

x64, ARM, Windows

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, 1778
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_313067: ; 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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

Lecture Overview

- This course has largely revolved around exploiting **x86** binaries on **Ubuntu 14.04 i386**
 - **Linux** is easier and a bit more academic
 - Same can be said about **32bit x86**



ubuntu

```
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 [ebp+var_0], eax
call sub_31486A
test eax, eax
jz short loc_313066
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+var_0]
push edi
call sub_314623
test eax, eax
jz short loc_313066
cmp [ebp+arg_0], esi
jz short loc_31308F
```

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

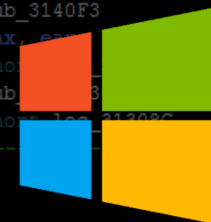
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C

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

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

Lecture Overview

- This course has largely revolved around exploiting **x86** binaries on **Ubuntu 14.04 i386**
 - **Linux** is easier and a bit more academic
 - Same can be said about **32bit x86**
- But how does exploitation change for **x86_64** systems? **ARM** devices? How about **Windows**?



Lecture Overview

- Architecture Differences
 - x86
 - x86_64
 - ARM
- Platform Differences
 - Windows

```
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
```

x86 Overview

- **x86** is a 32bit instruction set developed by Intel
 - Sometimes known as **x32**, **x86**, **IA32**

```
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], esi
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
```

x86 Overview

- **x86** is a 32bit instruction set developed by Intel
 - Sometimes known as **x32**, **x86**, **IA32**
- It's a CISC architecture that is super popular and used all around the world
 - **yadayadayada, you've been using it all semester**

```
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], esi
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 [ebp+arg_0], esi
cmp [ebp+arg_0], esi
jz short loc_31308F
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0Dh
call sub_31414B
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
```

x86 CPU



```
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
sub_31486A
eax, eax
short loc_31306D
esi
eax, [ebp+arg_0]
eax
esi, 1D0h
esi
[ebp+arg_4]
edi
sub_314623
eax, eax
short loc_31306D
[ebp+arg_0], esi
short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+55
0Dh
sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
sub_3140F3
eax, eax
short loc_31307D
sub_3140F3
short loc_31308C
```

```
loc_31307D: ; CODE XREF: sub_312FD8
```

```
call sub_3140F3
and eax, 0FFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

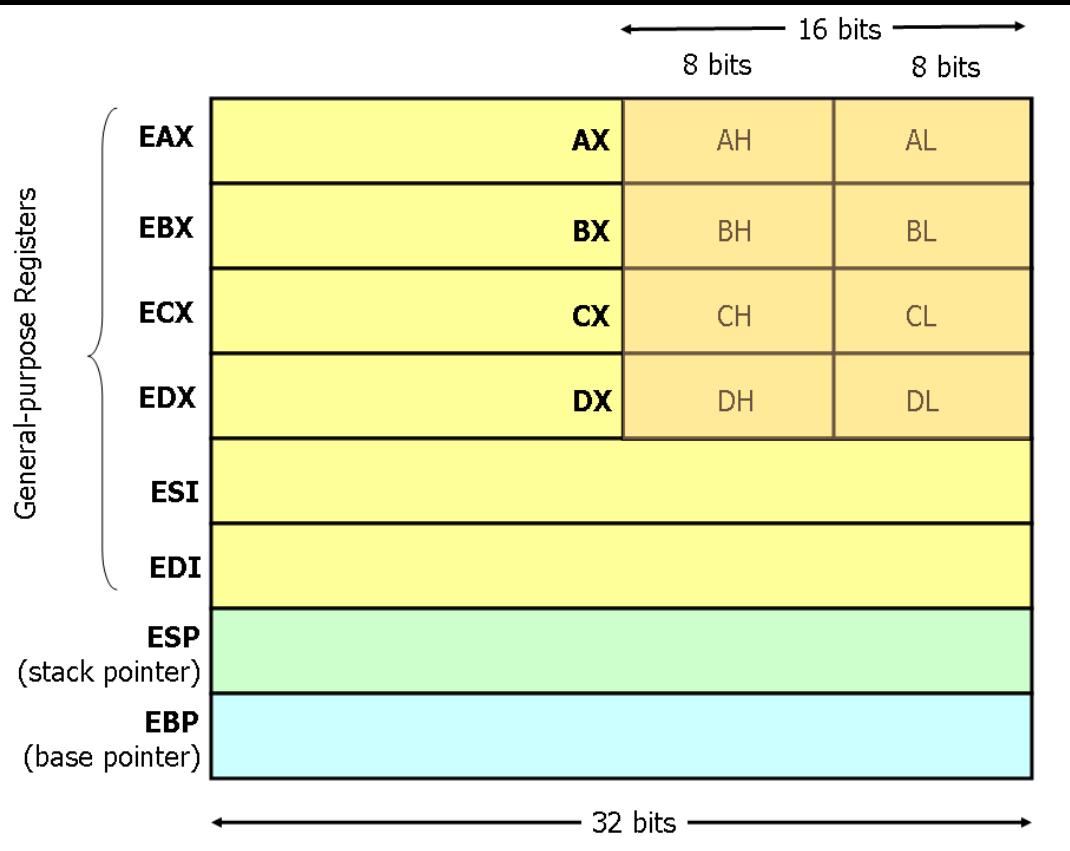
```
mov [ebp+var_4], eax
```

x86 Registers

