

# 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
```

; CODE XREF: sub\_312FD8  
; sub\_312FD8+59

```
push    0Dh
call    sub_31411B
```

; 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

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

```
push    edi
call    .sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jnz   short loc_313066
or    eax, [ebp+var_70]
or    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
push    esi
call    .sub_314623
test    eax, eax
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, 0FFFh
or     eax, 80070000h

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

# Modern Exploit Mitigations

- There's 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

The terminal shows the output of the `checksec` command on a file named `rop\_exit`. The output indicates that the file has RELRO protection, a stack canary, NX enabled, and neither PIE nor RPATH/RUNPATH are present. A red arrow points to the 'NX enabled' status.

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	FILE
Partial RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	./rop_exit

Below the terminal, several snippets of assembly code are shown, likely from the `rop\_exit` program, illustrating various mitigation mechanisms like stack protection and memory alignment checks.

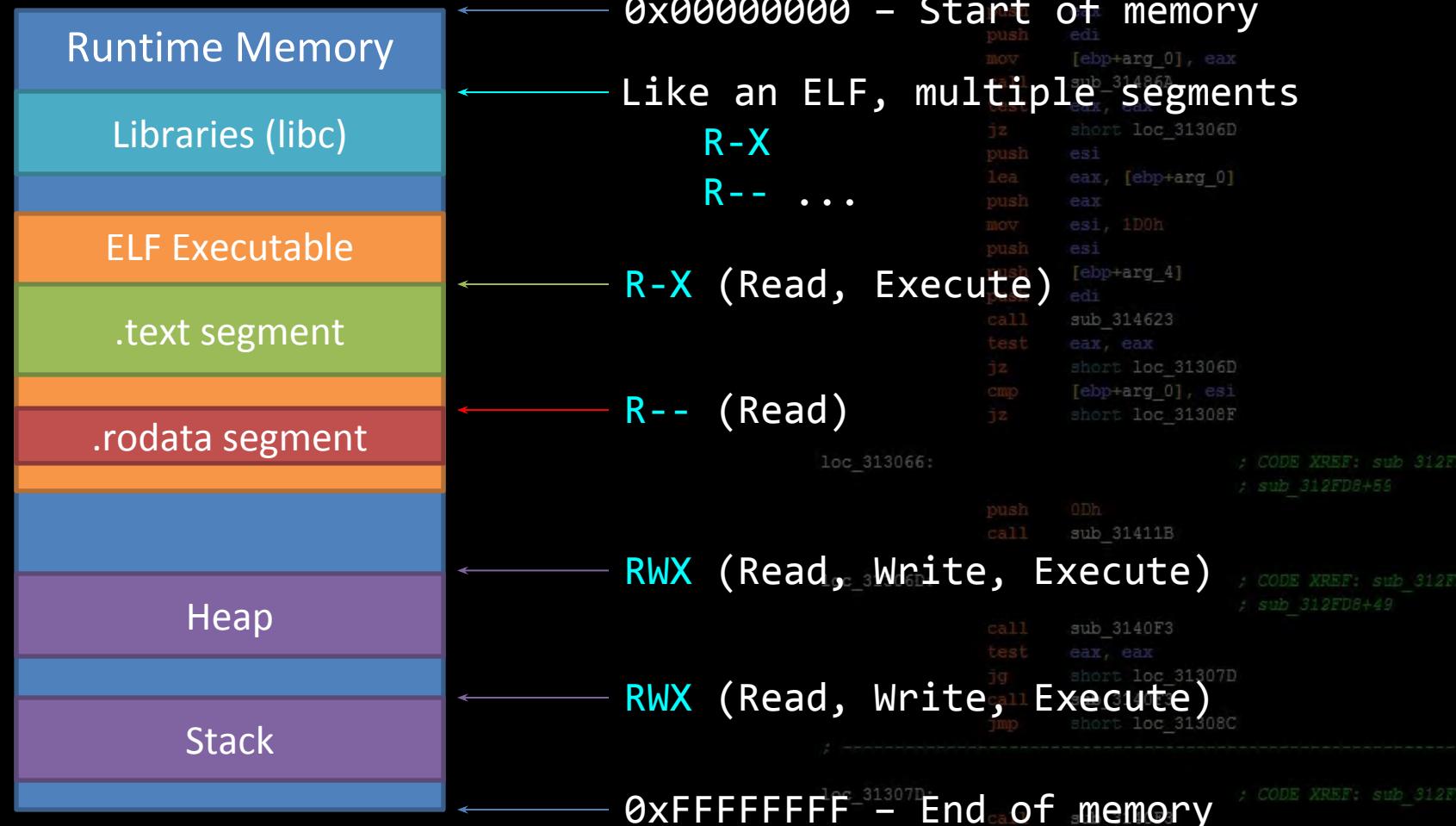
# Course Terminology

- **Data Execution Prevention**

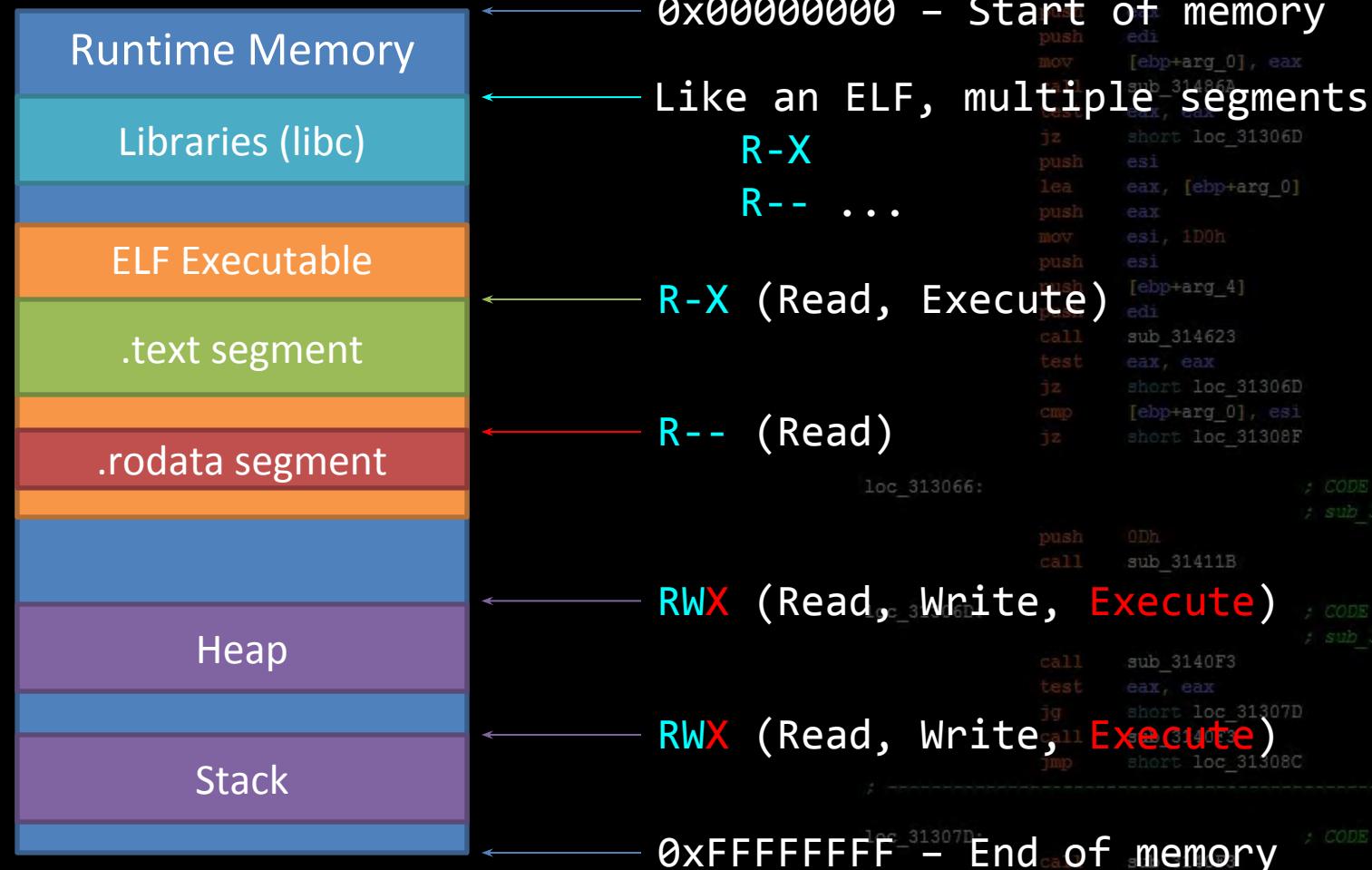
- An exploit mitigation technique used to ensure that only code segments are ever marked as executable
- Meant to mitigate code injection / **shellcode** payloads
- Also known as **DEP, NX, XN, XD, W^X**

```
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
pushn   esi
push    eax
push    edi
mov    [ebp+arg_0], eax
call    .sub_31486A
test    eax, eax
jz     short loc_31306D
lea    eax, [ebp+arg_0]
push    esi
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, 0FFFh
or     eax, 80070000h
loc_31308C:          ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

# Runtime Process Without DEP



# Runtime Process Without DEP



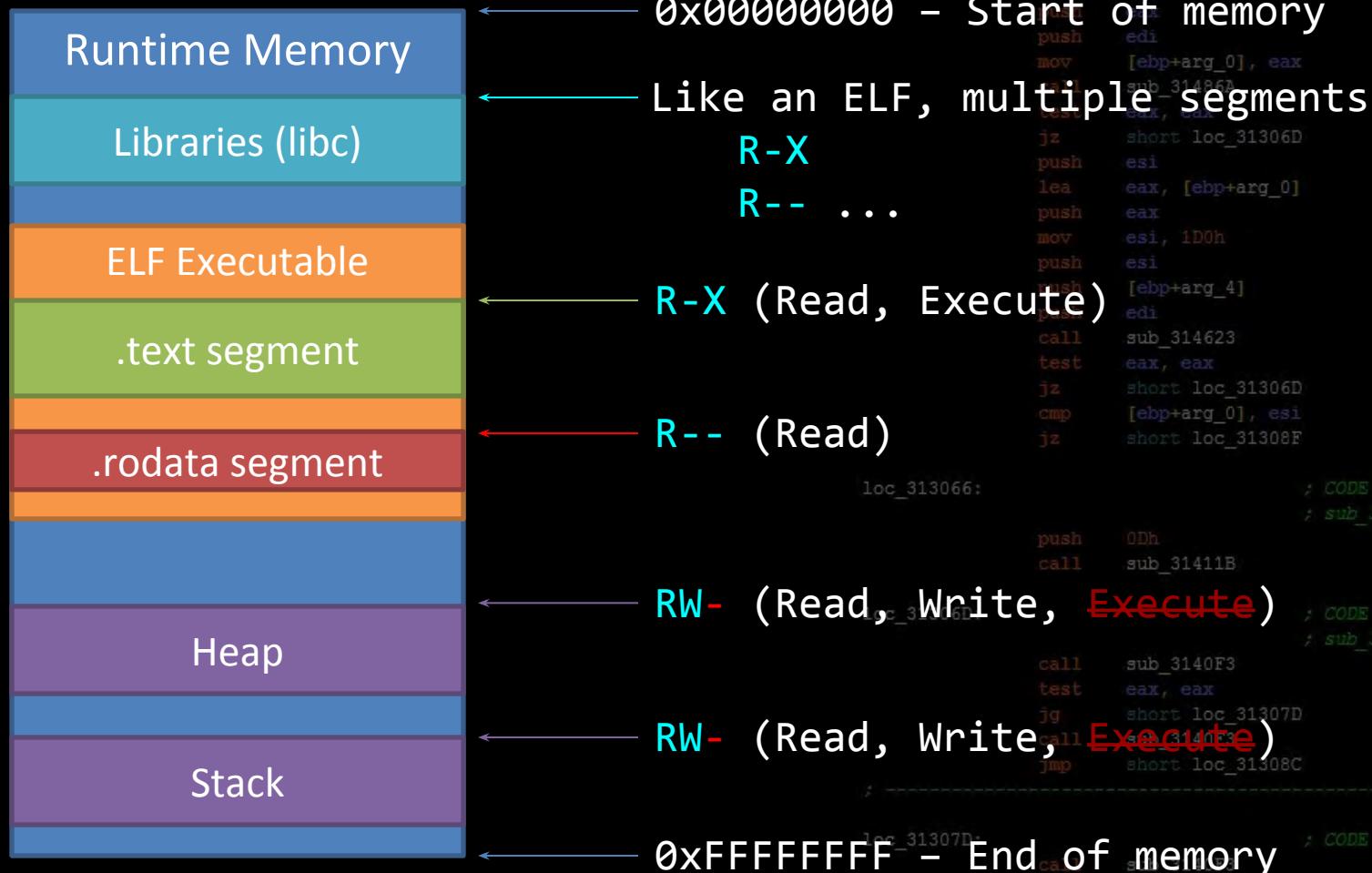
```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jz     short loc_313066
mov    eax, [ebp+var_70]
clt    [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    edi
mov    [ebp+arg_0], eax
call    sub_314861
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
```

