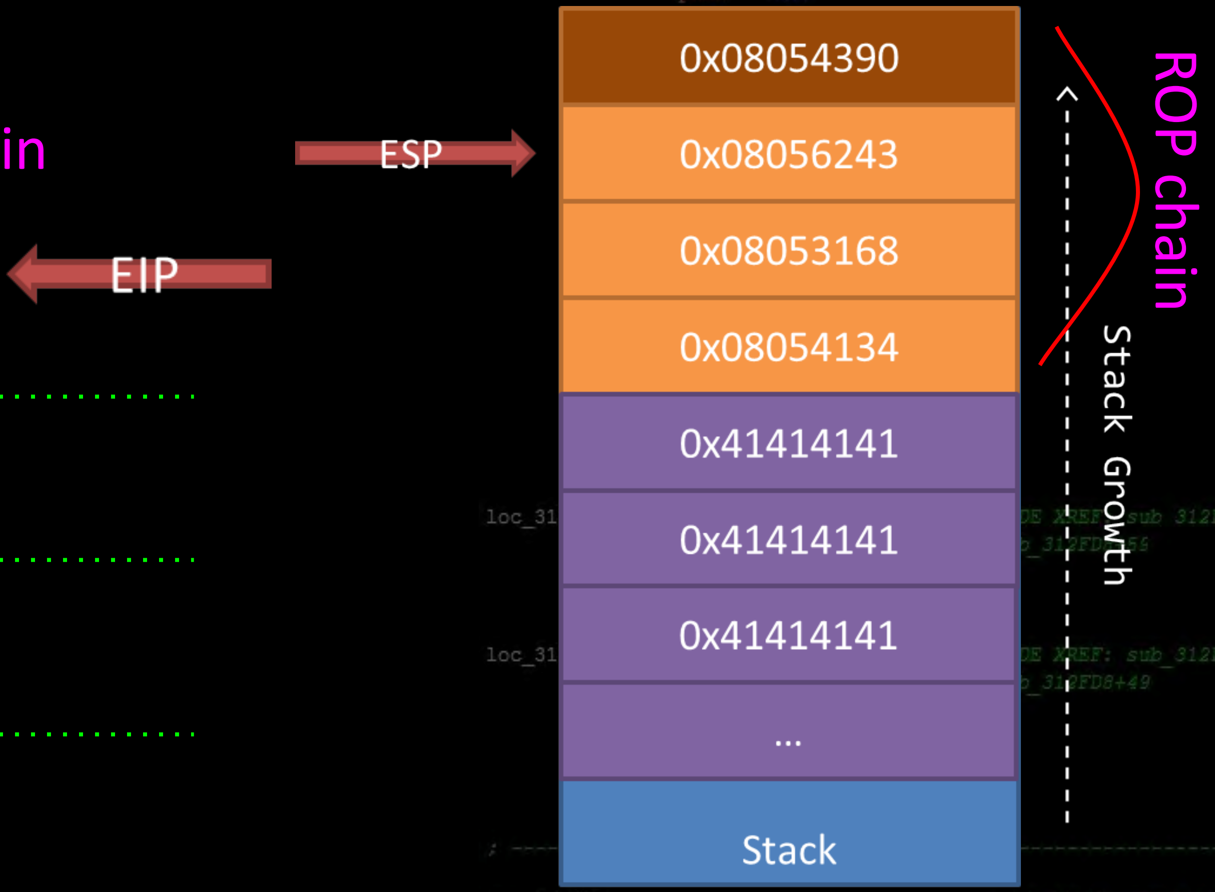
```
xor     eax, eax
ret
.....................................................
xor     ebx, ebx
ret
.....................................................
inc     eax
ret
.....................................................
int     0x80
```



Understanding ROP

exit(0) - ROP chain

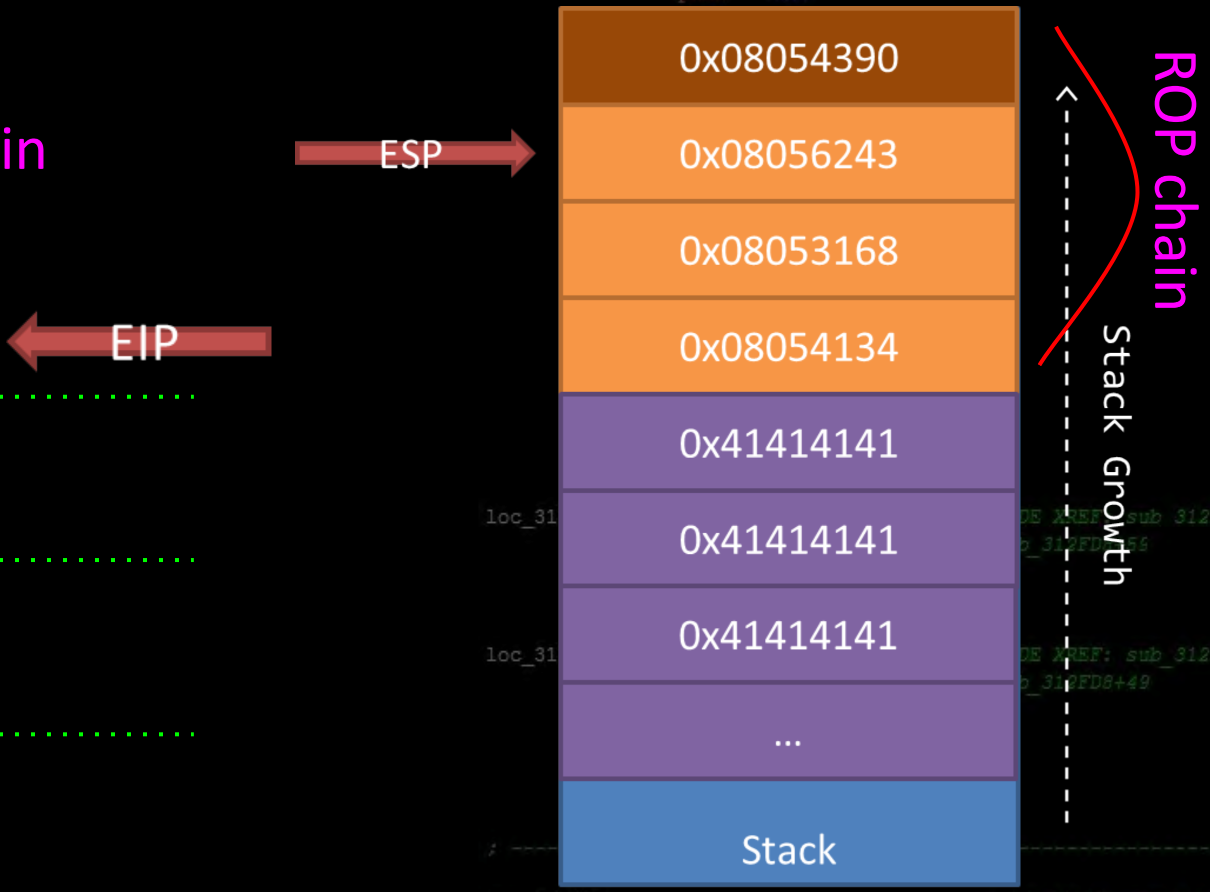
```
xor     eax, eax
ret
.........................................
xor     ebx, ebx
ret
.........................................
inc     eax
ret
.........................................
int     0x80
```



Understanding ROP

exit(0) - ROP chain

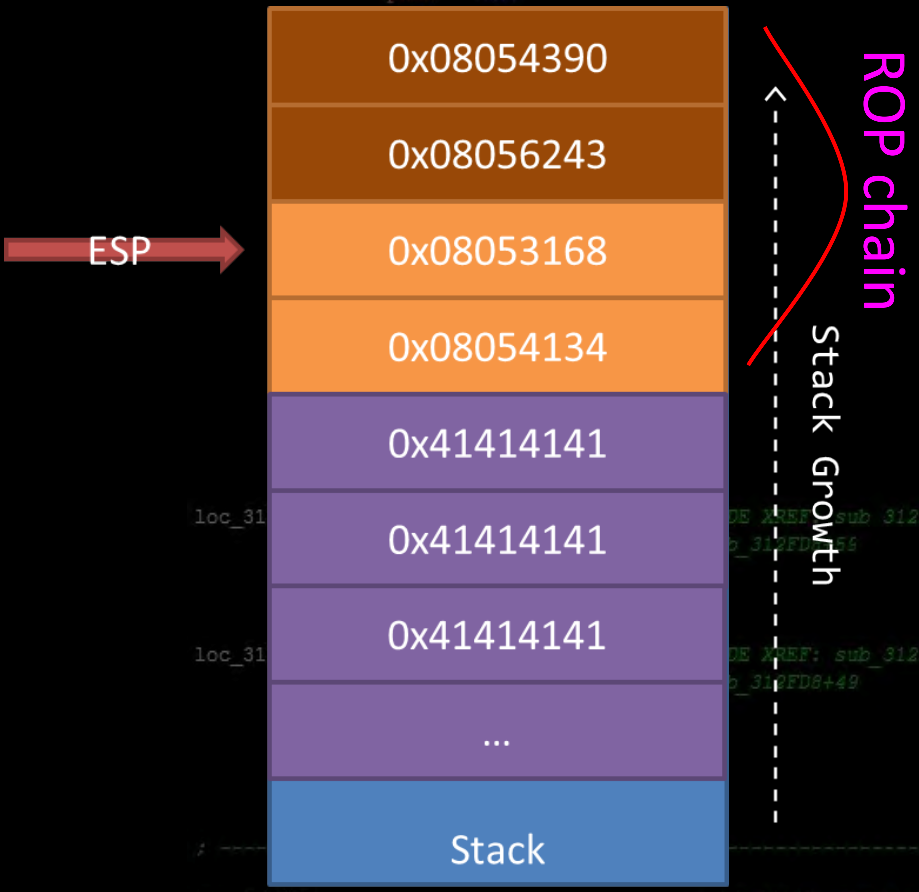
```
xor     eax, eax
ret
.............................................
xor     ebx, ebx
ret
.............................................
inc     eax
ret
.............................................
int     0x80
```



Understanding ROP

exit(0) - ROP chain

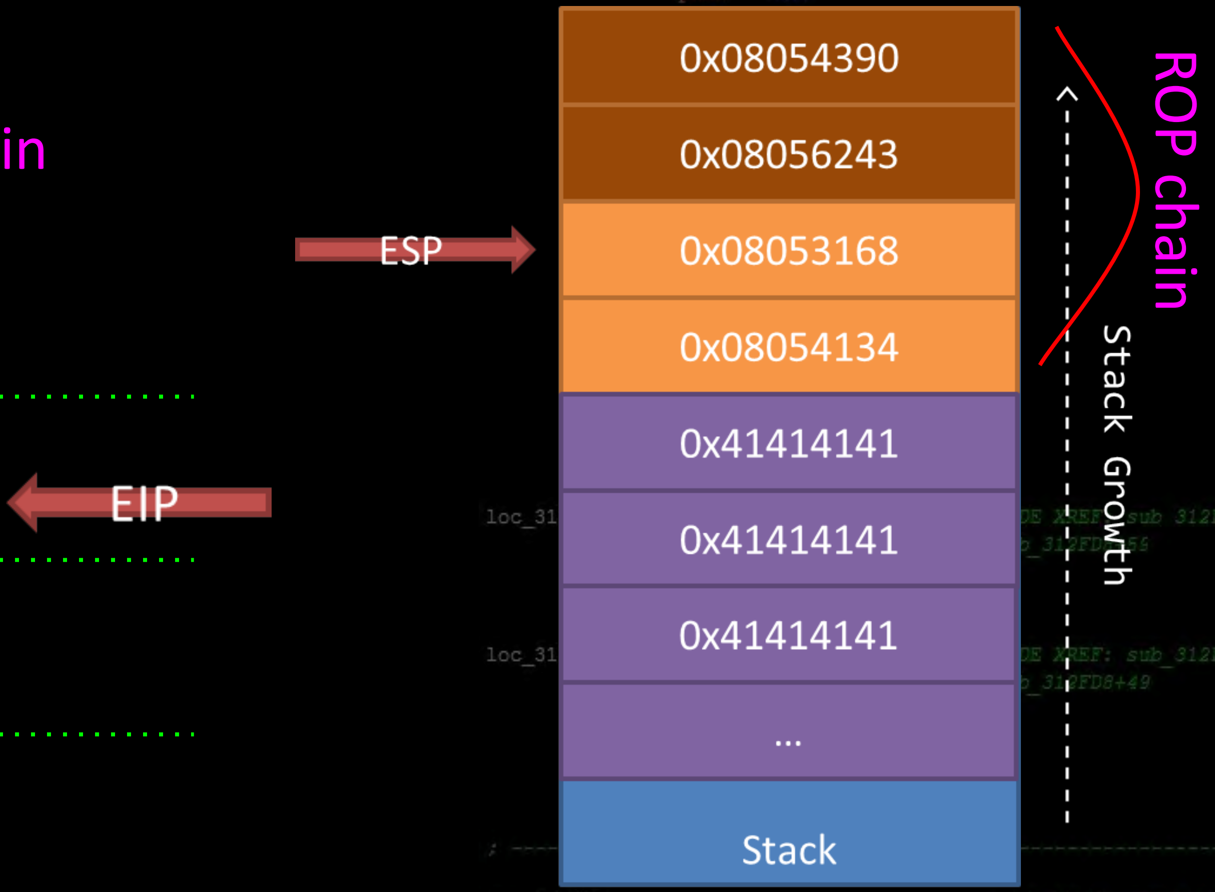
```
xor     eax, eax
ret
.........................................
xor     ebx, ebx      EIP
ret
.........................................
inc     eax
ret
.........................................
int     0x80
```