```
; CODE XREF: sub_312FD8
; sub_312FD8+59
push    0Dh
call    sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_314103
jmp    short loc_31308C
```

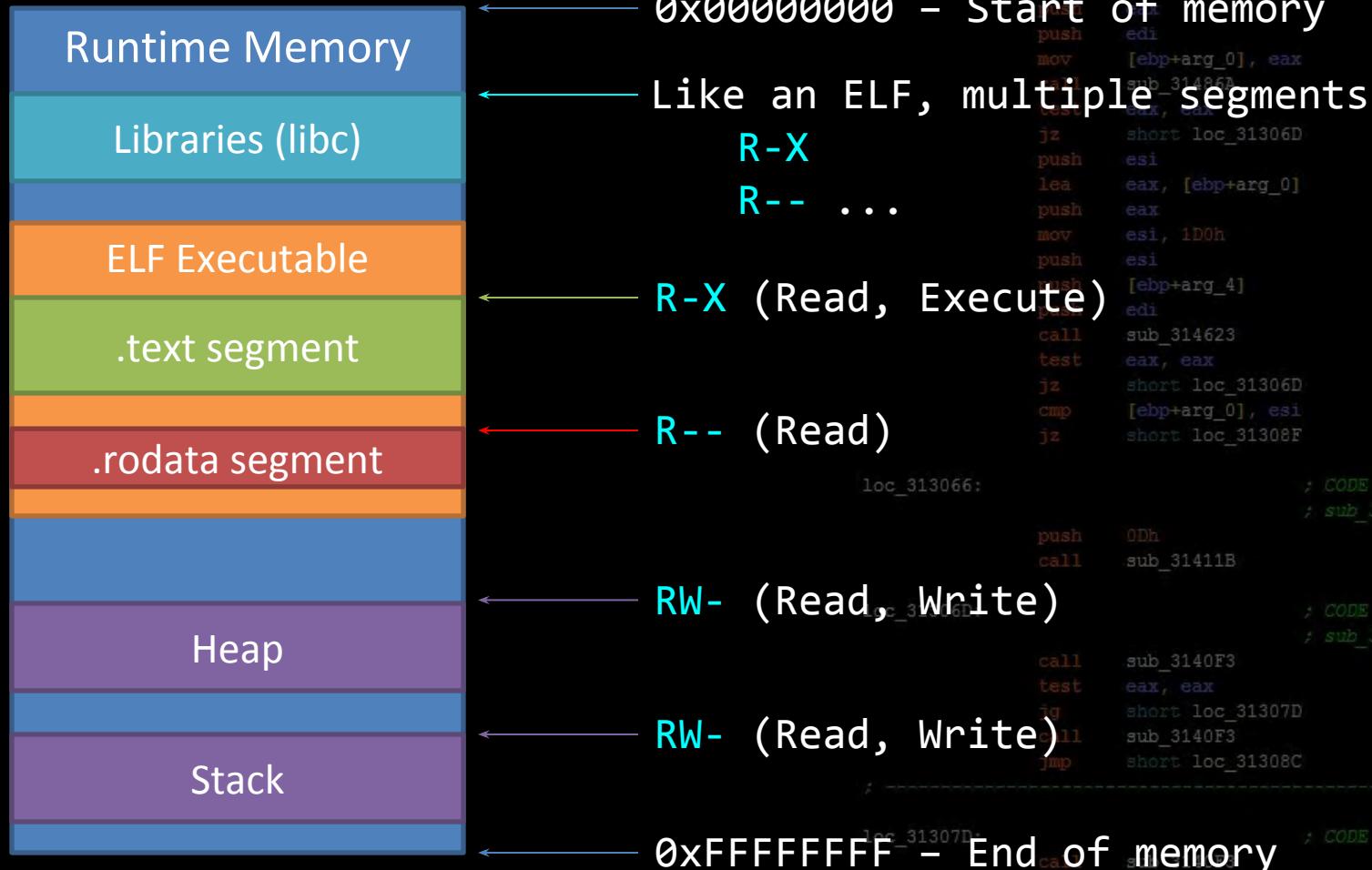
```
; CODE XREF: sub_312FD8
; sub_312FD8+49
and    eax, 0FFFFh
or     eax, 80070000h
```

```
; CODE XREF: sub_312FD8
; sub_312FD8+49
mov    [ebp+var_4], eax
```

# Runtime Process Without DEP



# Runtime Process With DEP



# 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

```
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
call    sub_314623
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
```

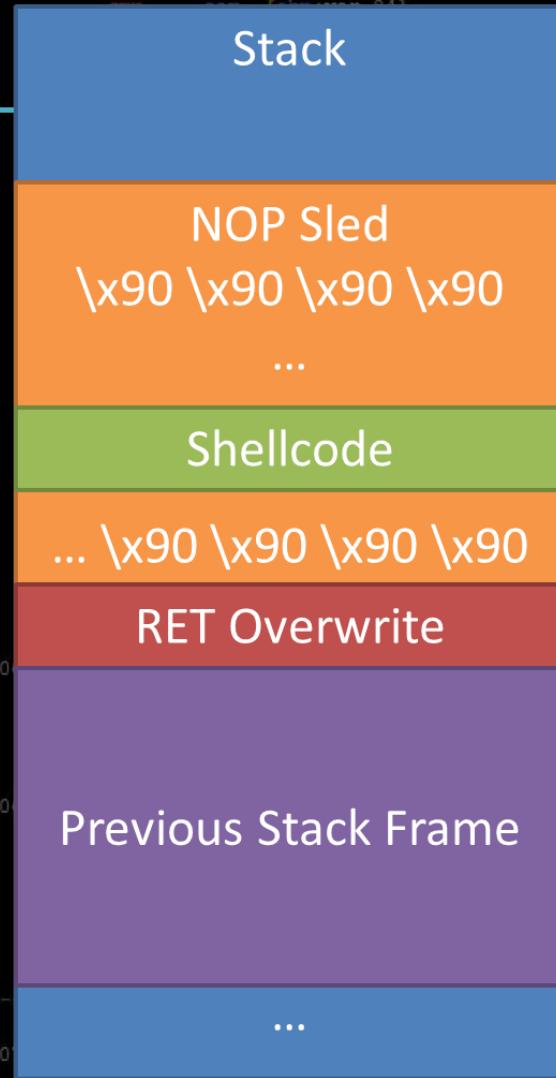
# DEP in Action

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

0xbffffd000 ---->  
(lower addrs)

0xc0000000 ---->  
(higher addrs)

DEP & ROP



# 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?

0xbffffd000 ---->  
(lower addrs)

DEP & ROP

Stack

NOP Sled

\x90 \x90 \x90 \x90

...

Shellcode

... \x90 \x90 \x90 \x90

RET Overwrite

Previous Stack Frame

0xc0000000 ---->  
(higher addrs)

loc\_31308C:

; CODE XREF: sub\_312FD8

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]

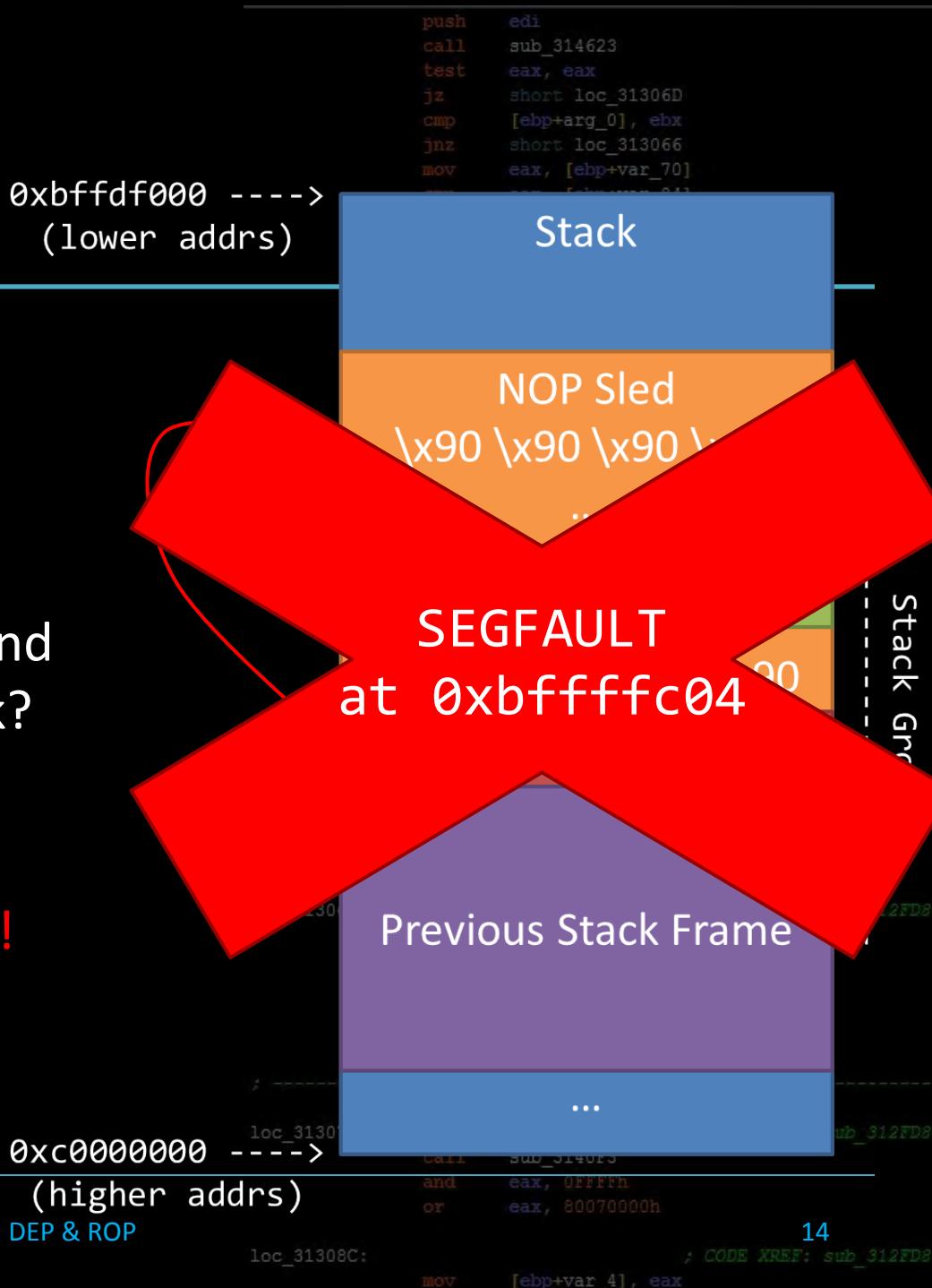
and eax, 0FFFh  
or eax, 80070000h

mov [ebp+var\_4], eax

# 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
; ----- ; CODE XREF: sub_312FD8
loc_31307D:                                ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
; ----- ; CODE XREF: sub_312FD8
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
; ----- ; CODE XREF: sub_312FD8
```

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

```
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, 0FFFh
or     eax, 80070000h
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

# 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



MBE - 03/10/15

DEP & ROP

21

# 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**

```
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
; --- loc_313066: ; CODE XREF: sub_312FD8+59
test    eax, eax
jz     short loc_31306D
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
lea    eax, [ebp+arg_0], esi
; --- loc_31308F: ; CODE XREF: sub_312FD8+59
push    edi
call    .sub_31411B
; --- loc_31306D: ; CODE XREF: 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+49
call    .sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
; --- loc_31308C: ; CODE XREF: sub_312FD8+49
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
```

# 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 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

```
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, 1D0
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
; CODE XREF: sub_312FD8+59 ; sub_312FD8+59
push    0Dh
call    .sub_31411B
; CODE XREF: sub_312FD8+49 ; 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+49
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
; -----
loc_31308C:                                ; CODE XREF: sub_312FD8+49
mov    [ebp+var_4], eax
```

# 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+59
    push 0Dh
    call sub_31411B

loc_31306D: ; CODE XREF: 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$
```

MBE - 03/10/15

DEP & ROF

28

# 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
```

loc\_313066:  
call sub\_314623  
test eax, eax  
jz short loc\_31306D  
cmp [ebp+arg\_0], esi  
jz short loc\_31308F

loc\_31306D:  
push [ebp+arg\_4]  
push edi  
call sub\_31462A  
test eax, eax  
jz short loc\_31306D  
push esi  
lea eax, [ebp+var\_70]  
push eax  
push edi  
call sub\_31462A  
test eax, eax  
jz short loc\_31306D  
push esi  
lea eax, [ebp+var\_84]  
jb short loc\_313066  
sub eax, [ebp+var\_84]  
push esi  
push esi  
push eax  
push edi  
call sub\_31462A  
test eax, eax  