Understanding ROP

exit(0) - ROP chain

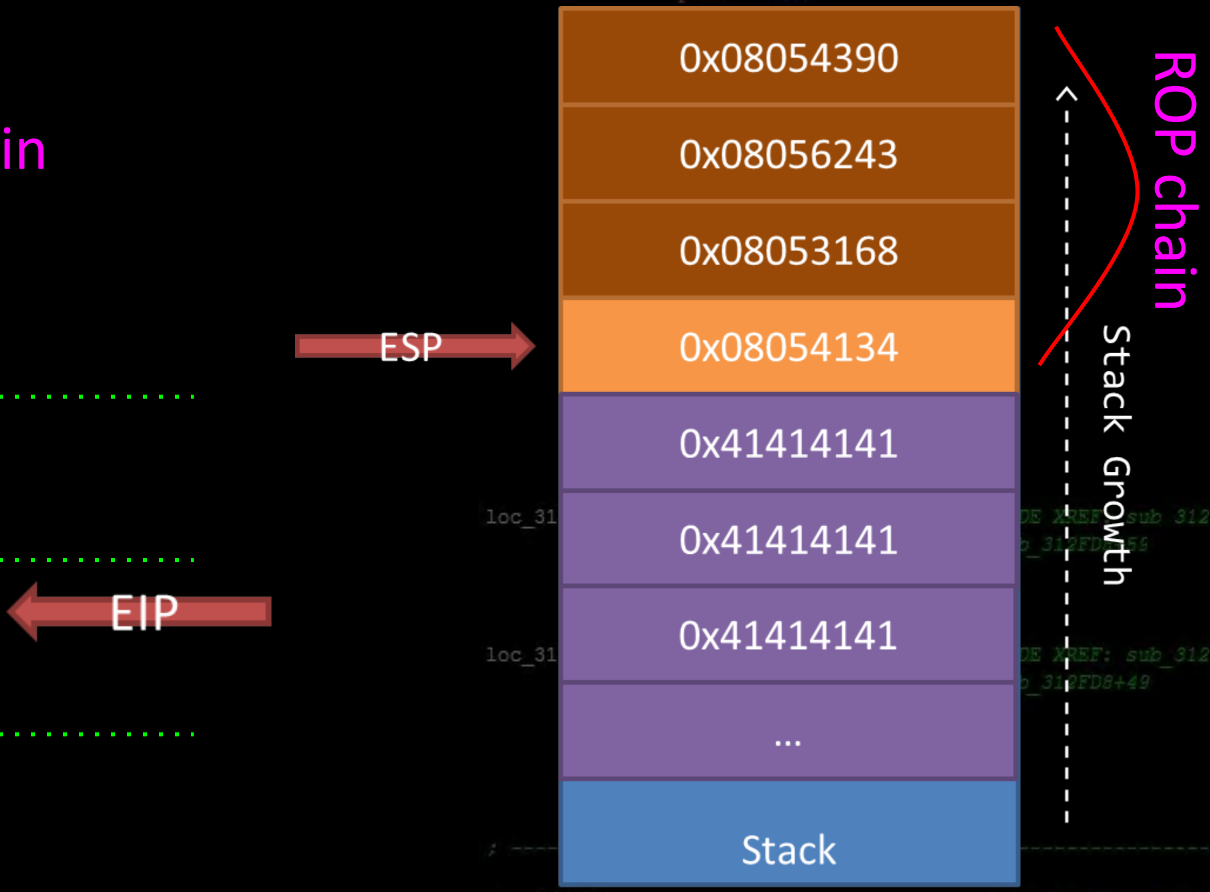
```
xor     eax, eax
ret
.................................................
xor     ebx, ebx
ret
.................................................
inc     eax
ret
.................................................
int     0x80
```



Understanding ROP

exit(0) - ROP chain

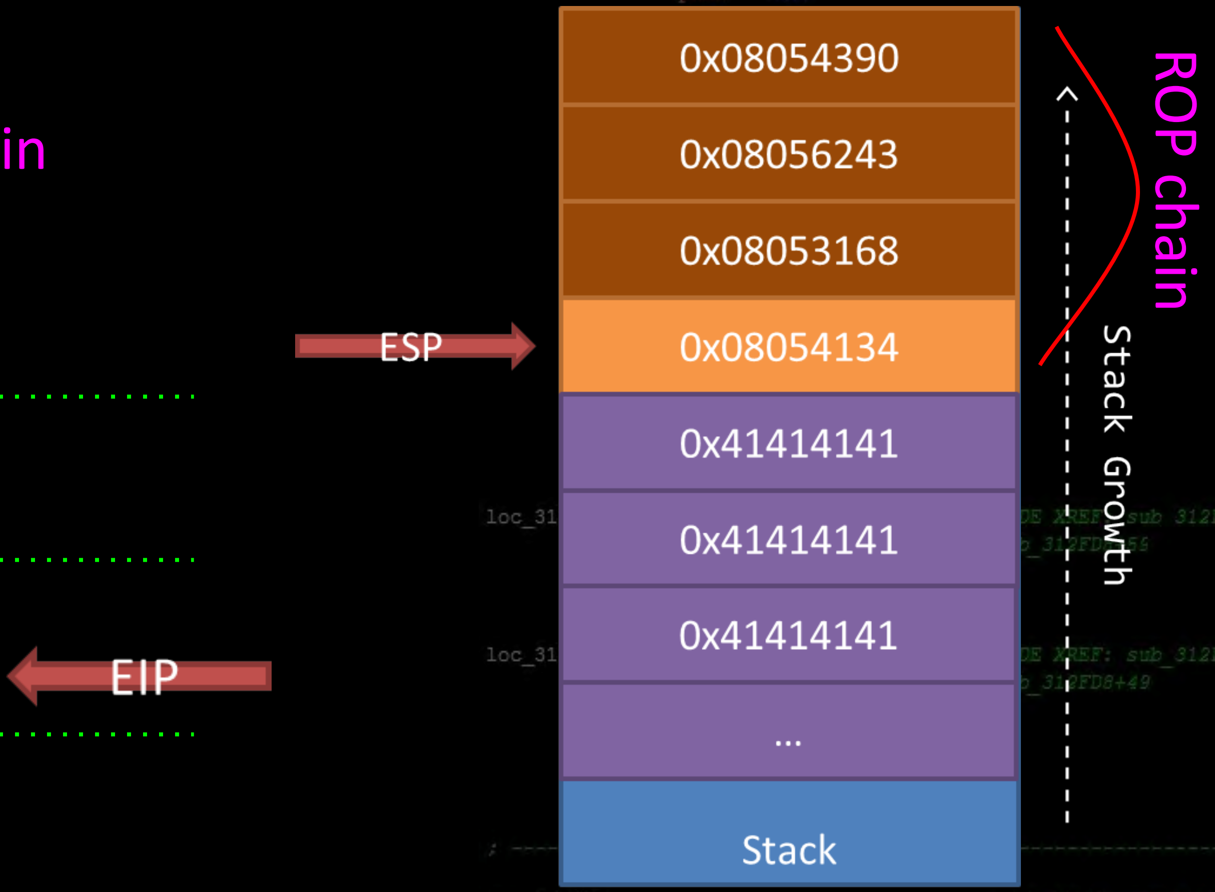
```
xor     eax, eax
ret
.............................................
xor     ebx, ebx
ret
.............................................
inc     eax
ret
.............................................
int     0x80
```



Understanding ROP

exit(0) - ROP chain

```
xor     eax, eax
ret
...............................................
xor     ebx, ebx
ret
...............................................
inc     eax
ret
...............................................
int     0x80
```

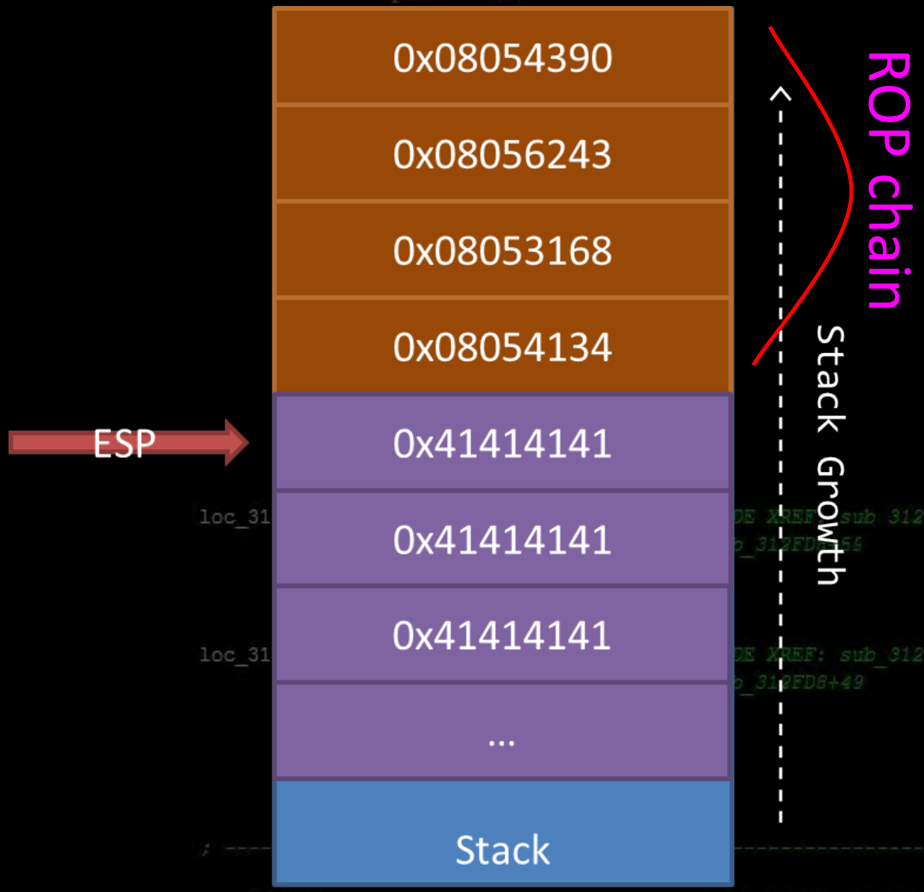


Understanding ROP

exit(0) - ROP chain

```
xor     eax, eax
ret
.......................................
xor     ebx, ebx
ret
.......................................
inc     eax
ret
.......................................
int     0x80
```

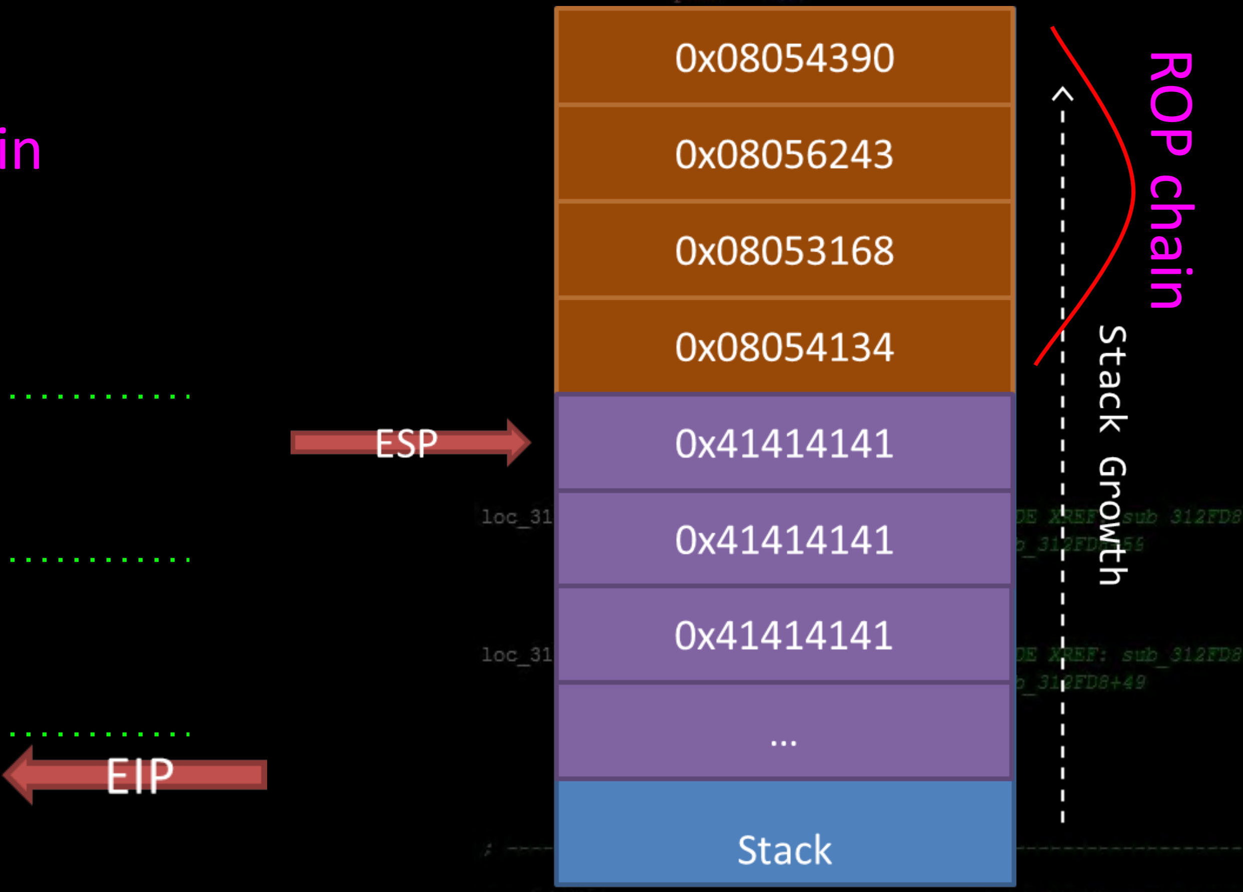
EIP



Understanding ROP

exit(0) - ROP chain

```
xor     eax, eax
ret
.........................................
xor     ebx, ebx
ret
.........................................
inc     eax
ret
.........................................
int     0x80
exits ...
```