jz short loc\_31306D  
push esi  
lea eax, [ebp+arg\_0]  
push eax  
push edi  
call sub\_31462A  
test eax, eax  
jz short loc\_31306D  
push esi  
lea eax, [ebp+arg\_4]  
push edi  
call sub\_314623  
test eax, eax  
jz short loc\_31306D  
cmp [ebp+arg\_0], esi  
jz short loc\_31308F

; CODE XREF: sub\_312FD8+59 ; sub\_312FD8+59

loc\_31308C:  
call sub\_3140F3  
test eax, eax  
jg short loc\_31307D  
call sub\_3140F3  
jmp short loc\_31308C

; CODE XREF: sub\_312FD8+49 ; sub\_312FD8+49

call sub\_3140F3  
and eax, 0FFFFh  
or eax, 80070000h

; CODE XREF: sub\_312FD8+49 ; sub\_312FD8+49

loc\_31308C:  
mov [ebp+var\_4], eax

# Understanding ROP

exit(0) - ROP chain

xor eax, eax

ret

xor ebx, ebx

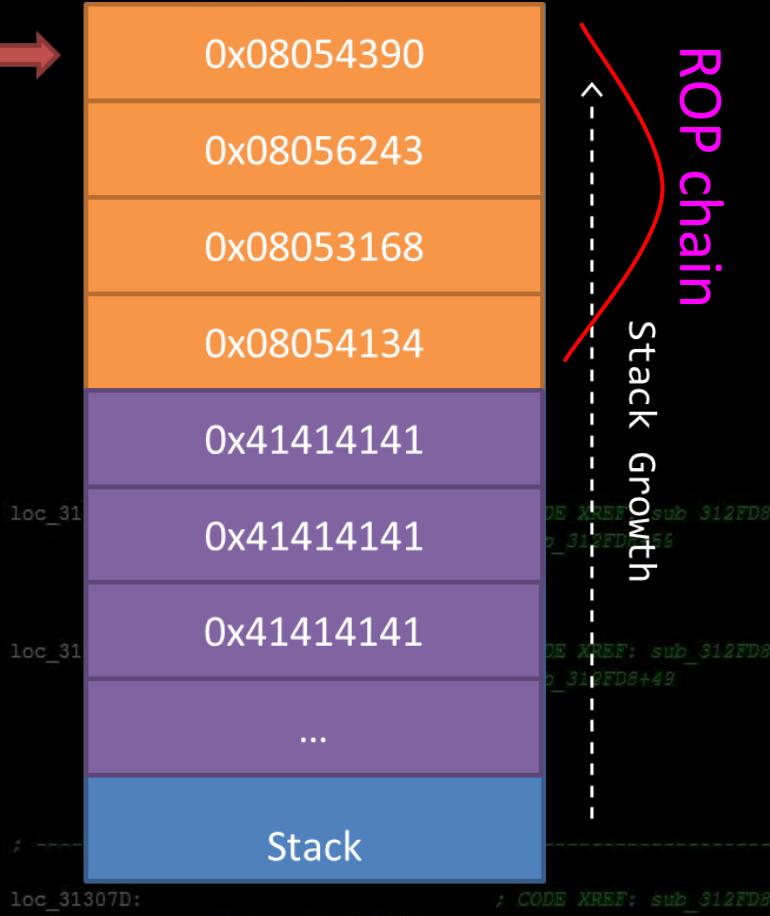
ret

inc eax

ret

int 0x80

ESP →



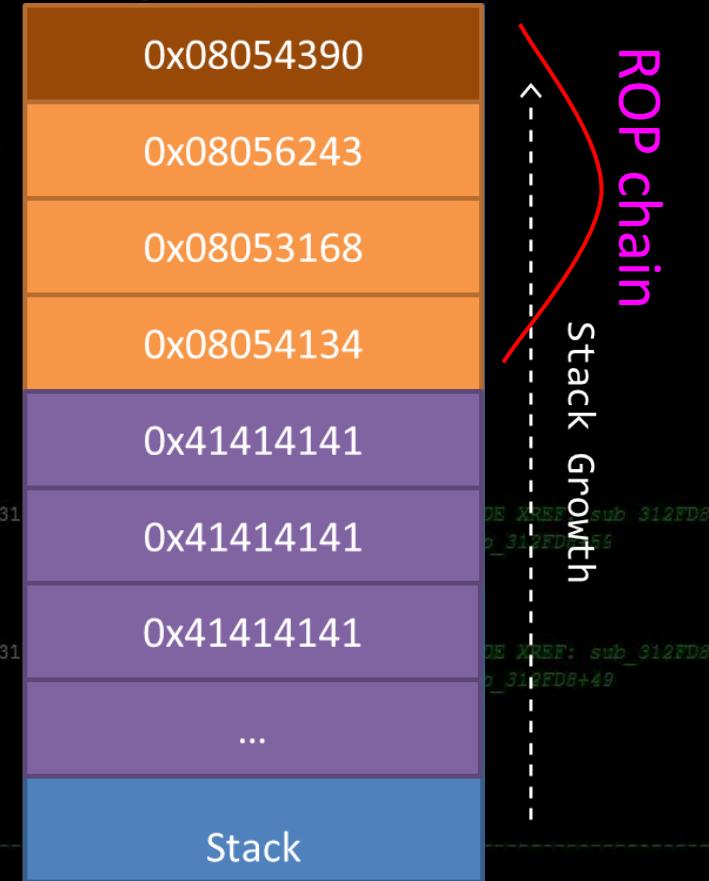
# Understanding ROP

exit(0) - ROP chain

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

ESP →

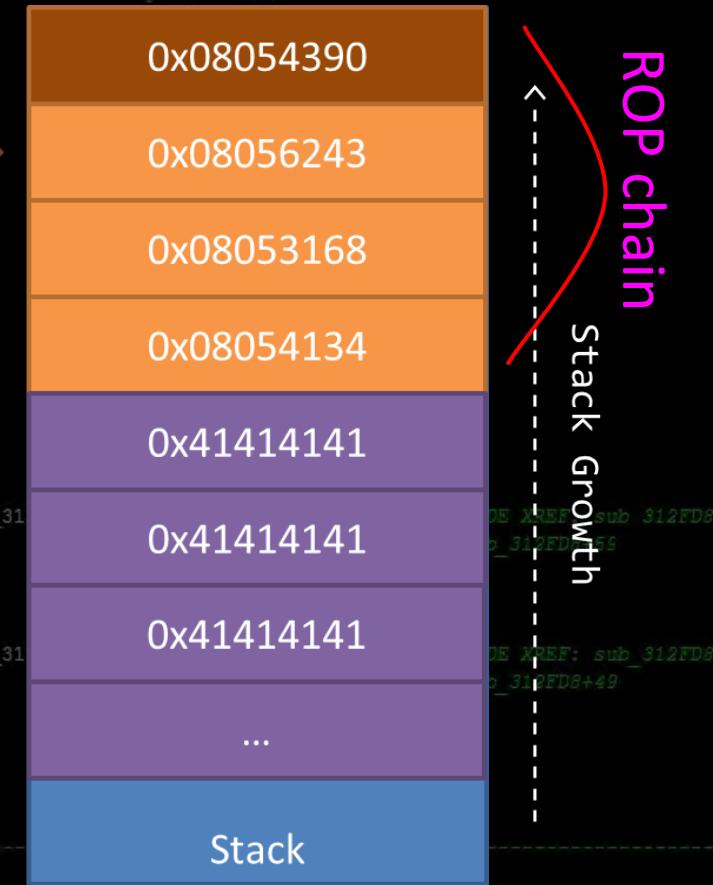
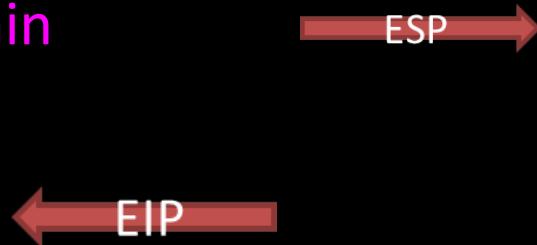
← EIP



# Understanding ROP

exit(0) - ROP chain

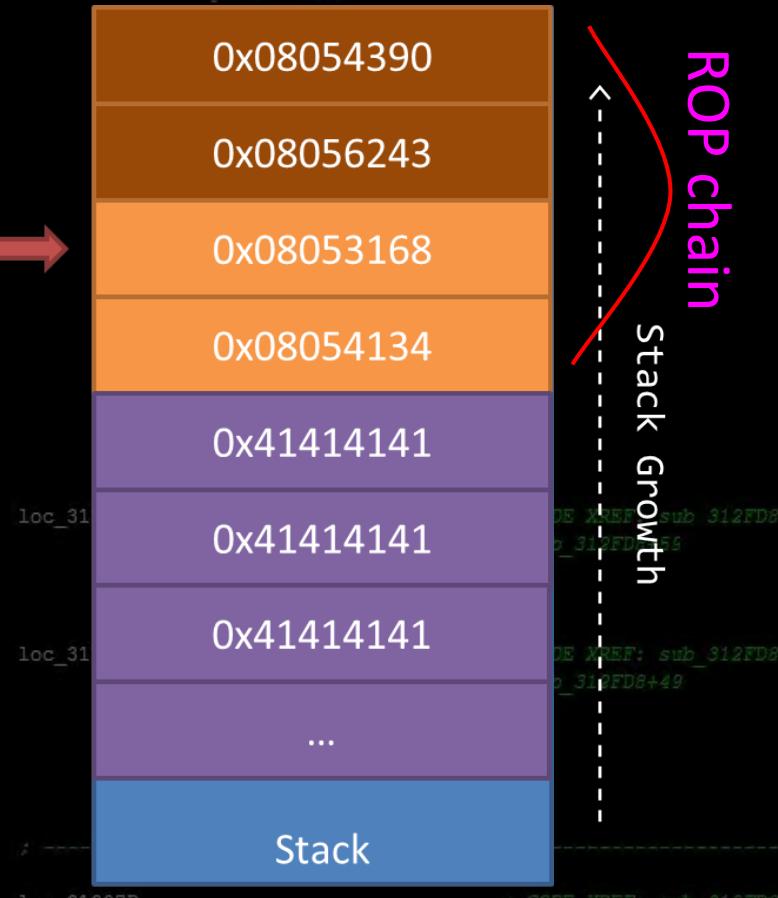
```
xor    eax, eax  
ret  
-----  
xor    ebx, ebx  
ret  
-----  
inc    eax  
ret  
-----  
int    0x80
```



# Understanding ROP

exit(0) - ROP chain

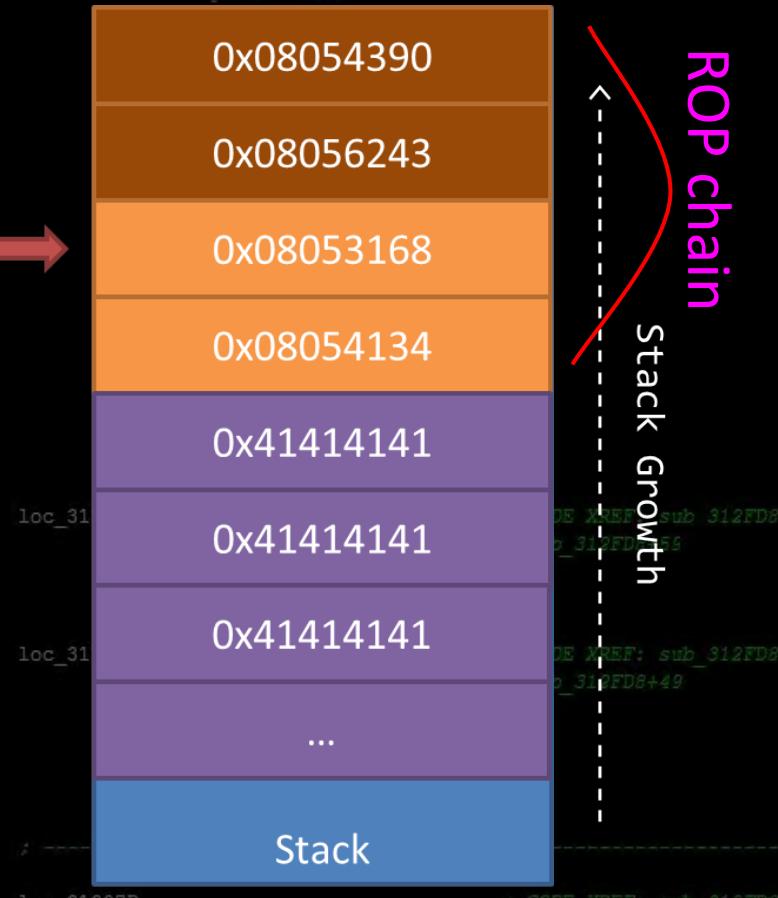
```
xor      eax, eax          ESP →  
ret  
.....  
xor      ebx, ebx ← EIP  
ret  
.....  
inc      eax  
ret  
.....  
int     0x80
```



# Understanding ROP

exit(0) - ROP chain

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



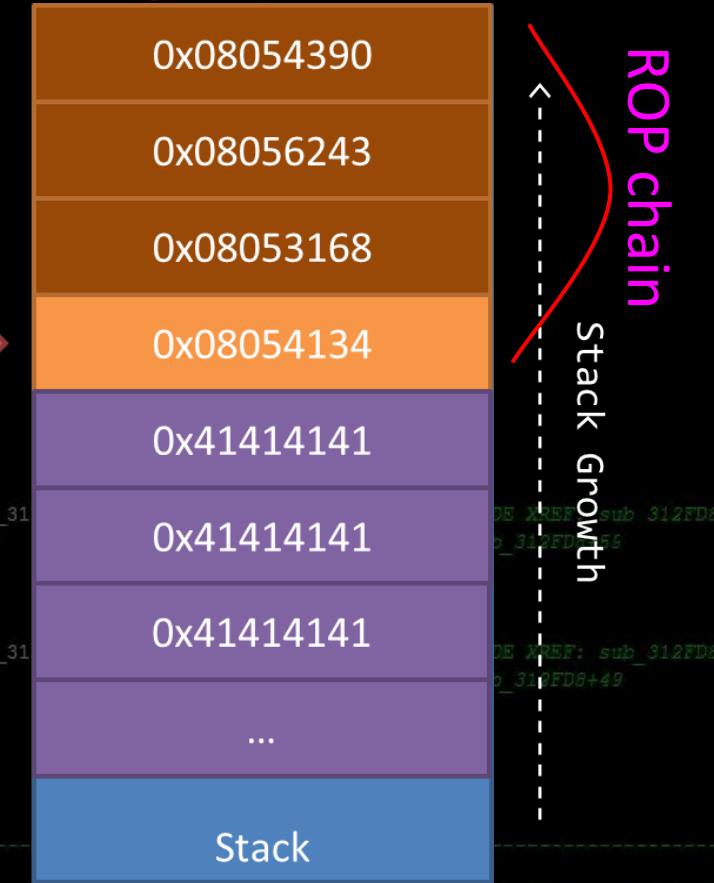
# Understanding ROP

exit(0) - ROP chain

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

ESP →

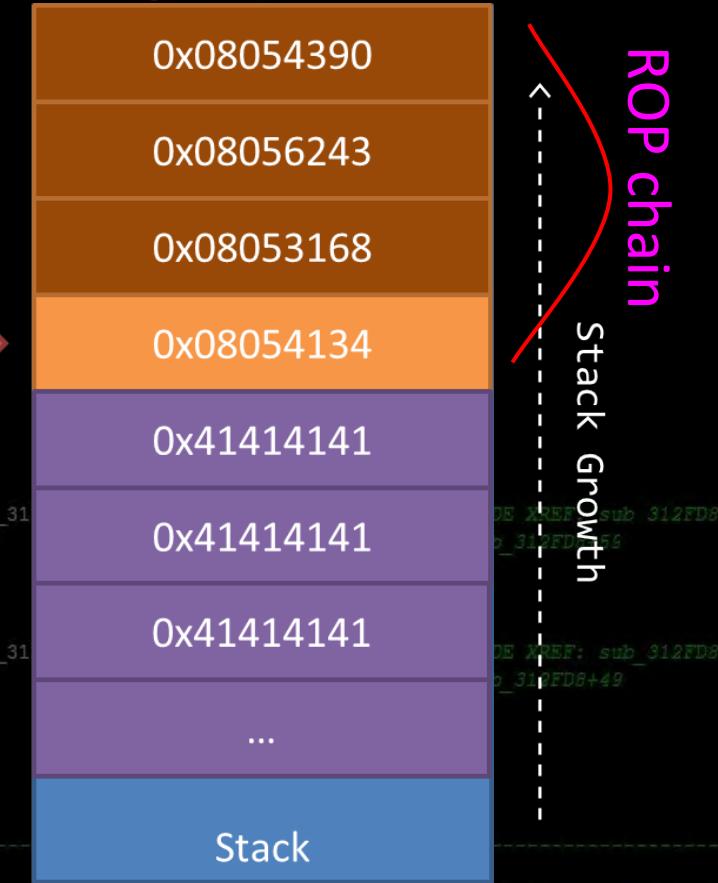
← EIP



# Understanding ROP

exit(0) - ROP chain

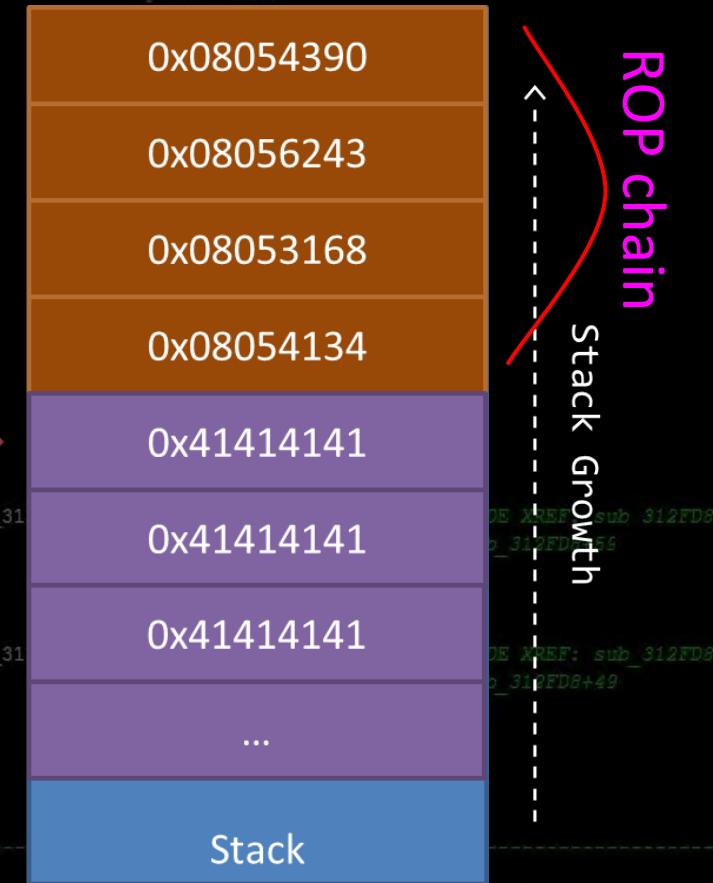
```
xor      eax, eax
ret
-----
xor      ebx, ebx
ret
-----
inc      eax
ret
-----
int     0x80
```



# Understanding ROP

exit(0) - ROP chain

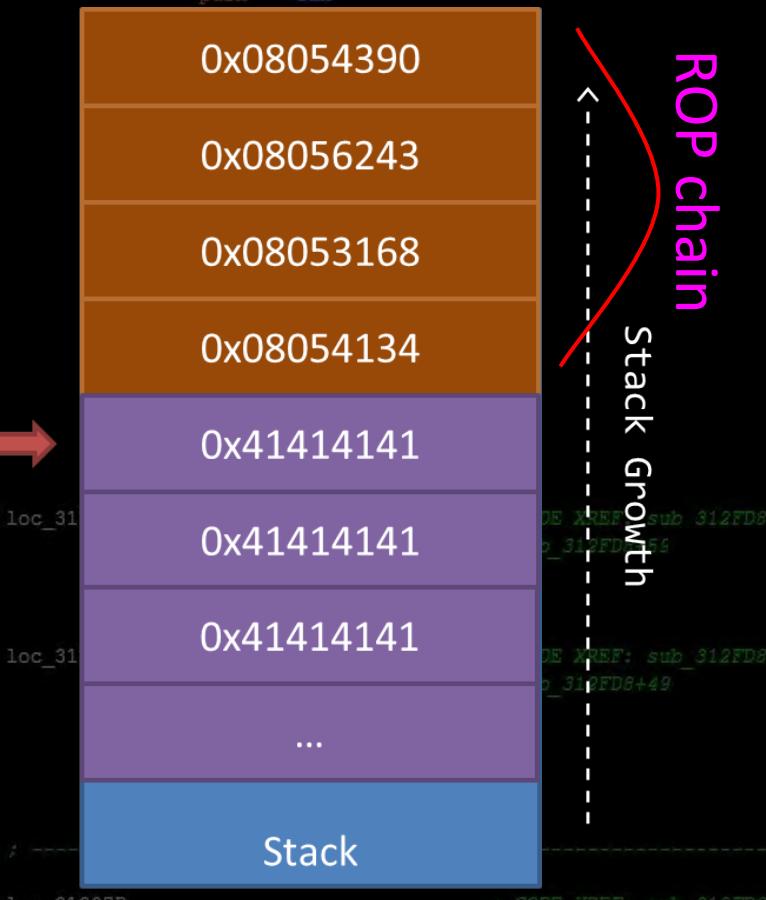
```
xor      eax, eax  
ret  
.....  
xor      ebx, ebx  
ret  
.....  
inc      eax  
ret  
.....  
int      0x80
```



# Understanding ROP

exit(0) - ROP chain

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