Bypassing DEP with ROP

- We called exit(0) without using any sort of shellcode!
- With that said, writing ROP can be difficult and you will usually have to get creative with what gadgets you find

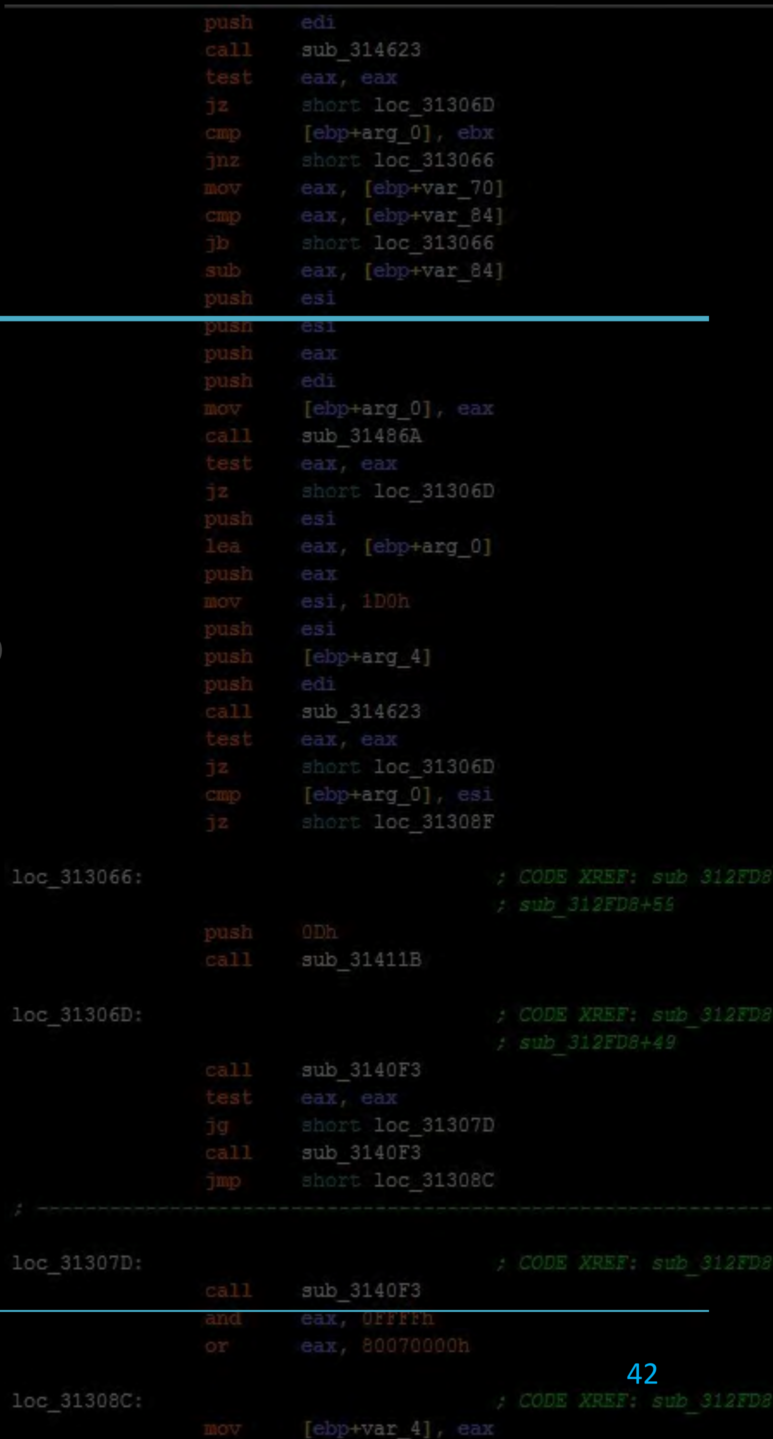
/levels/lecture/rop/rop_exit

- Play around with ROP on the warzone
- Can you make a ROP chain to set arbitrary exit values? 0? 200? 64?

Relevant Tips/Tools/Commands

- `$ ropgadget --binary ./rop_exit > /tmp/gadgetzXYZ.txt`
  - `$ cat /tmp/gadgetzXYZ.txt | grep "pop eax" | grep …`
- `$ asm`
  - easy way to get the bytes for gadgets you're looking for
- `$ gdbpeda`
  - searchmem, find raw bytes in an executing program
  - ropsearch, a crappy rop gadget finder
- python

```
def q(addr):
    return struct.pack("I", addr)
```



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Lecture Overview

1. Introducing DEP
2. The History of DEP
3. Bypassing DEP with ROP
4. Stack Pivoting

Typical Constraints in ROP

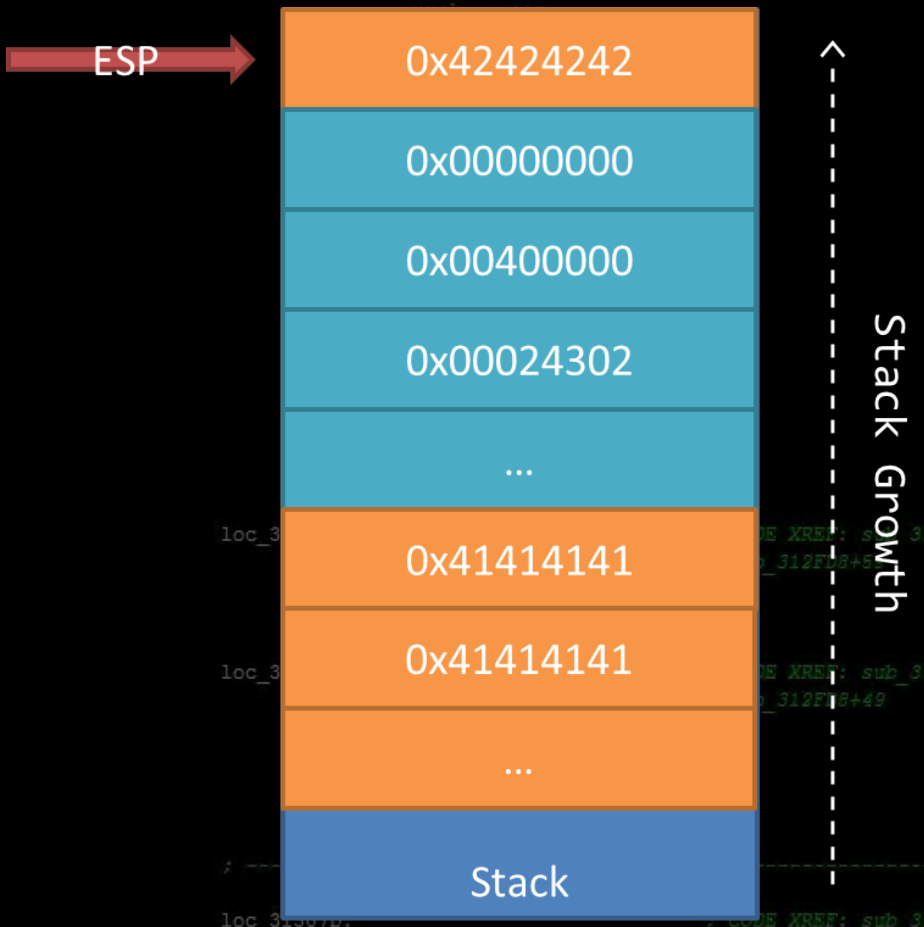
- Typically in modern exploitation you might only get one targeted overwrite rather than a straight stack smash

- What can you do when you only have one gadget worth of execution?
  - Answer: Stack Pivoting

Stack Pivoting

You control the orange

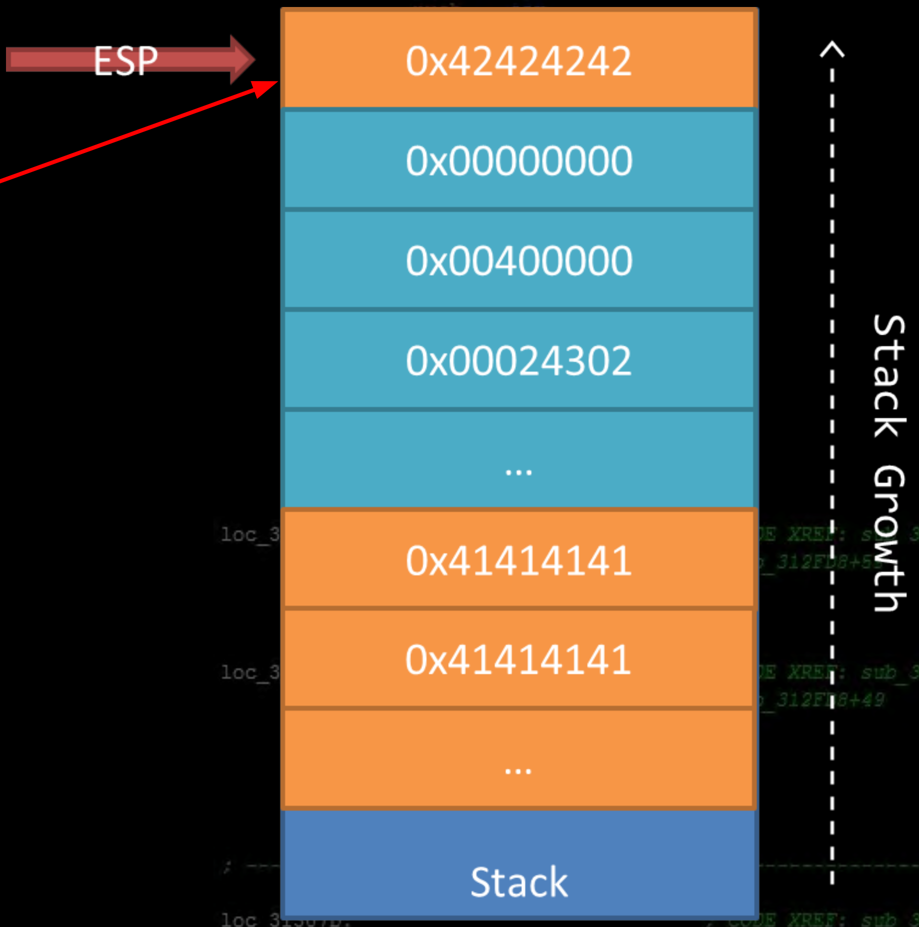
You have one gadget before you drop into arbitrary data on the stack



Stack Pivoting

You control the orange

You have one gadget before you drop into arbitrary data on the stack

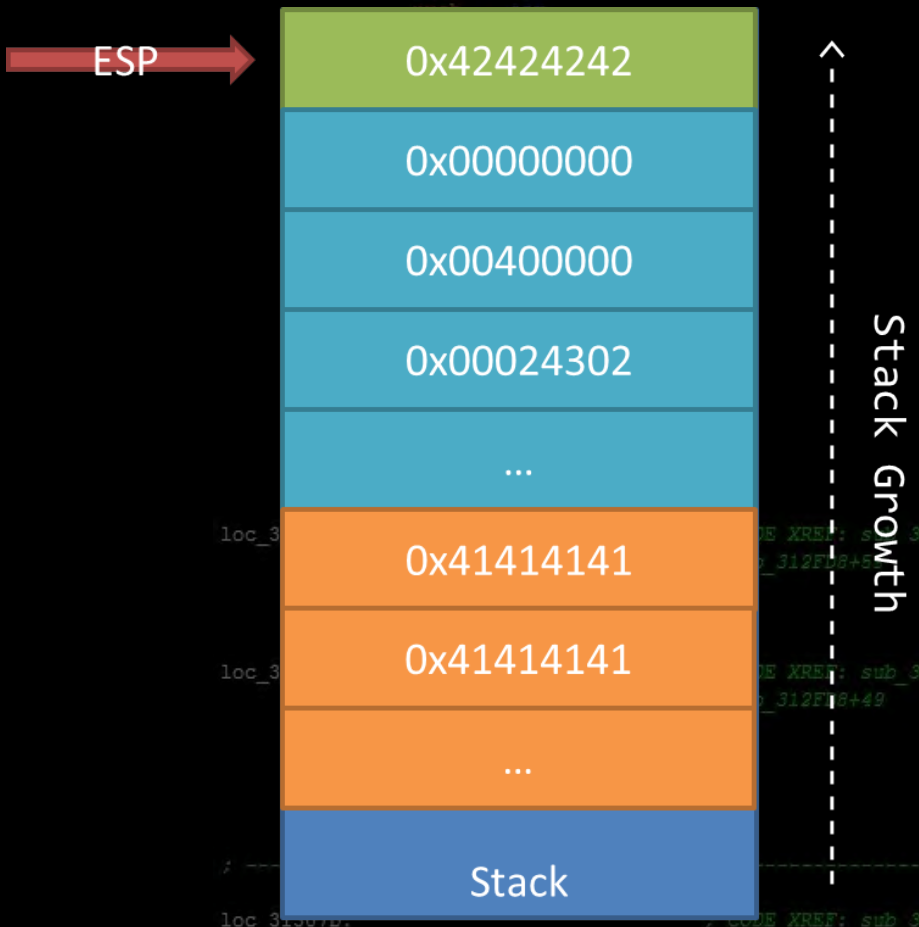


Stack Pivoting

You control the orange

You have one gadget before you drop into arbitrary data on the stack

Use your one gadget to move ESP into a more favorable location (Stack Pivot)