```
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
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
mov    esi, 1D0h
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
```

# /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?

```
push    edi
call    .sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
inc    short loc_313066
add    eax, [ebp+var_70]
add    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
[ebp+arg_0], eax
sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
[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
```

# 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)
```

# 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, 0FFFh
or     eax, 80070000h

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

# 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**

```
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
pushn   esi
push    eax
push    edi
[...]
call    .sub_31462A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
[...]
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
;

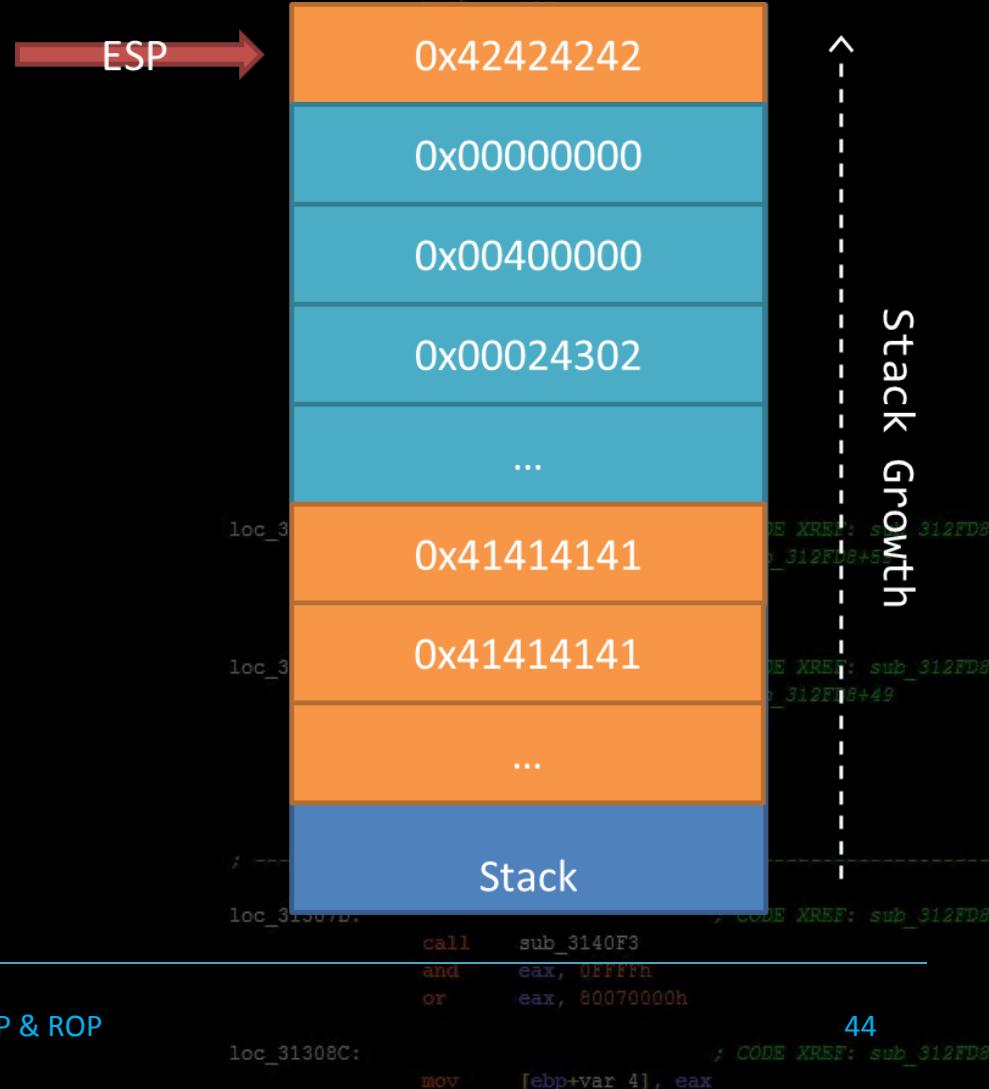
loc_31307D:          ; CODE XREF: sub_312FD8
call    .sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h

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

# Stack Pivoting

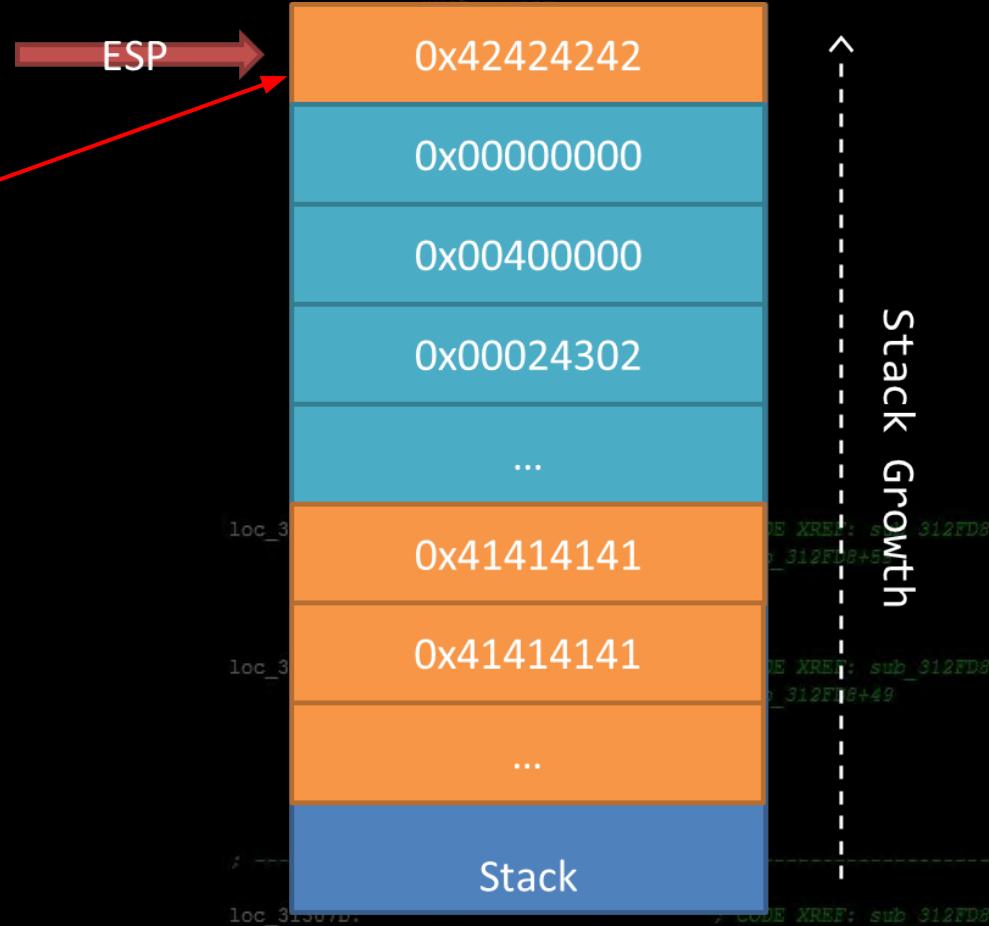
You control the orange

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

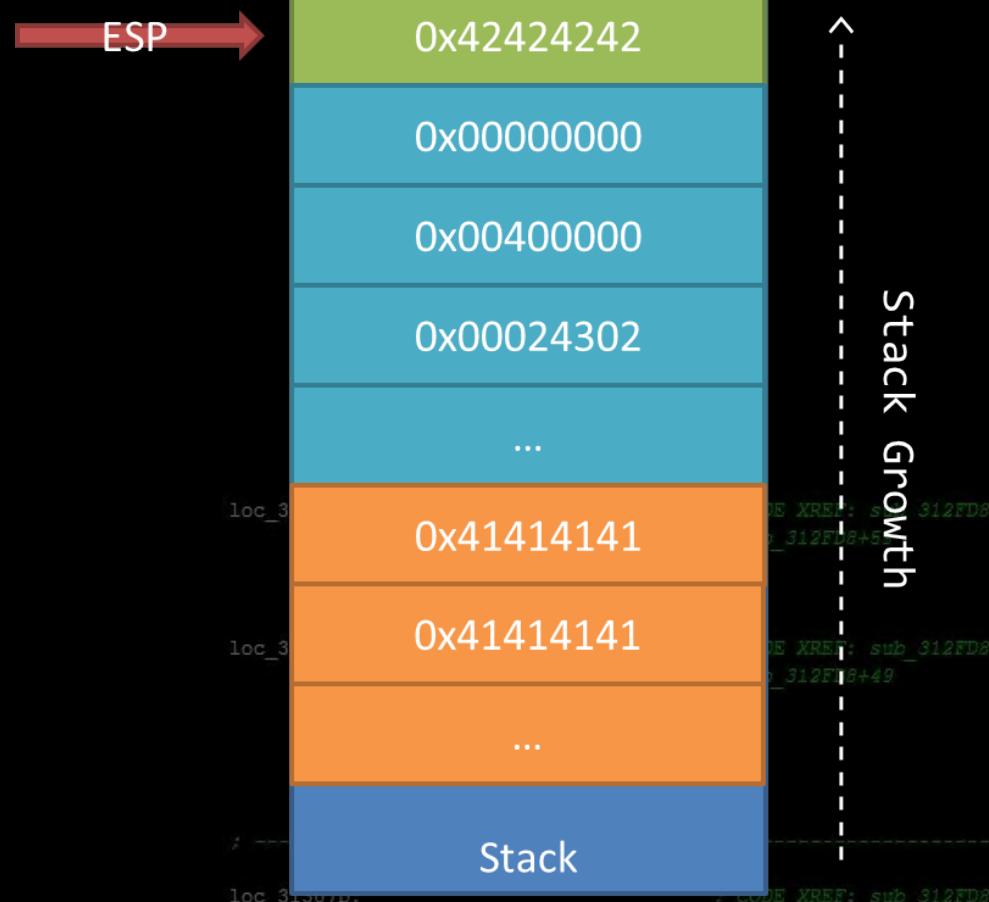


# Stack Pivoting

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)**



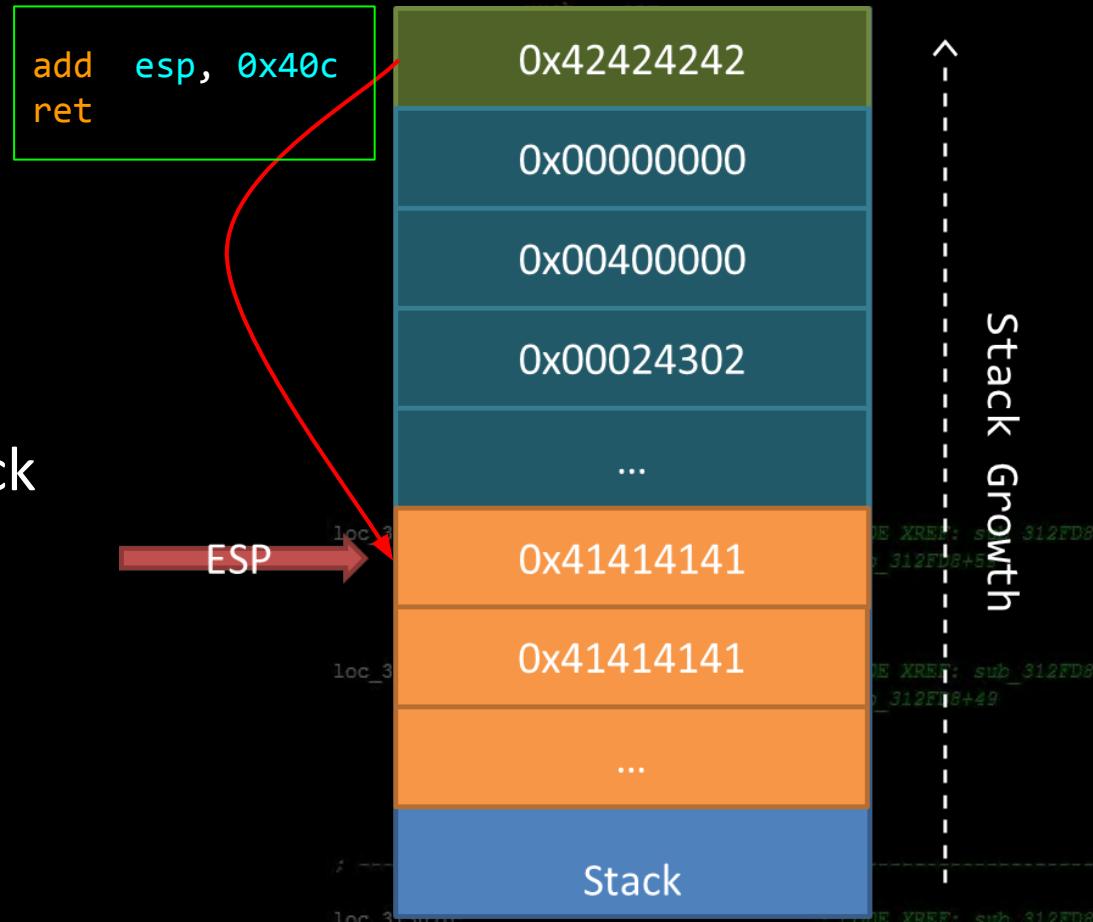
# Stack Pivoting

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)**

ESP

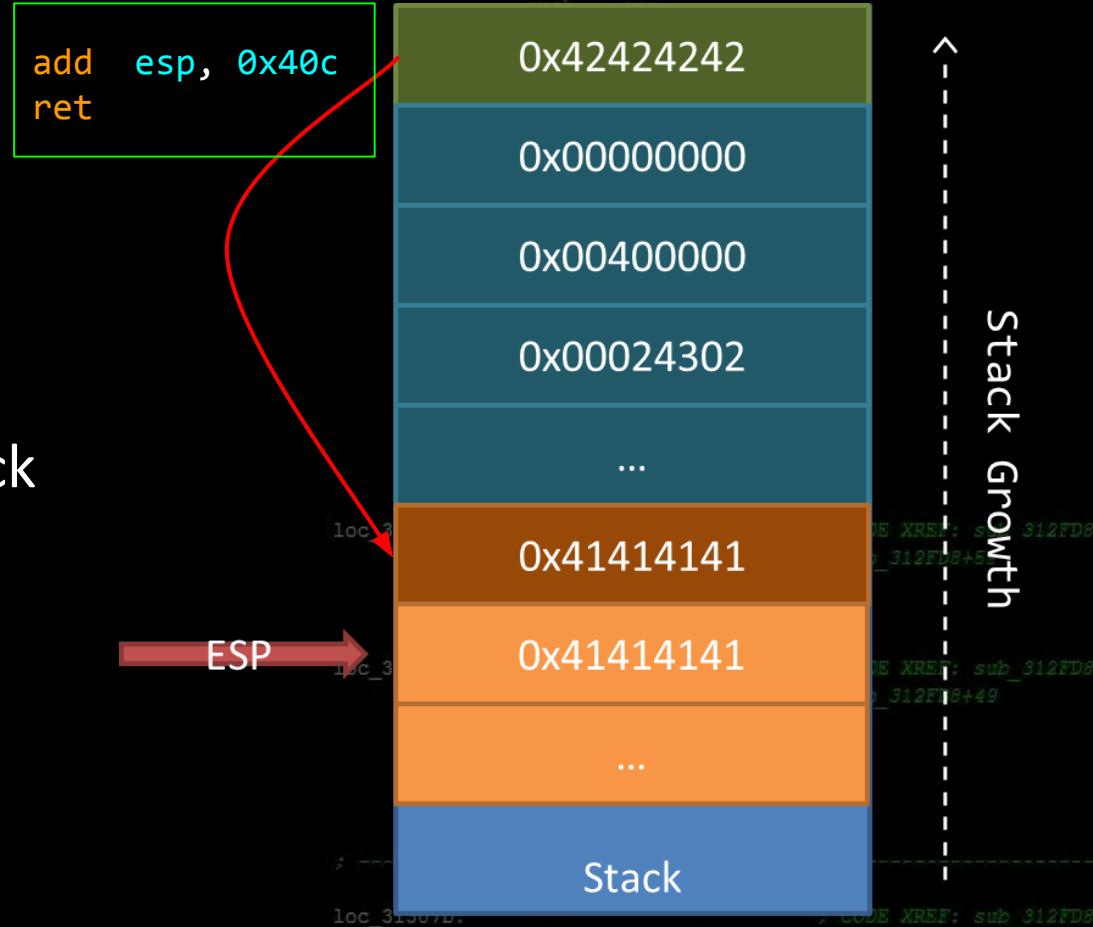


# Stack Pivoting

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 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