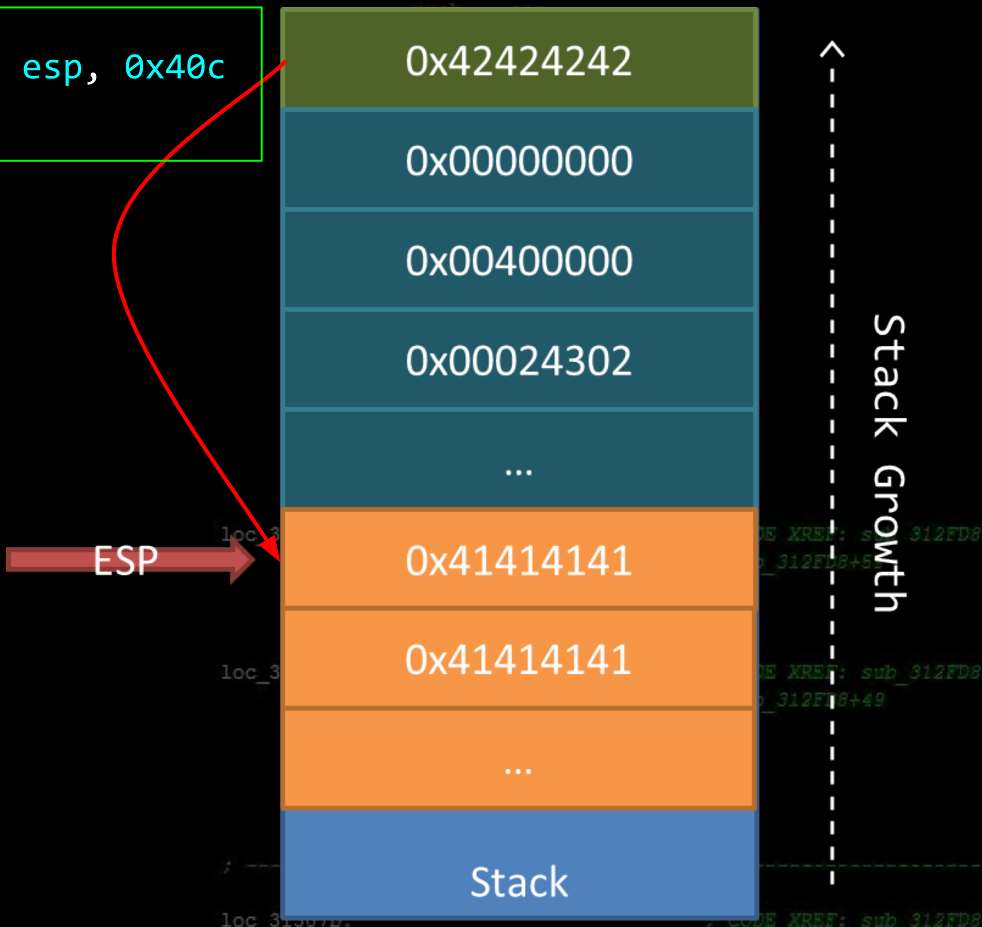
Stack Pivoting

You control the orange

You have one gadget before you drop into arbitrary data on the stack

Use your one gadget to move ESP into a more favorable location (Stack Pivot)

```
add   esp, 0x40c
ret
```



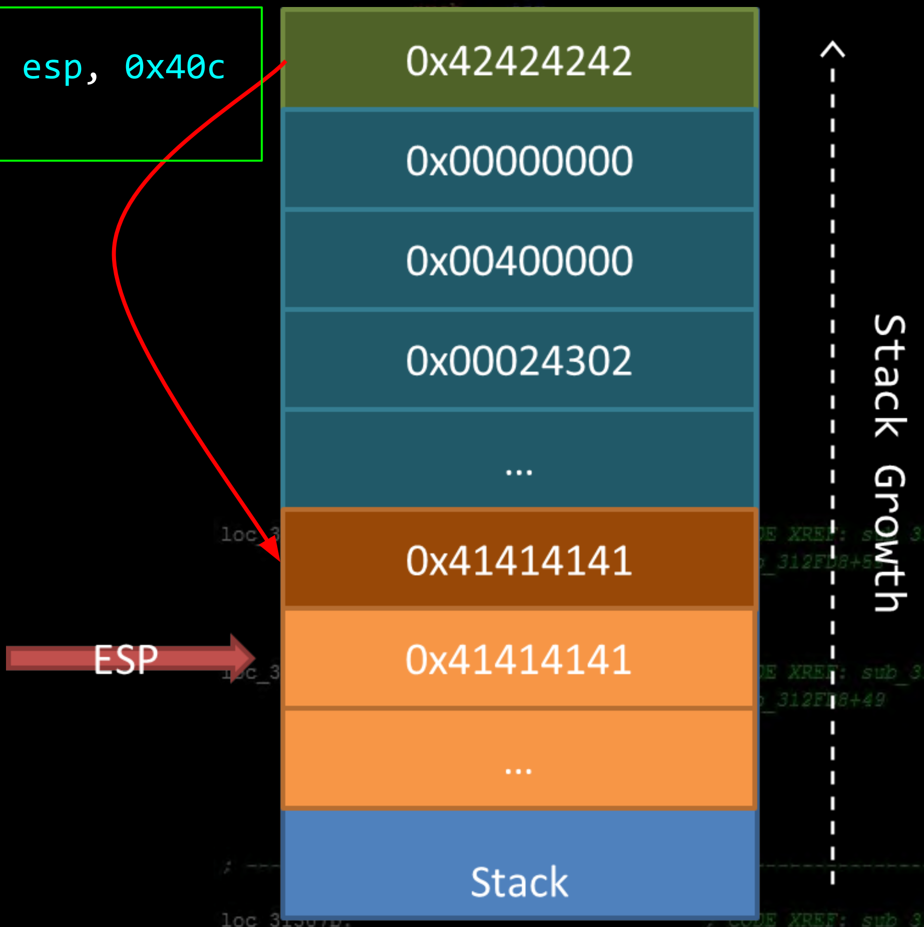
Stack Pivoting

```
add   esp, 0x40c
ret
```

You control the orange

You have one gadget before you drop into arbitrary data on the stack

Use your one gadget to move ESP into a more favorable location (Stack Pivot)



Stack Pivoting Tips

```
add     esp, 0xXXXX
ret
```

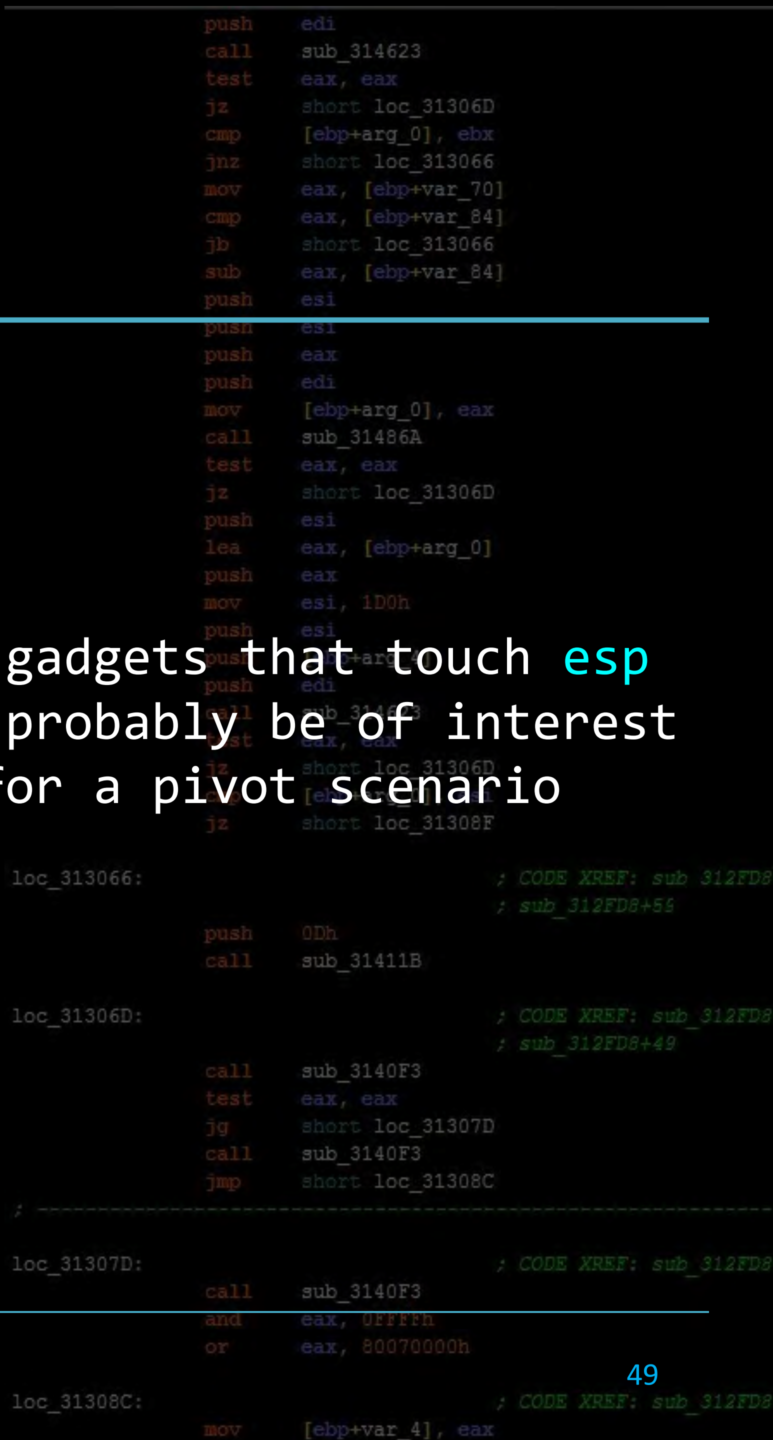
```
sub     esp, 0xXXXX
ret
```

```
ret 0xXXXX
```

```
leave     ; (mov esp, ebp)
ret
```

```
xchg eXX, esp
ret
```

any gadgets that touch esp
will probably be of interest
for a pivot scenario



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [illegible]
                push    edi
                call    [illegible]
                test    eax, eax
                jz      short loc_31306D
                [illegible]
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Stack Pivoting Tips

- You may not find an exact pivot, or you may need to pivot multiple times!

- You can always pad your ROP Chains with ROP NOPs which are simply gadgets that point to ret's

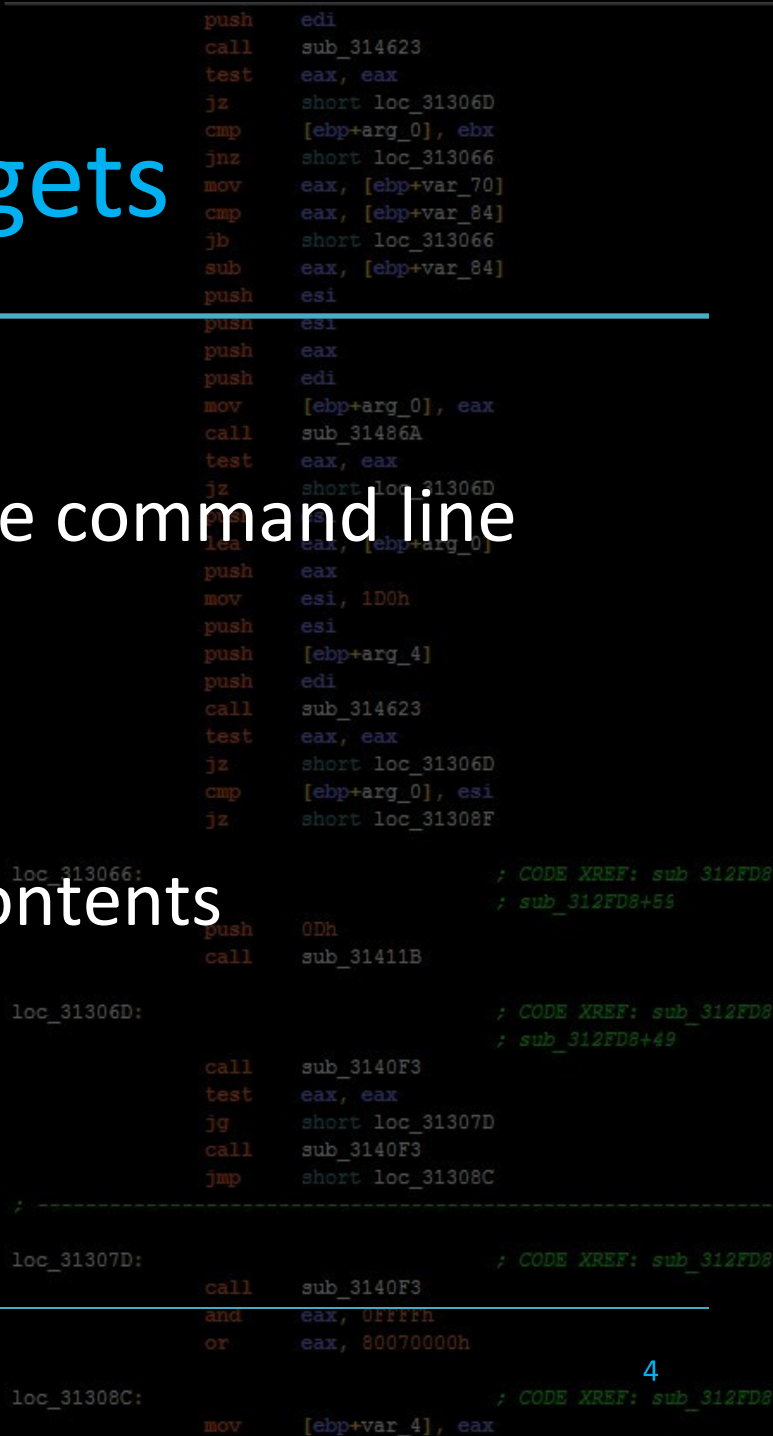
/levels/lecture/rop/rop_pivot

- Play around with Stack Pivoting on the warzone

ret2libc

- ‘ret2libc’ is a technique of ROP where you return to functions in standard libraries (libc), rather than using gadgets

- If you know the addresses of the functions you want to ROP through in libc (assuming libc exists), ret2libc is easier than making a ROP chain with gadgets

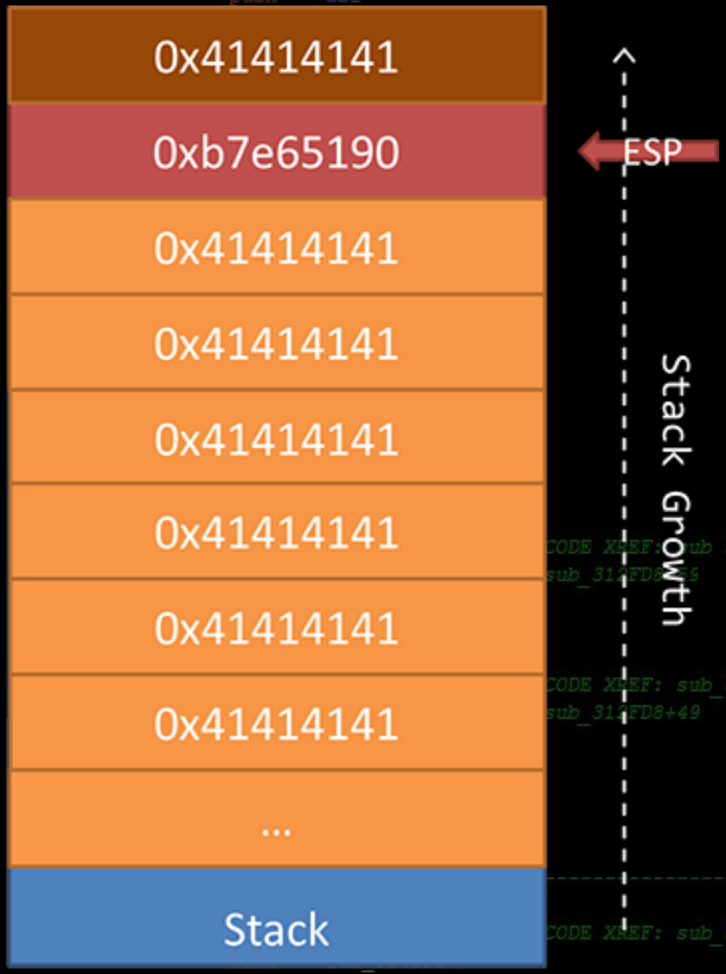


Common ret2libc Targets

- system()
  - Executes something on the command line
  - system("cat flag.txt");
- (f) open() / read() / write()
  - Open/Read/Write a file contents

ret2libc example

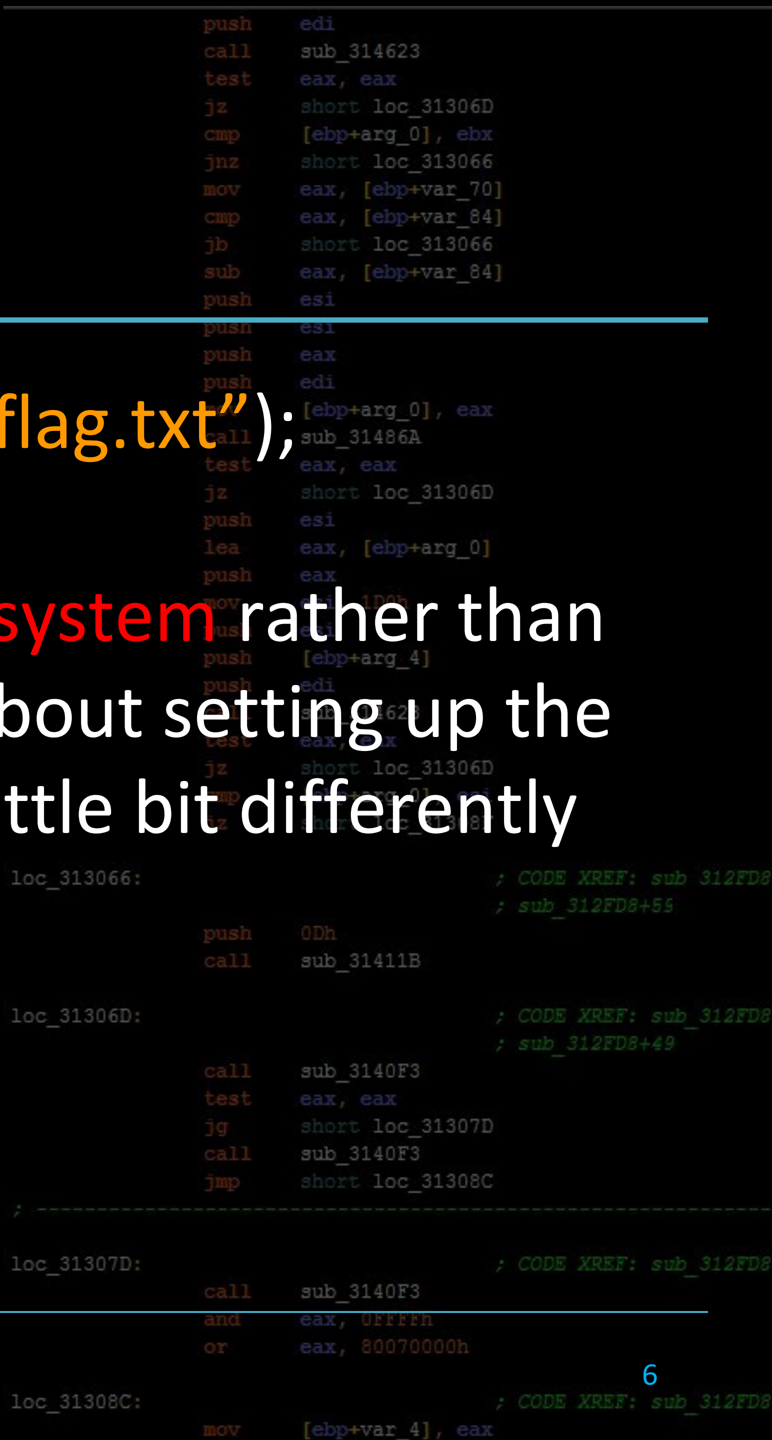
system() --->



```
0x08045430:  ret      EIP
.........................................
system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
   [esp+0x10]
...
```

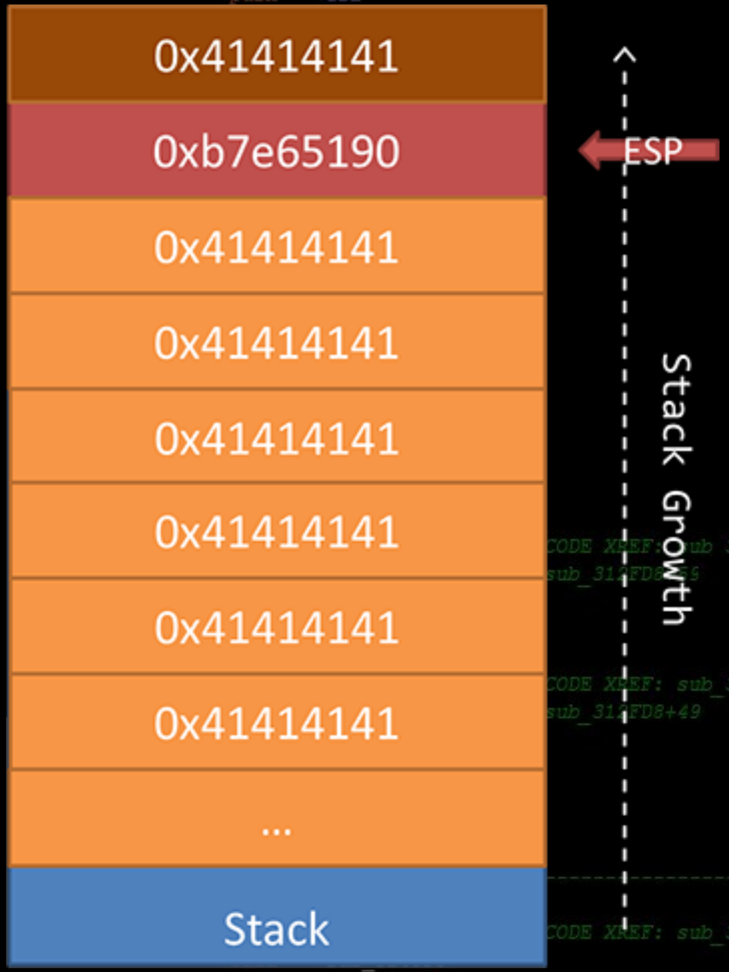
Returning to System

- We want to call system("cat flag.txt");
- Because we are ROPing into system rather than calling it, you have to think about setting up the stack (to pass arguments) a little bit differently



ret2libc example

system() --->

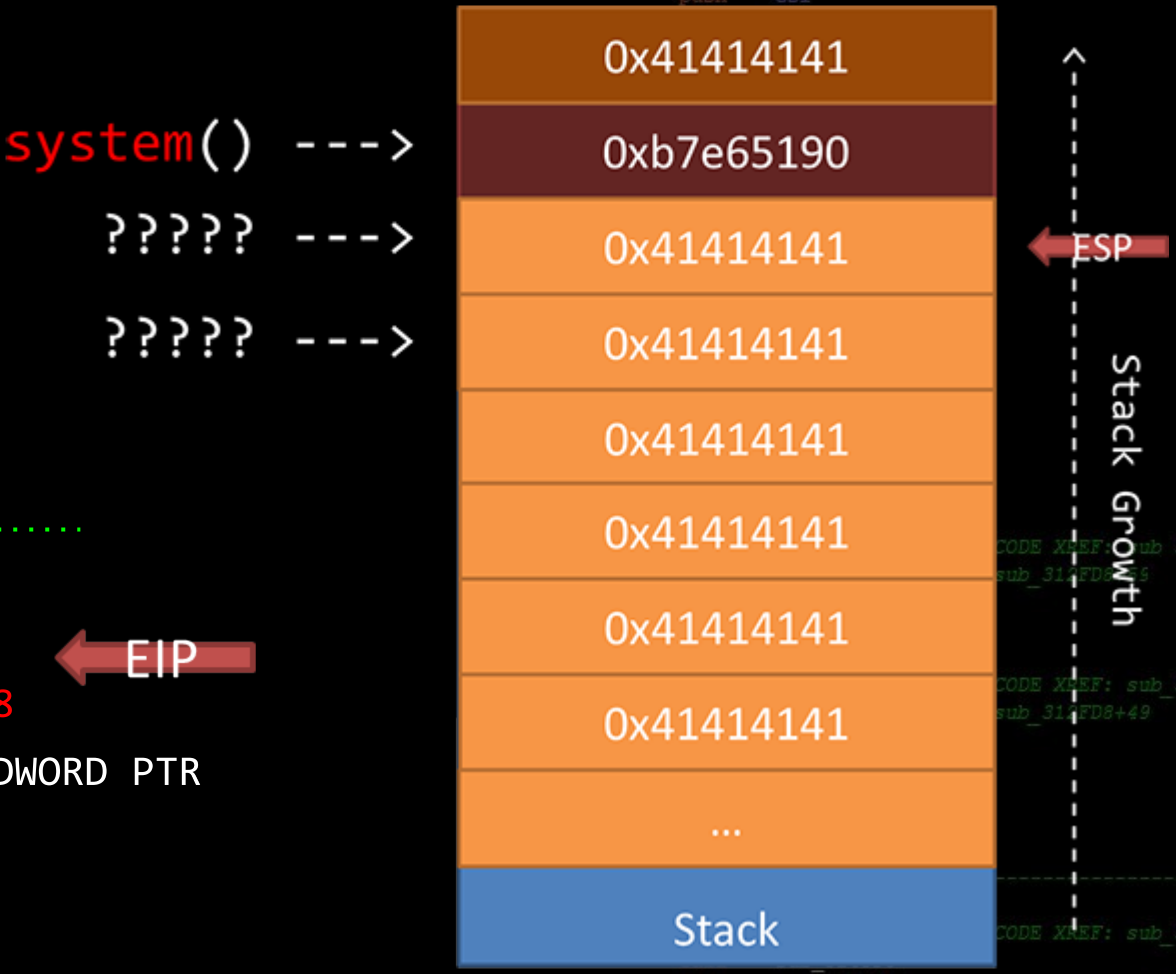


```
0x08045430:   ret
```

EIP

```
.....................................
system()
0xb7e65190:   push   ebx
0xb7e65191:   sub    esp, 8
0xb7e65194:   mov    eax, DWORD PTR
    [esp+0x10]
...
```