```
loc_313066:          ; CODE XREF: sub_312FD8+59  
                    ; sub_312FD8+59  
push  0Dh  
call  sub_31411B  
  
loc_31306D:          ; CODE XREF: sub_312FD8+49  
                    ; 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**

```
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
; CODE XREF: sub_312FD8+11
call    sub_31460A
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
; CODE XREF: sub_312FD8+12
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, 0FFFh
or     eax, 80070000h
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

# /levels/lecture/rop/rop\_pivot

- Play around with Stack Pivoting on the warzone

```
push    edi
call    .sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
int    short loc_313066
or     ea, [ebp+var_70]
jmp    ebx, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
int    [ebp+arg_01], eax
sal    sub_314623
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
```

# 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

```
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
pushn   esi
push    eax
push    edi
mov    [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz     short loc_31306D
push    eax
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
; CODE XREF: sub_312FD8
; sub_312FD8+59
push    0Dh
call    sub_31411B
loc_313066:
; CODE XREF: sub_312FD8
; sub_312FD8+49
push    0Dh
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

← EIP

.....  
system()

0xb7e65190: push ebx

0xb7e65191: sub esp, 8

0xb7e65194: mov eax, DWORD PTR  
[esp+0x10]

...

system() - - - >



# 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

```
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
[ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    eax, 1Dh
push    eax
push    [ebp+arg_4]
push    edi
sub    1Dh
test    eax, eax
jz     short loc_31306D
jmp    loc_313068
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313066:
push    0Dh
call    sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31306D:
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8
```

# ret2libc example

0x08045430: ret

← EIP

.....  
system()

0xb7e65190: push ebx

0xb7e65191: sub esp, 8

0xb7e65194: mov eax, DWORD PTR  
[esp+0x10]

...