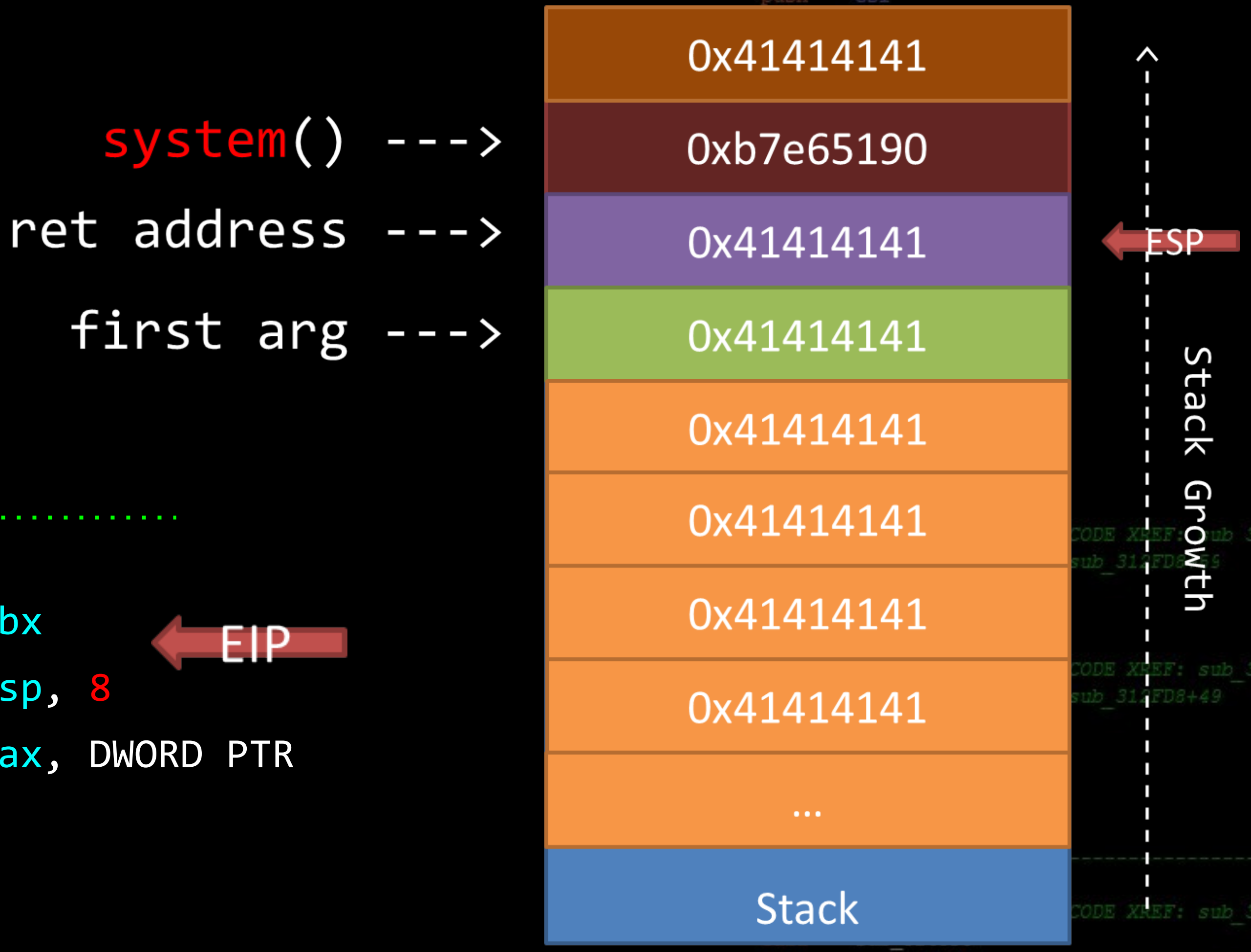
ret2libc example



```
0x08045430:  ret

.........................................
system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
   [esp+0x10]
...
```

EIP

ret2libc example

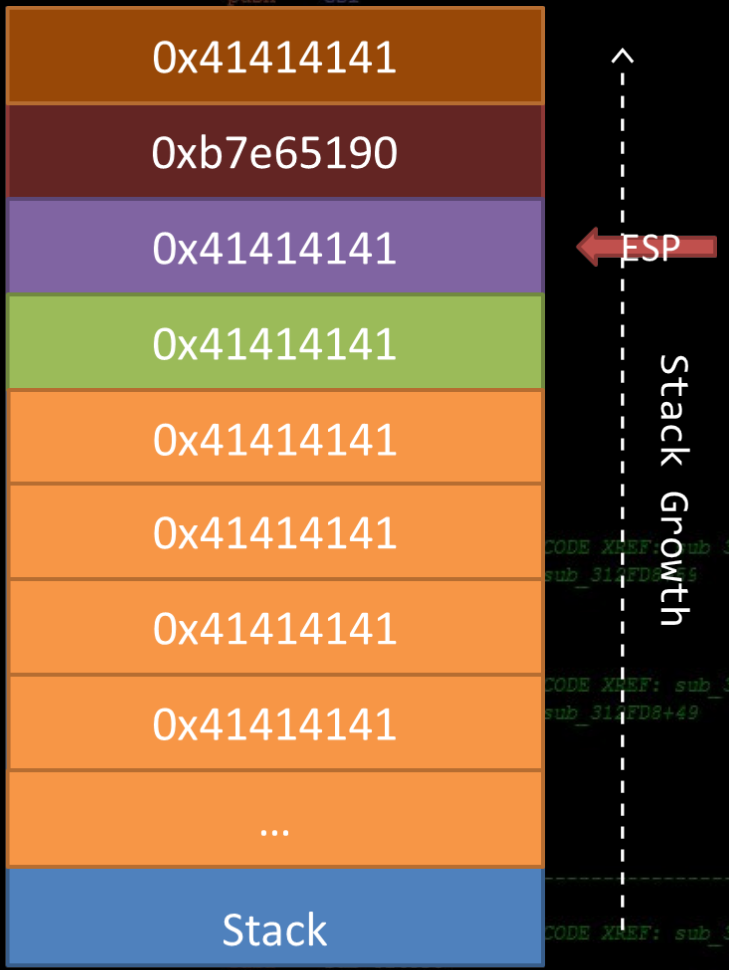


```
0x08045430:  ret

.........................................
system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
    [esp+0x10]
...
```

ret2libc example

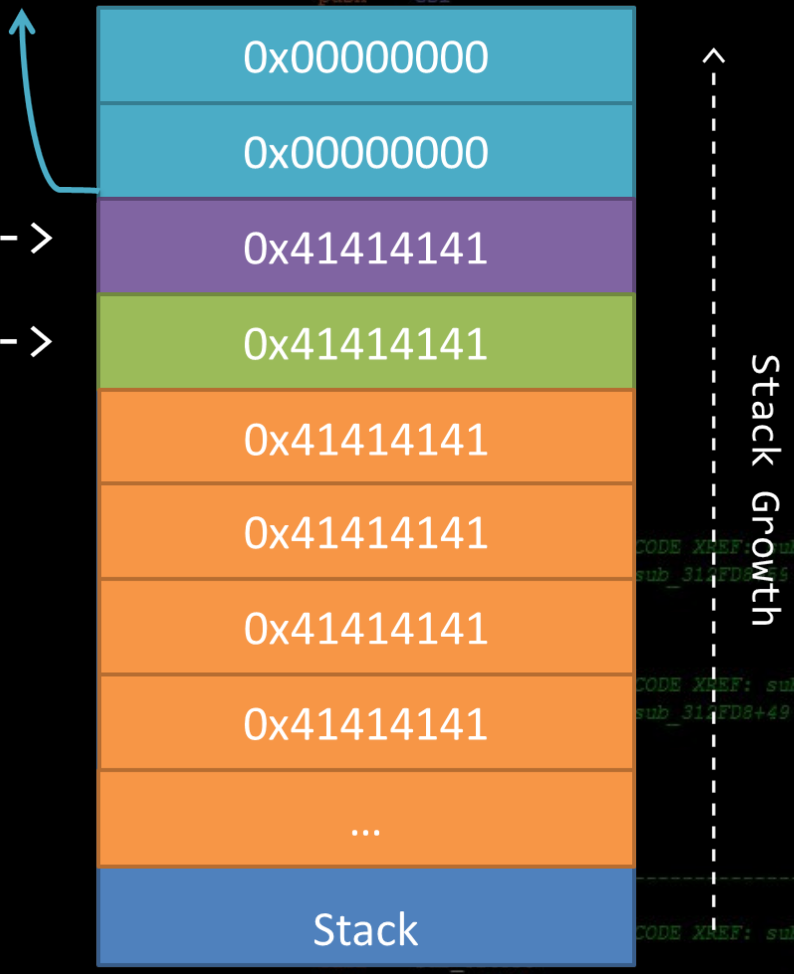
```
system()
0xb7e65190:   push    ebx
0xb7e65191:   sub     esp, 8
0xb7e65194:   mov     eax, DWORD PTR [esp+0x10]
...
```

ret2libc example



```
0x08045430:  ret

..........................................
system()
0xb7e65190:  push   ebx          <- EIP
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
   [esp+0x10]
...
```

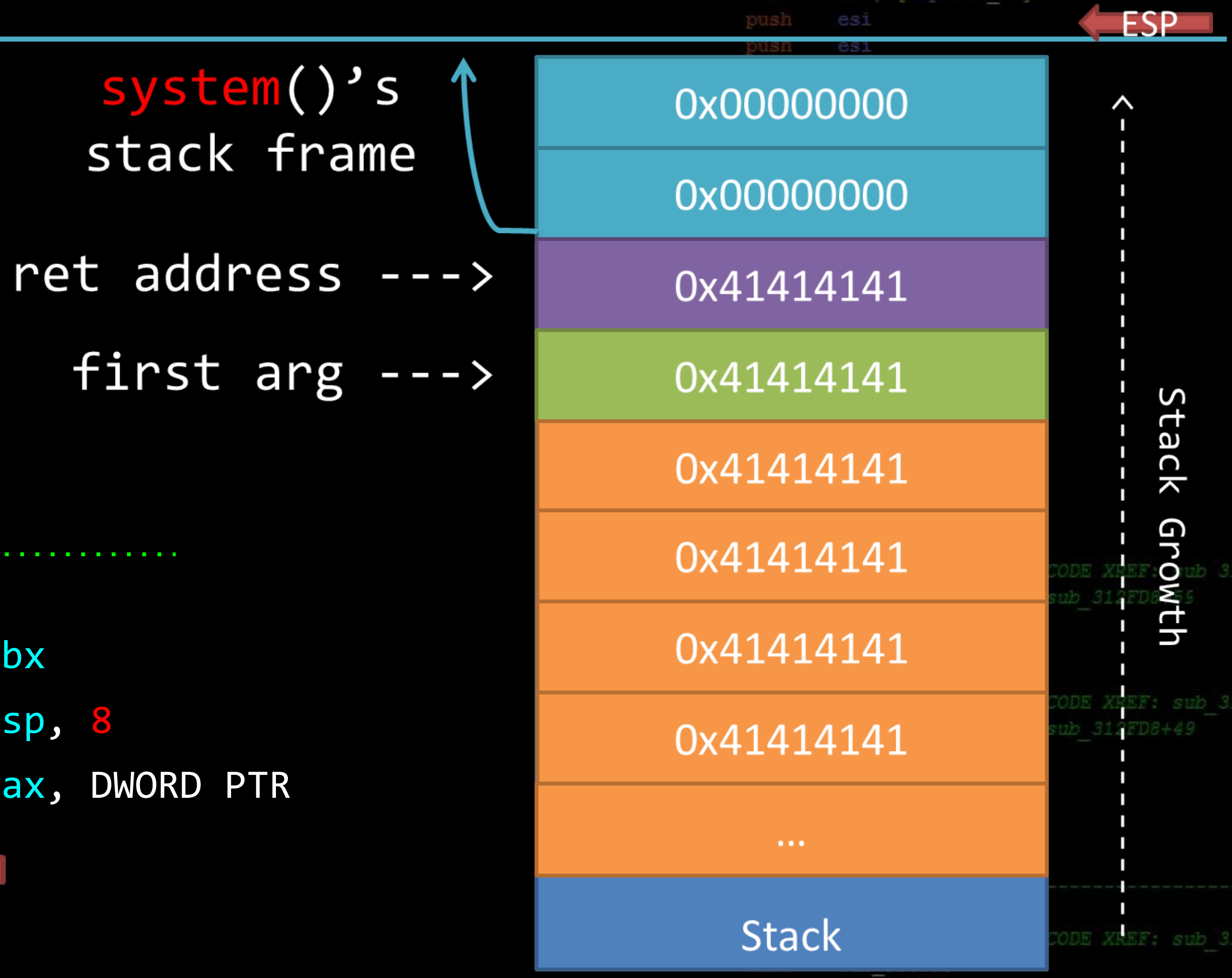
ret2libc example



```
0x08045430:  ret

..........................................

system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
   [esp+0x10]
...
```