system() - - - >



# ret2libc example

0x08045430: ret

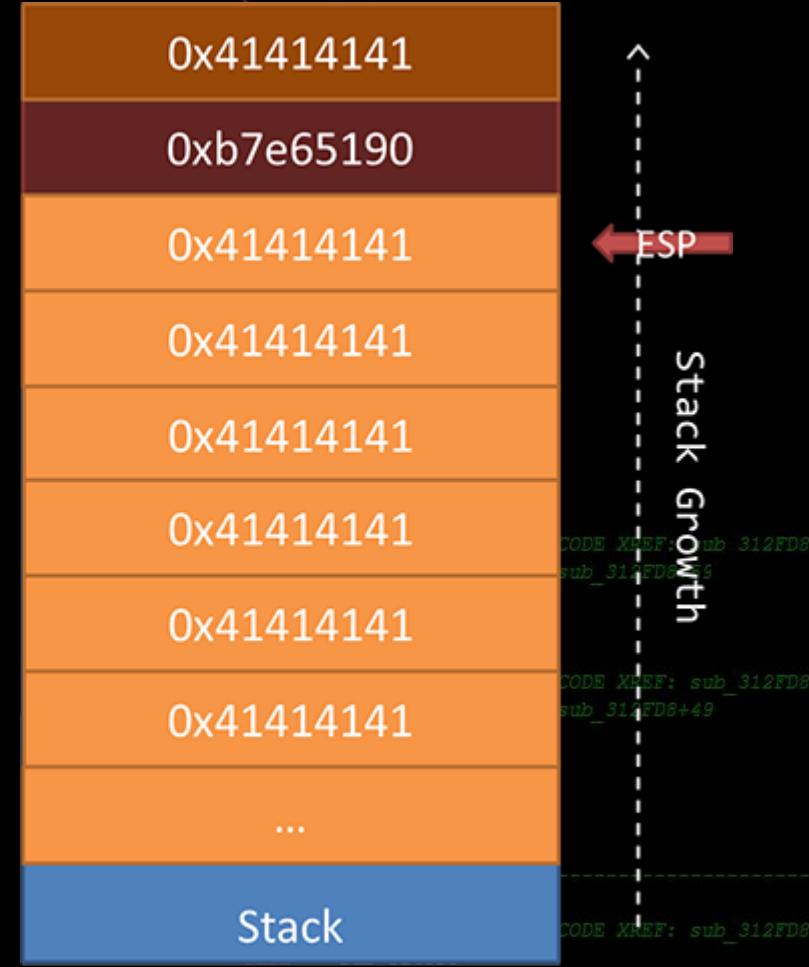
`system()`

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

• • •

MBE - 03/13/15

DEP & ROP

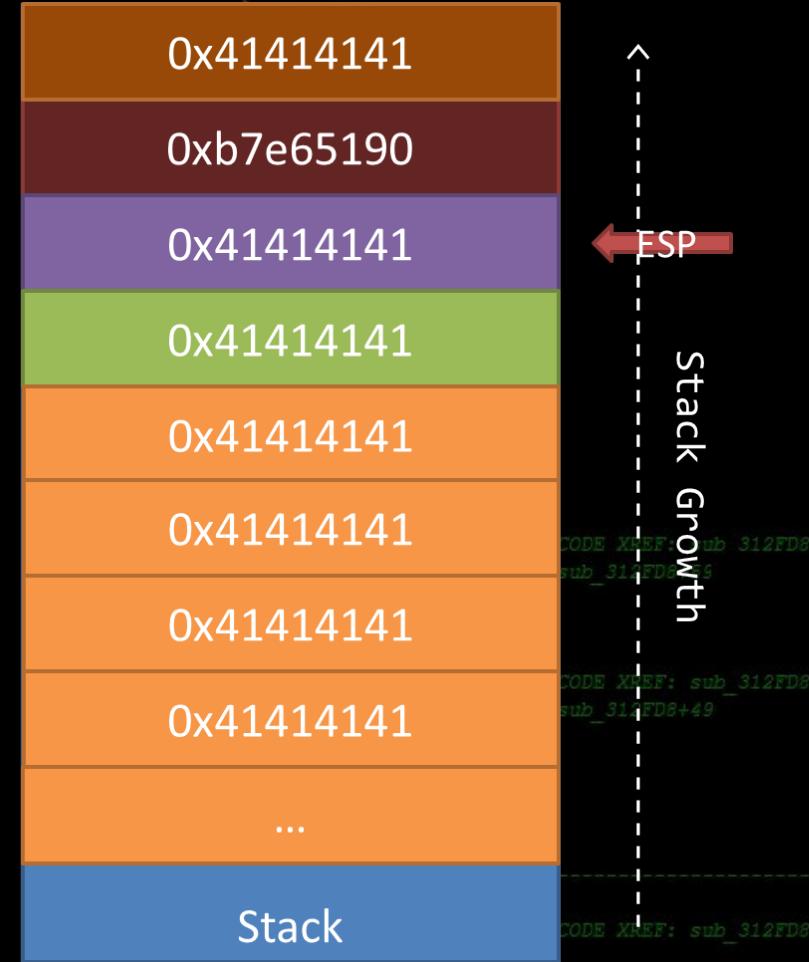


```
and    eax, 0FFFFF  
or     eax, 80070000h
```

# ret2libc example

```
system() --->
ret address --->
first arg --->

0x08045430: ret
.....
system()
0xb7e65190: push    ebx      ← EIP
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]

• • •

```
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+59  
call sub\_31411B

sub\_312FD8+59

sub\_312FD8+49

loc\_31306D: ; CODE XREF: sub\_312FD8+49
call sub\_3140F3
test eax, eax
jg short loc\_31307D
call sub\_3140F3
jmp short loc\_31308C

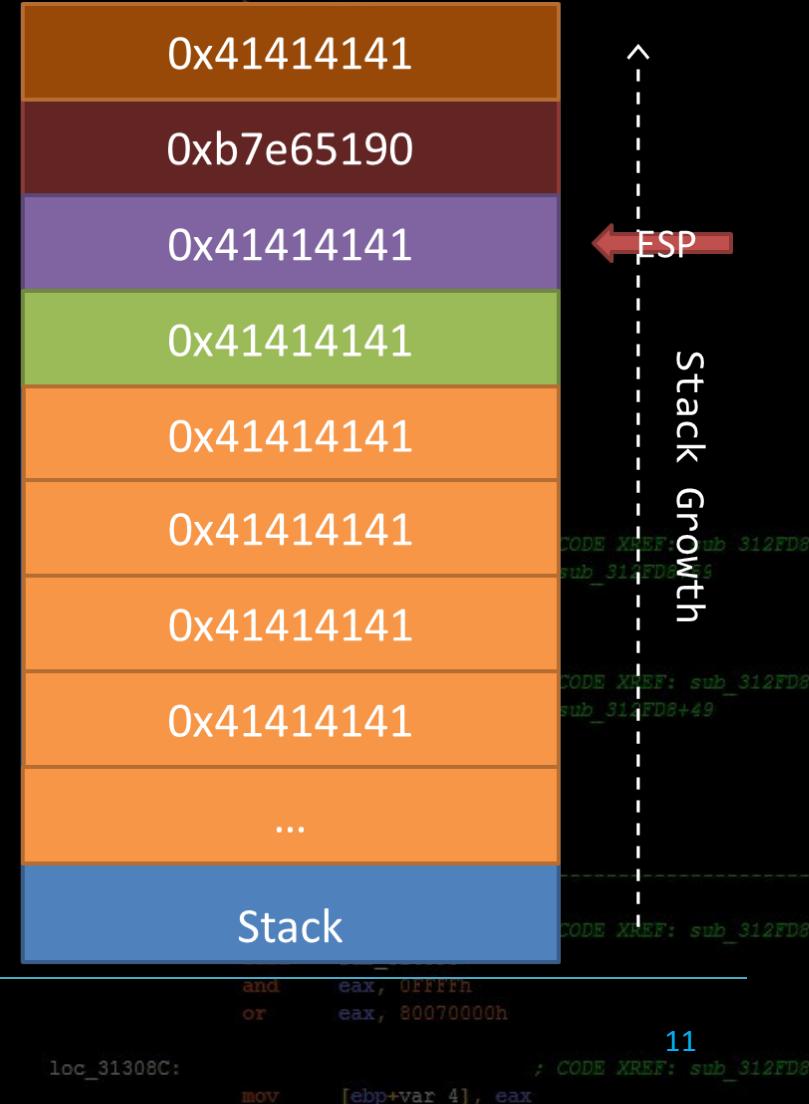
loc\_31307D: ; CODE XREF: sub\_312FD8+49
call sub\_3140F3
and eax, 0FFFFh
or eax, 80070000h

loc\_31308C: ; CODE XREF: sub\_312FD8+49
mov [ebp+var\_4], eax

# ret2libc example

```
system() --->
ret address --->
first arg --->

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

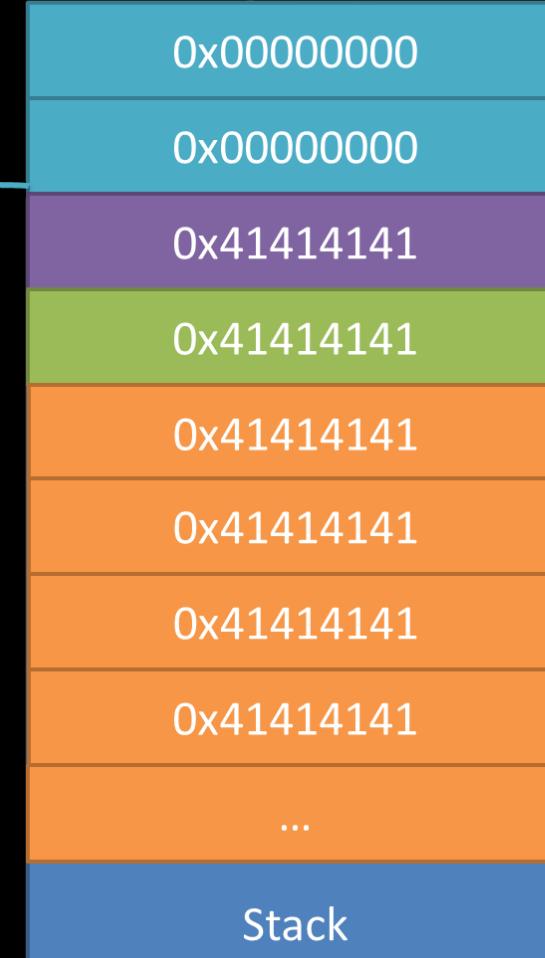
0xb7e65194: mov eax, DWORD PTR  
[esp+0x10]      ← EIP

...

system()'s  
stack frame

ret address --->

first arg --->



# ret2libc example

0x08045430: ret

.....  
system()

0xb7e65190: push ebx

0xb7e65191: sub esp, 8

0xb7e65194: mov eax, DWORD PTR  
[esp+0x10]      ← EIP

...

system()'s  
stack frame

ret address --->

first arg --->

0x00000000

0x00000000

0x41414141

0x41414141

0x41414141

0x41414141

0x41414141

0x41414141

...

Stack

ESP

Stack Growth

# 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+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
```

# ret2libc example

0x08045430: ret

← EIP

.....  
system()

0xb7e65190: push ebx

0xb7e65191: sub esp, 8

0xb7e65194: mov eax, DWORD PTR  
[esp+0x10]

...

system() - - - >



# ret2libc example

system() --->  
ret address --->  
first arg --->  
“cat flag.txt”

0x08045430: ret ← EIP

.....