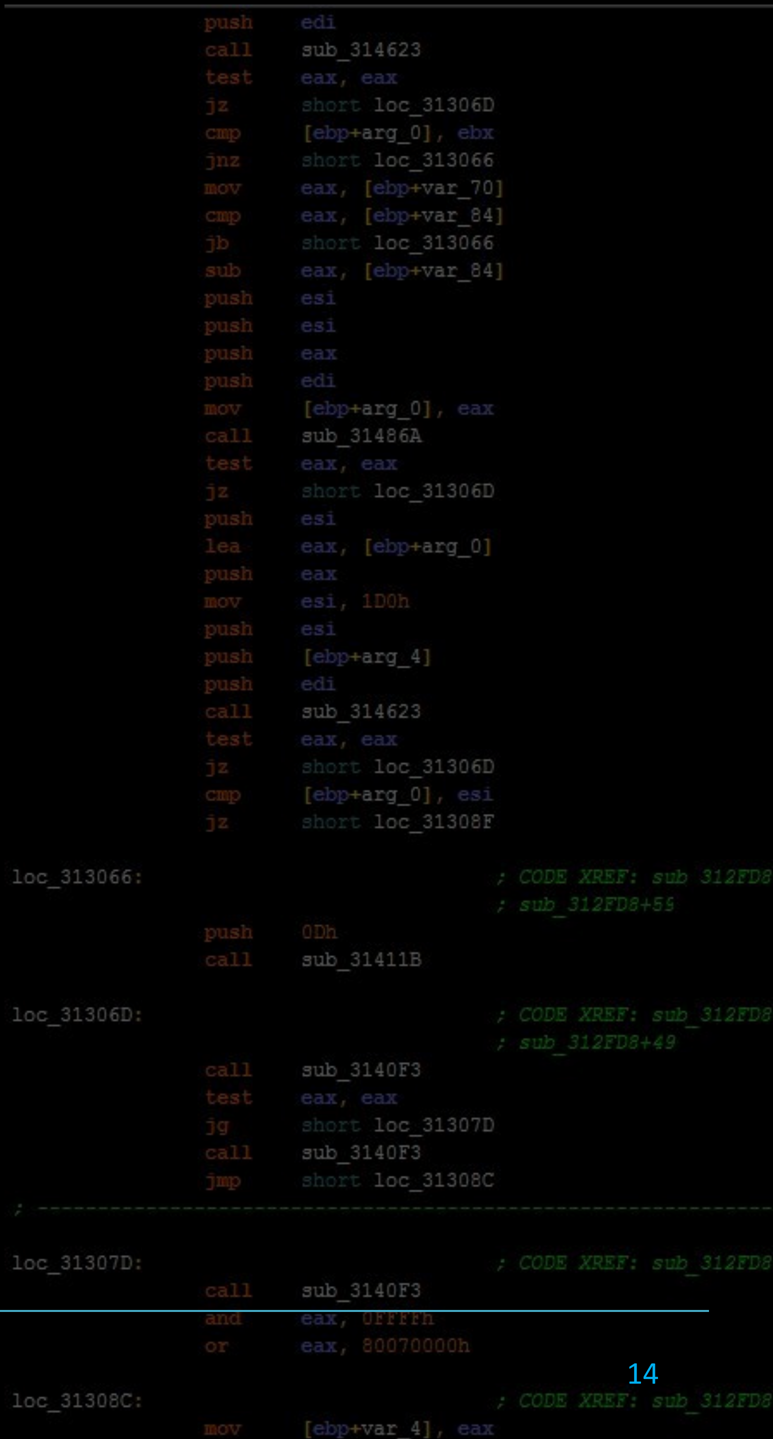
EIP

ret2libc example



```
0x08045430:   ret

..........................................

system()
0xb7e65190:   push   ebx
0xb7e65191:   sub    esp, 8
0xb7e65194:   mov    eax, DWORD PTR
   [esp+0x10]
...
```

REWIND

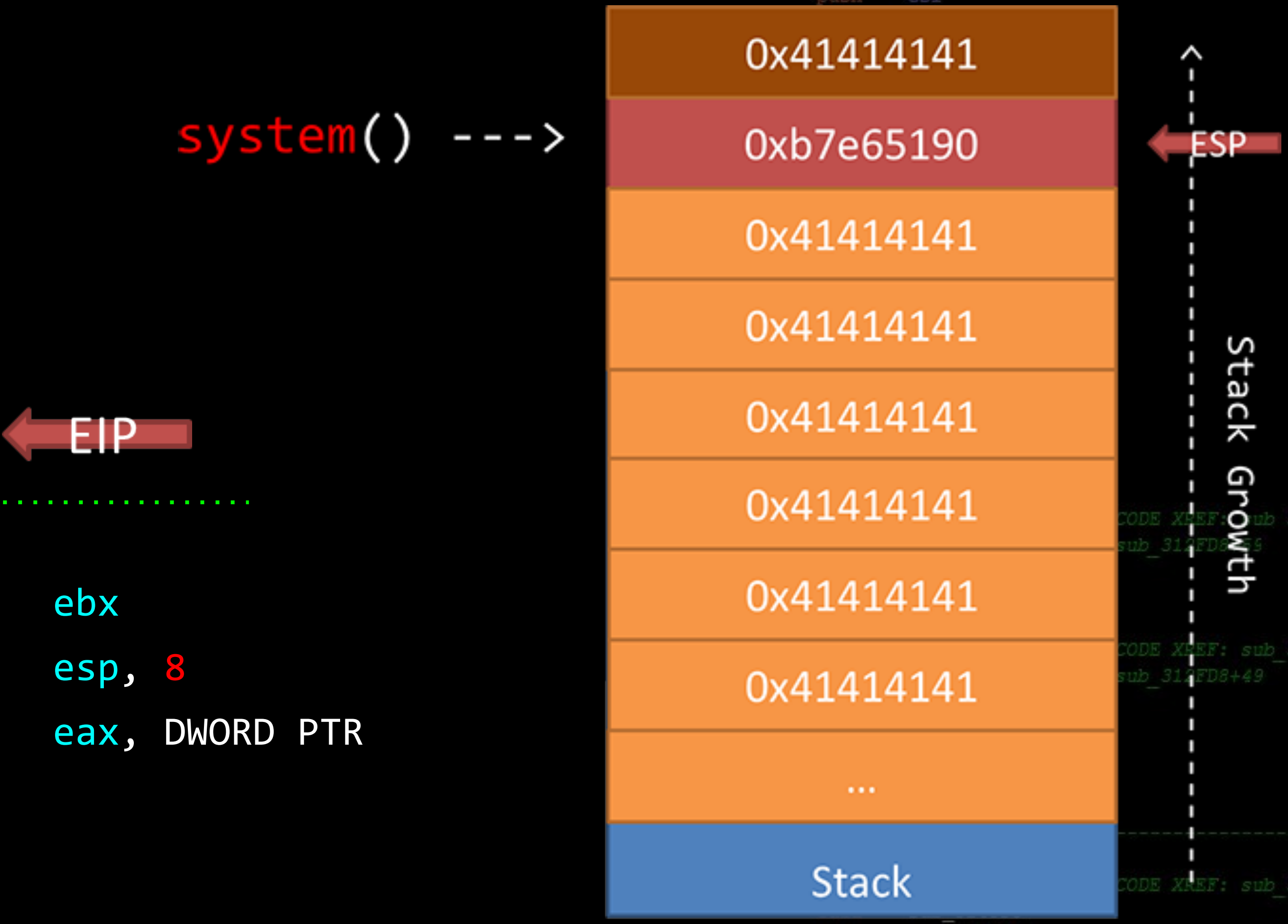


```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

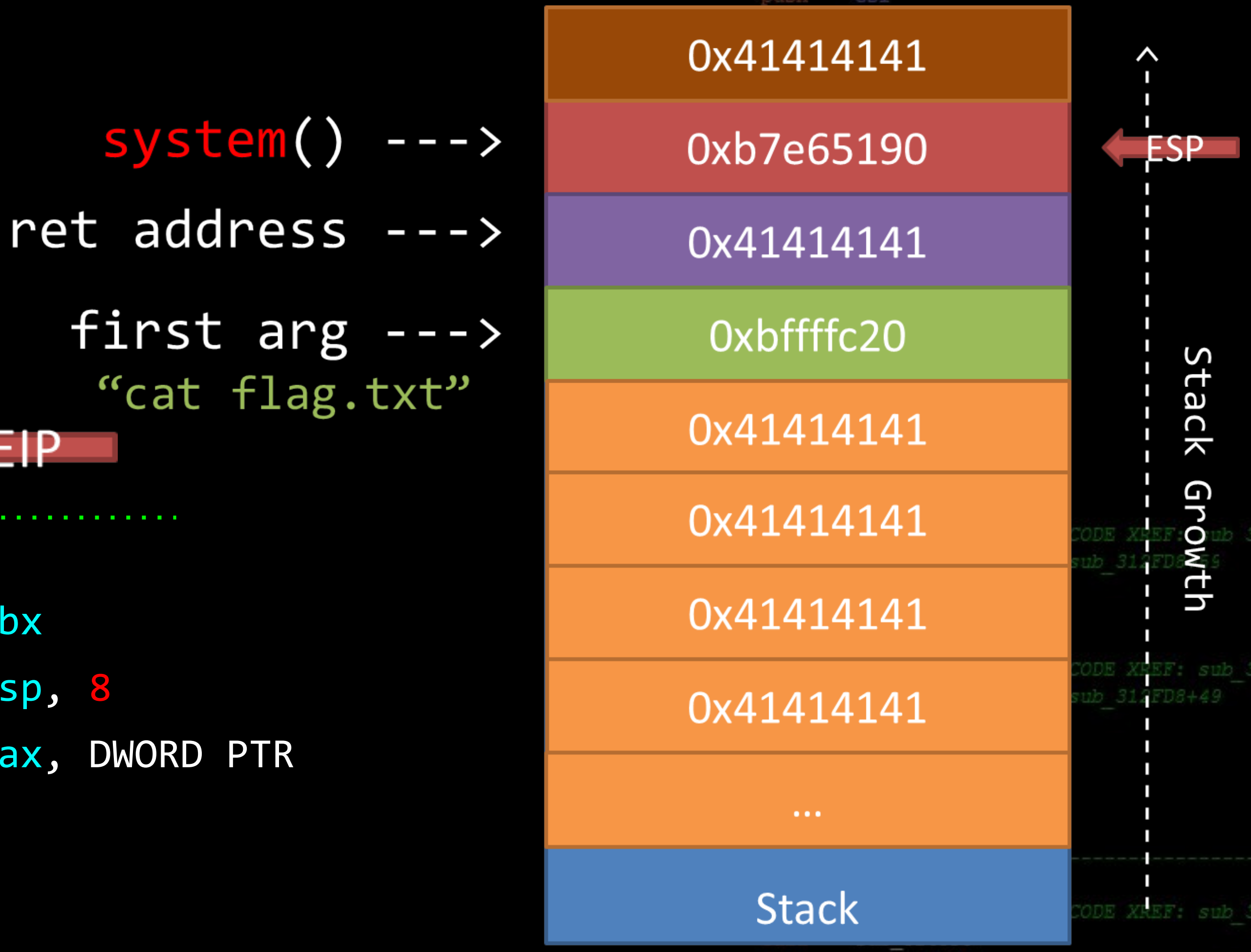
ret2libc example

```
0x08045430:   ret
```

```
system()
0xb7e65190:   push    ebx
0xb7e65191:   sub     esp, 8
0xb7e65194:   mov     eax, DWORD PTR
   [esp+0x10]
...
```



ret2libc example



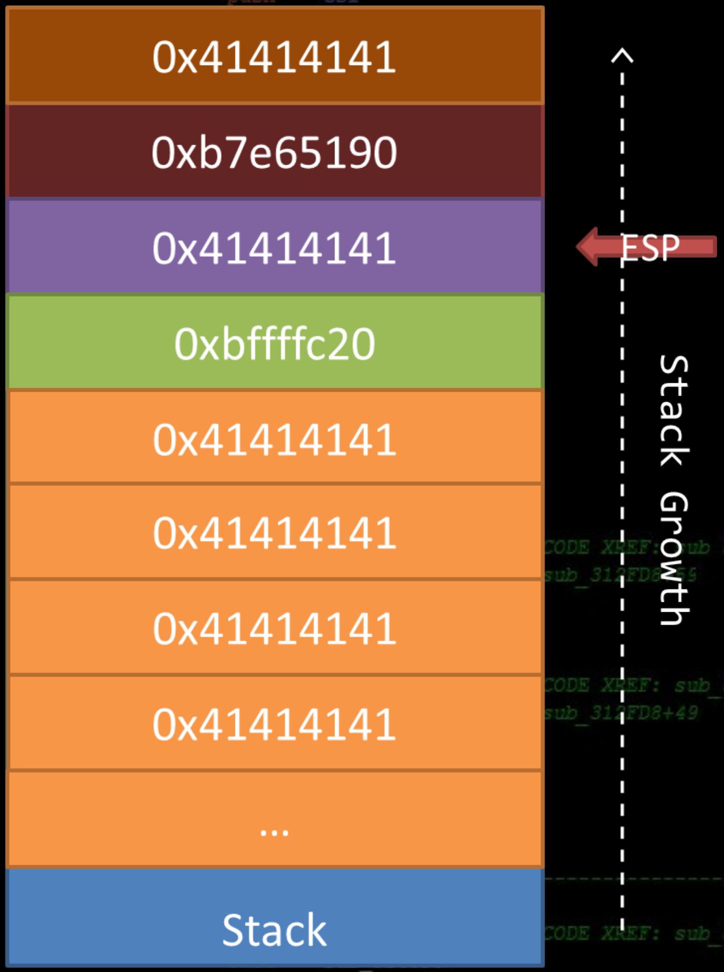
```
0x08045430:  ret        EIP

.........................................
system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
   [esp+0x10]
...
```

ret2libc example

```
system() --->
ret address --->
first arg --->
"cat flag.txt"
```

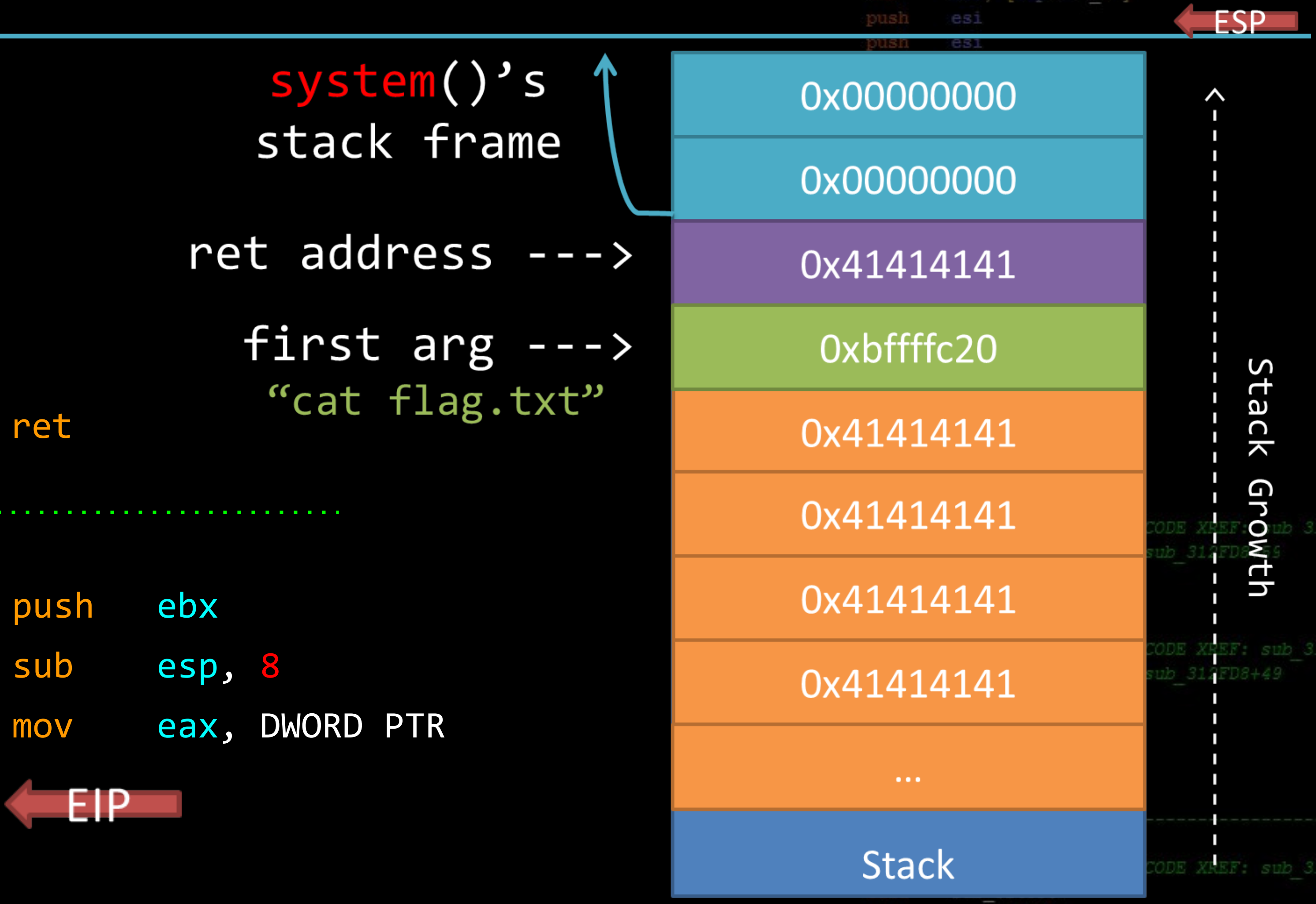
```
0x08045430:  ret

.....................................
system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
   [esp+0x10]
...
```

EIP

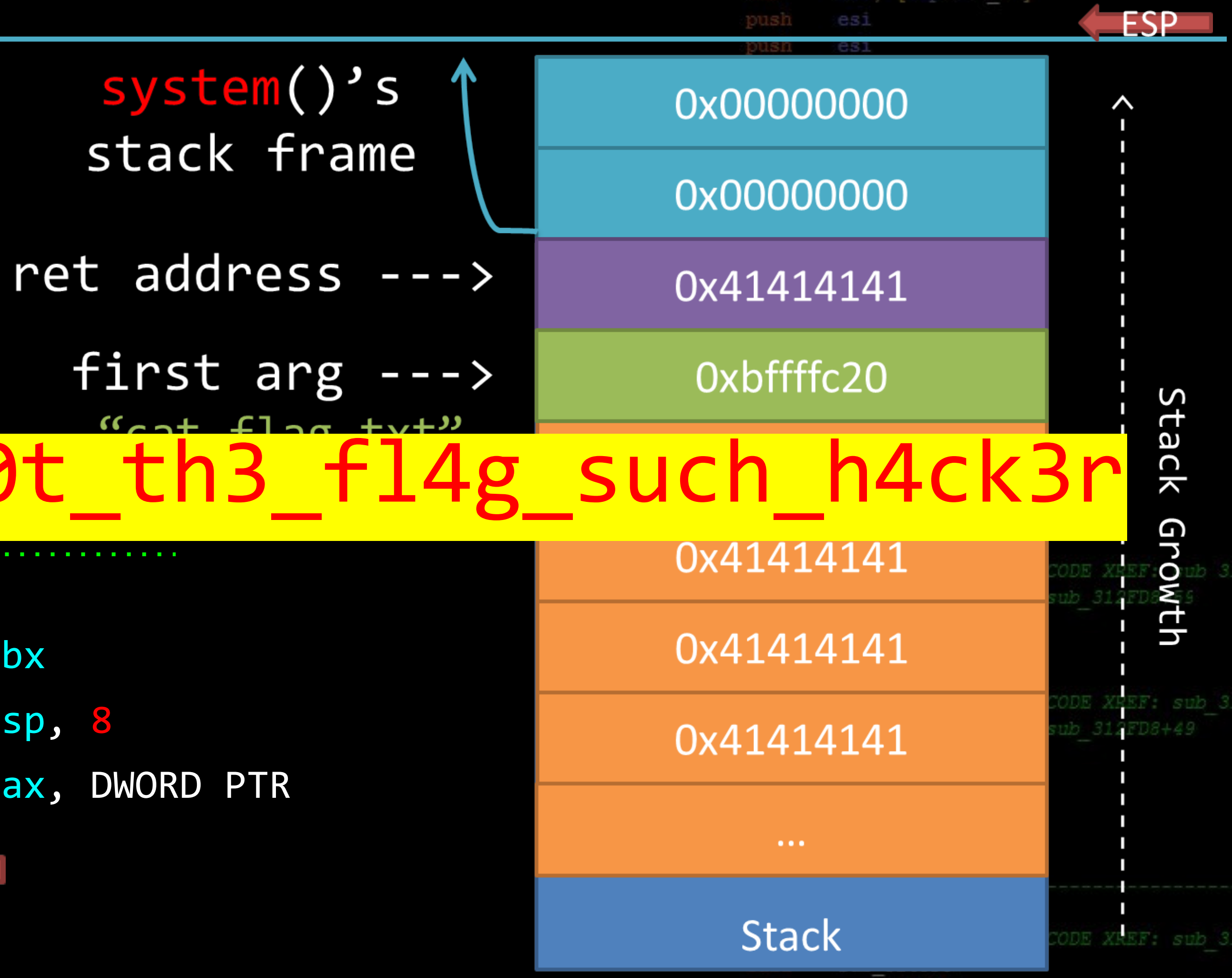


ret2libc example

```
0x08045430:   ret

.....................................
system()
0xb7e65190:   push   ebx
0xb7e65191:   sub    esp, 8
0xb7e65194:   mov    eax, DWORD PTR
   [esp+0x10]
...
```



ret2libc example

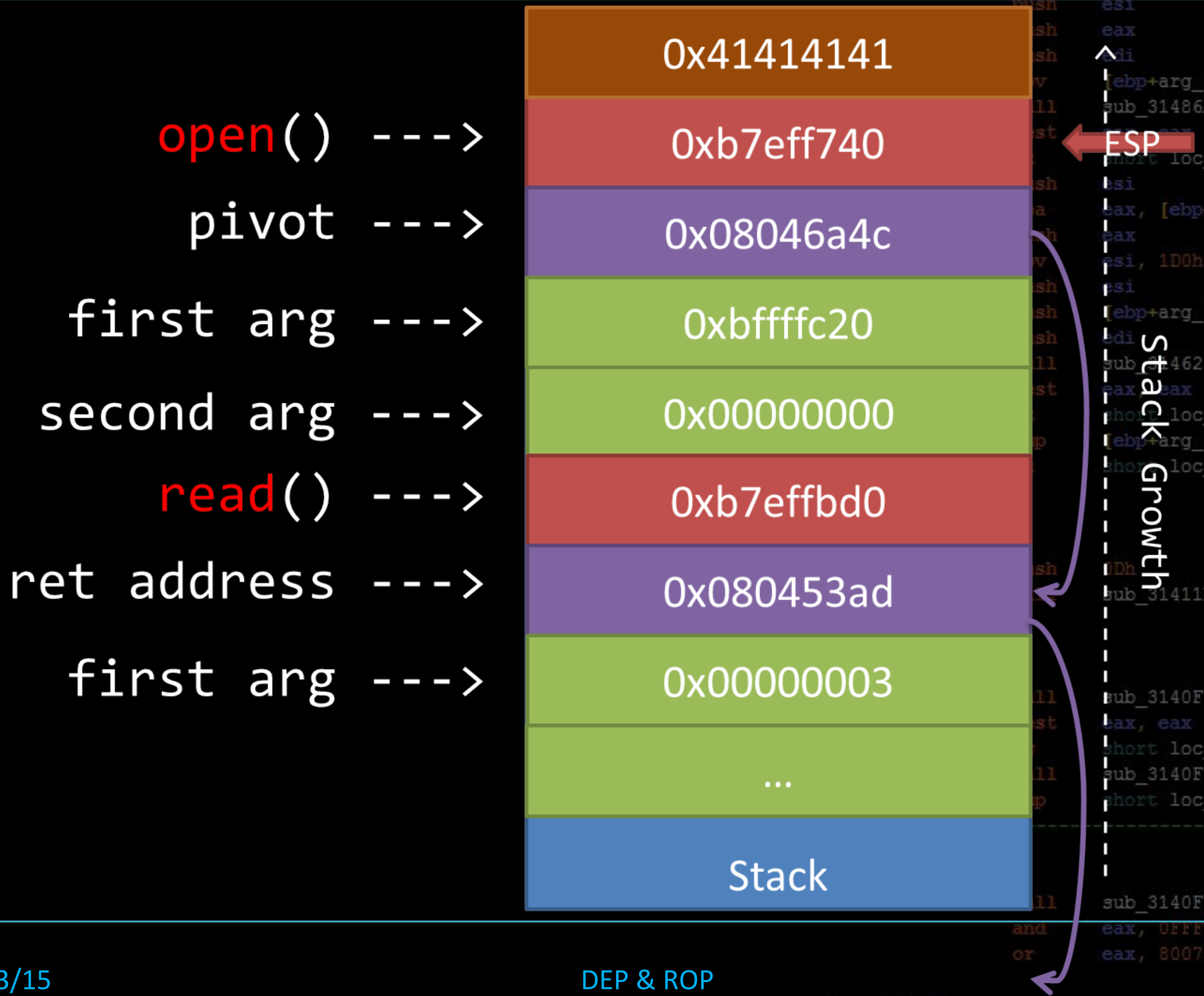


0x08

w0w_u_g0t_th3_fl4g_such_h4ck3r

```
system()
0xb7e65190:  push   ebx
0xb7e65191:  sub    esp, 8
0xb7e65194:  mov    eax, DWORD PTR
    [esp+0x10]   EIP
...
```

Chaining Calls



open() --->
pivot --->
first arg --->
second arg --->
read() --->
ret address --->
first arg --->
0x41414141
0xb7eff740
0x08046a4c
0xbffffc20
0x00000000
0xb7effbd0
0x080453ad
0x00000003
...
Stack
ESP
Stack Growth