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



# ret2libc example

0x08045430: ret

.....  
system()

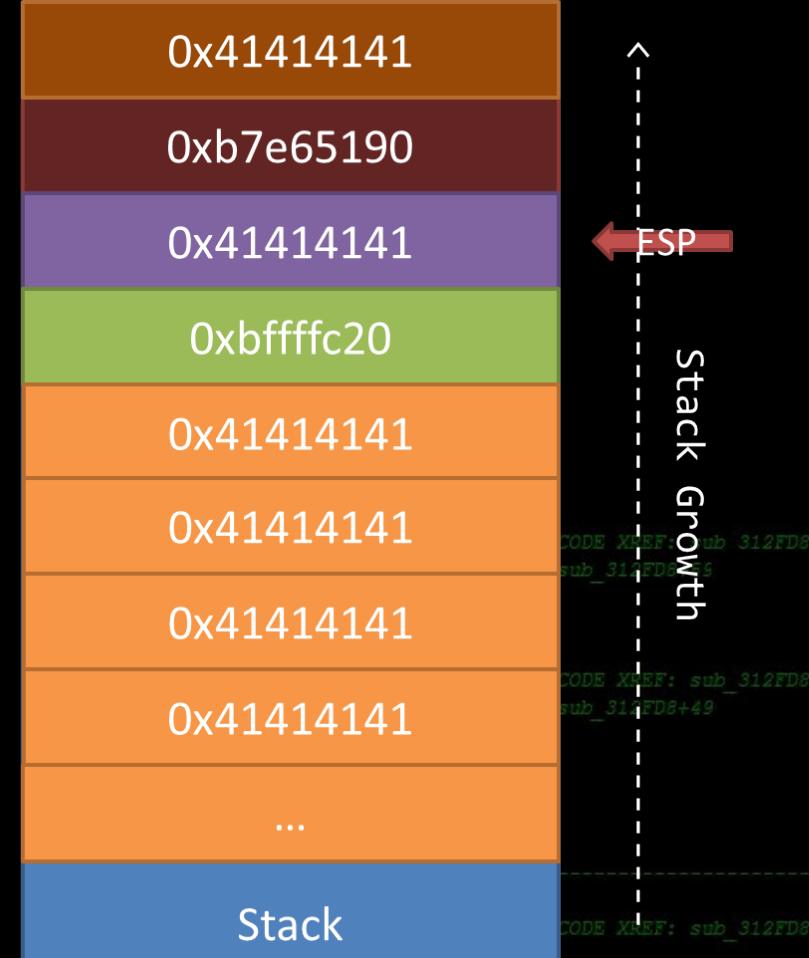
0xb7e65190: push ebx        
0xb7e65191: sub esp, 8  
0xb7e65194: mov eax, DWORD PTR [esp+0x10]

...

system() --->

ret address --->

first arg --->  
“cat flag.txt”



# ret2libc example

0x08045430: ret

.....  
system()

0xb7e65190: push ebx

0xb7e65191: sub esp, 8

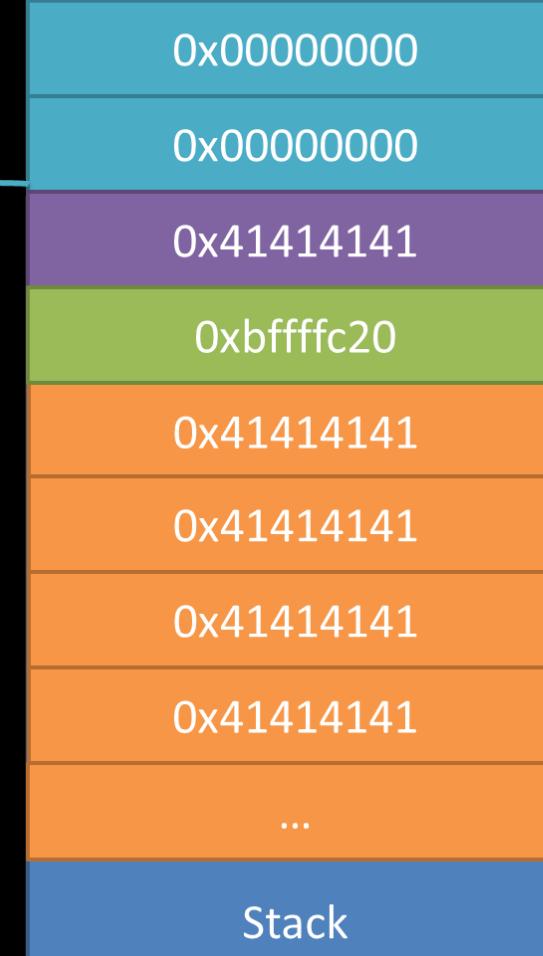
0xb7e65194: mov eax, DWORD PTR  
[esp+0x10]      ← EIP

...

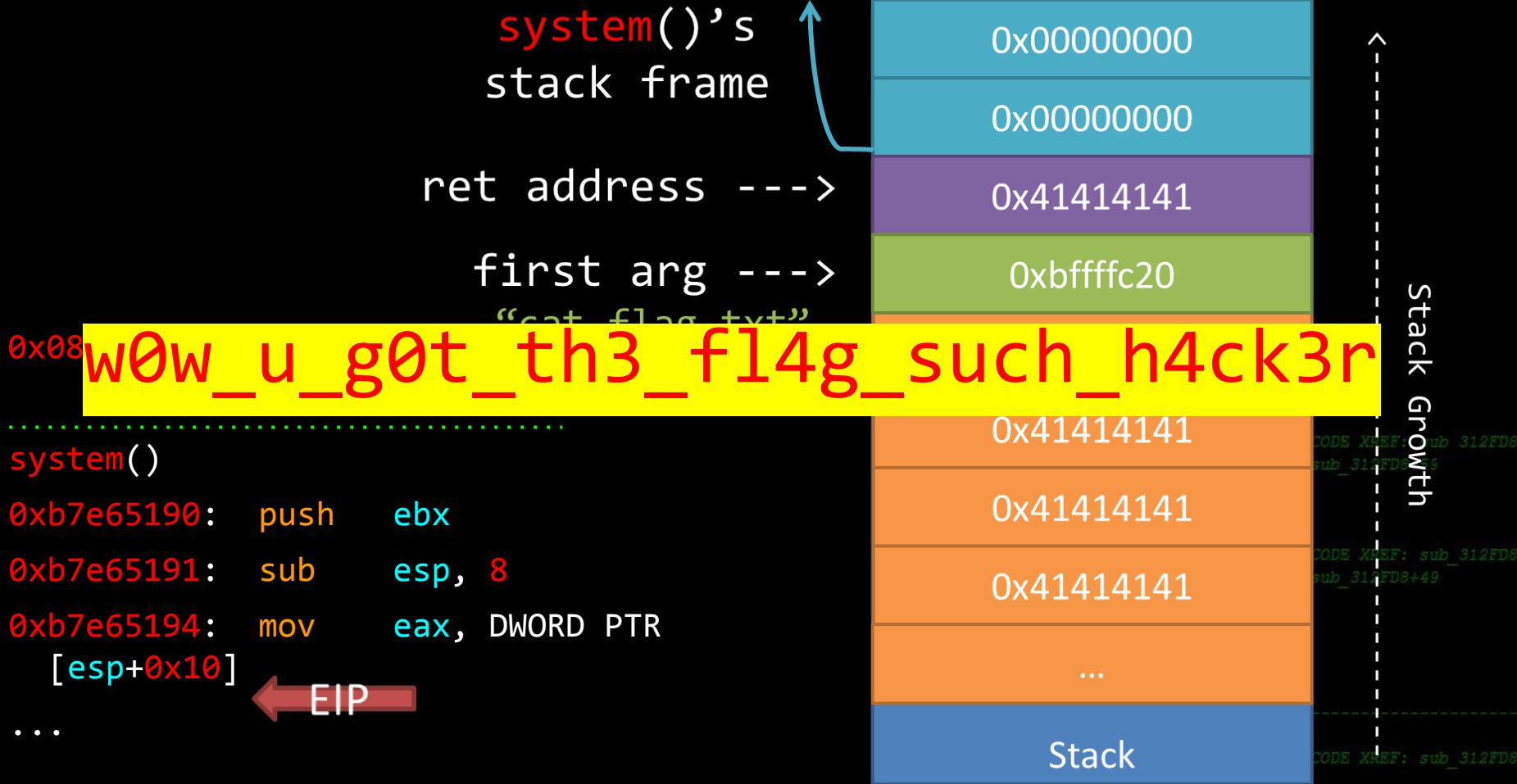
system()'s  
stack frame

ret address --->

first arg --->  
“cat flag.txt”

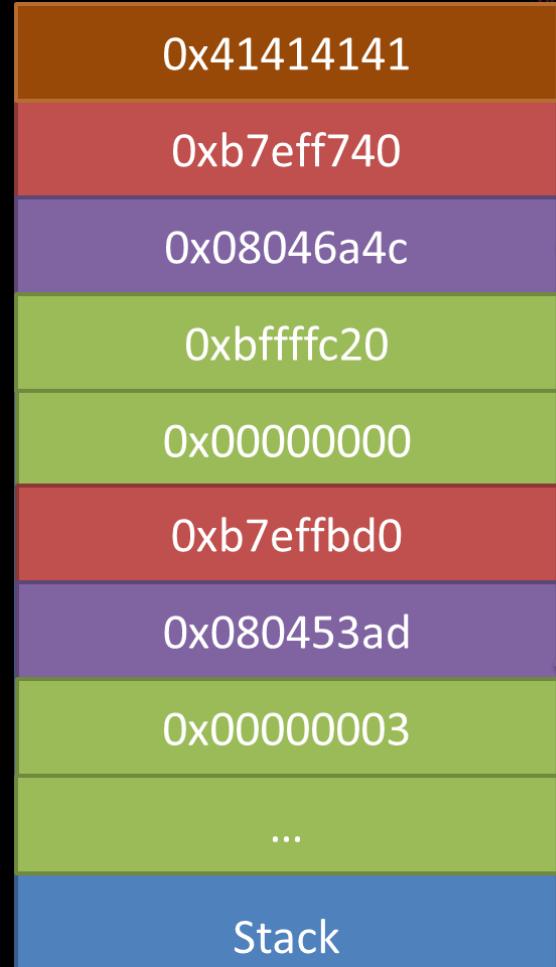


# ret2libc example



# Chaining Calls

open() --->  
pivot --->  
first arg --->  
second arg --->  
read() --->  
ret address --->  
first arg --->



```
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
sh    eax
sh    edi
v    [ebp+arg_0], eax
sub_31486A
st    short loc_31306D
sh    si
a    ax, [ebp+arg_0]
sh    si, 1D0h
si    si
sh    ebp+arg_4
di    sub_314623
sub_314623
eax    eax
short loc_31306D
short loc_31306F
short loc_31308F
short loc_31308E
short loc_31308D
short loc_31308C
short loc_31308B
short loc_31308A
short loc_313089
short loc_313088
short loc_313087
short loc_313086
short loc_313085
short loc_313084
short loc_313083
short loc_313082
short loc_313081
short loc_313080
short loc_31307F
short loc_31307E
short loc_31307D
short loc_31307C
short loc_31307B
short loc_31307A
short loc_313079
short loc_313078
short loc_313077
short loc_313076
short loc_313075
short loc_313074
short loc_313073
short loc_313072
short loc_313071
short loc_313070
short loc_313069
short loc_313068
short loc_313067
short loc_313066
short loc_313065
short loc_313064
short loc_313063
short loc_313062
short loc_313061
short loc_313060
short loc_313059
short loc_313058
short loc_313057
short loc_313056
short loc_313055
short loc_313054
short loc_313053
short loc_313052
short loc_313051
short loc_313050
short loc_313049
short loc_313048
short loc_313047
short loc_313046
short loc_313045
short loc_313044
short loc_313043
short loc_313042
short loc_313041
short loc_313040
short loc_313039
short loc_313038
short loc_313037
short loc_313036
short loc_313035
short loc_313034
short loc_313033
short loc_313032
short loc_313031
short loc_313030
short loc_313029
short loc_313028
short loc_313027
short loc_313026
short loc_313025
short loc_313024
short loc_313023
short loc_313022
short loc_313021
short loc_313020
short loc_313019
short loc_313018
short loc_313017
short loc_313016
short loc_313015
short loc_313014
short loc_313013
short loc_313012
short loc_313011
short loc_313010
short loc_313009
short loc_313008
short loc_313007
short loc_313006
short loc_313005
short loc_313004
short loc_313003
short loc_313002
short loc_313001
short loc_313000
```

ESP

Stack Growth