```

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
486A
tax
loc_31306D
[ebp+arg_0]
D0h
arg_4]
4623
tax
loc_31306D
arg_0], esi
loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+55
411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
40F3
tax
loc_31307D
40F3
loc_31308C

```

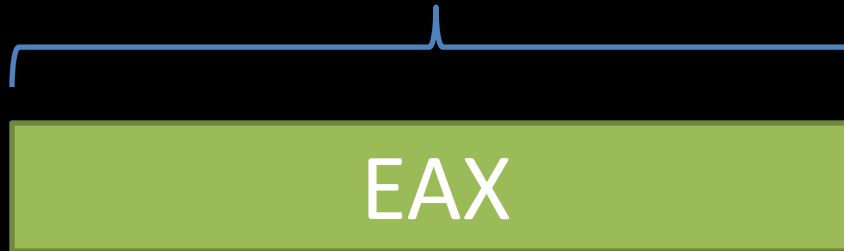
```

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

```

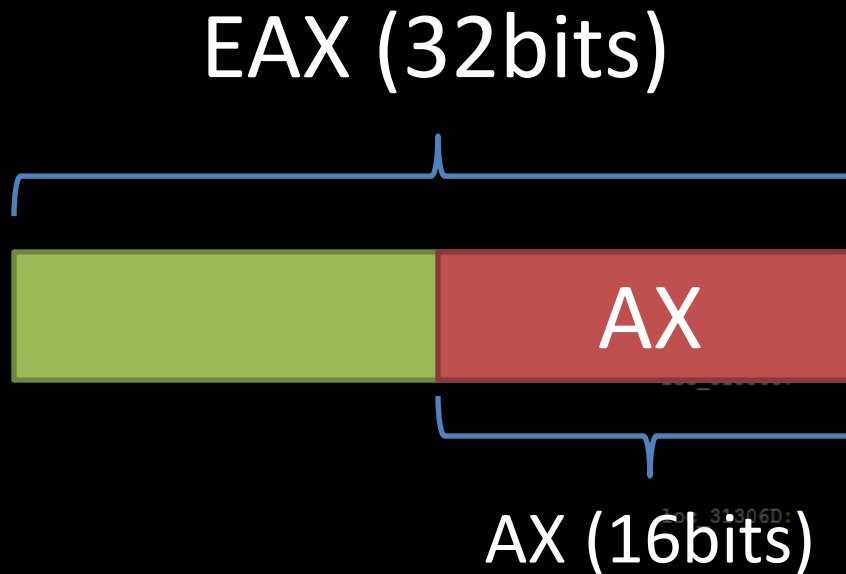

x86 Registers

EAX (32bits)

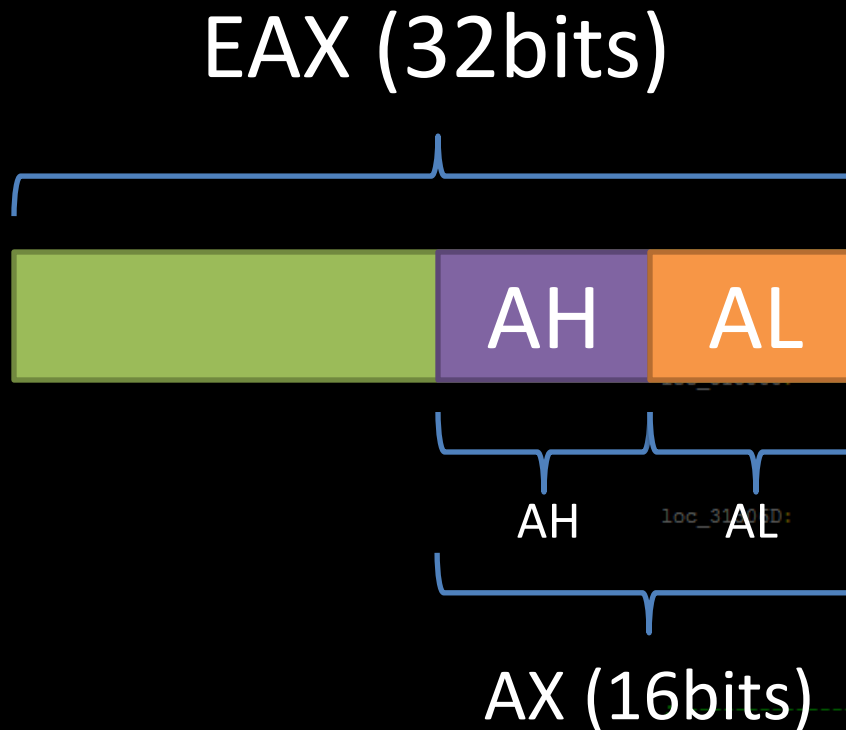


```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jb short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
push edi
mov [ebp+arg_0], eax
call sub_31486A
test eax, eax
jz short loc_31306D
push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

x86 Registers



x86 Registers



```
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+55
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
```

x86 Calling Conventions

- **cdecl**
 - **Caller** cleans up the stack
 - Unknown or variable # of arguments, eg **printf()**
- **stdcall**
 - **Callee** cleans up the stack
 - Standard calling convention for the **Win32 API**
- **fastcall**
 - First two arguments are put into **ECX**, and **EDX**, the rest are put onto the stack

```
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
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+55
push 0Dh
call sub_31411B
```

```
loc_31306D: ; CODE XREF: sub_312FD8+49
```

```
call sub_3140F3
test eax, eax
jnz short loc_31307D
call sub_3140F3
jmp short loc_31308C
```

```
loc_31307D: ; CODE XREF: sub_312FD8
```

```
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

x86 Misc Notes

- **x86** is like the wild west in computing
 - “it’s like it was designed to be **exploited**”

```
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: call    sub_31486A, eax
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
```

x86 Misc Notes

- **x86** is like the wild west in computing
 - “it’s like it was designed to be **exploited**”
 - No instruction alignment, and you can jump in the middle of instructions (great for **ROP Gadgets**)

```
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_31307D
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
```

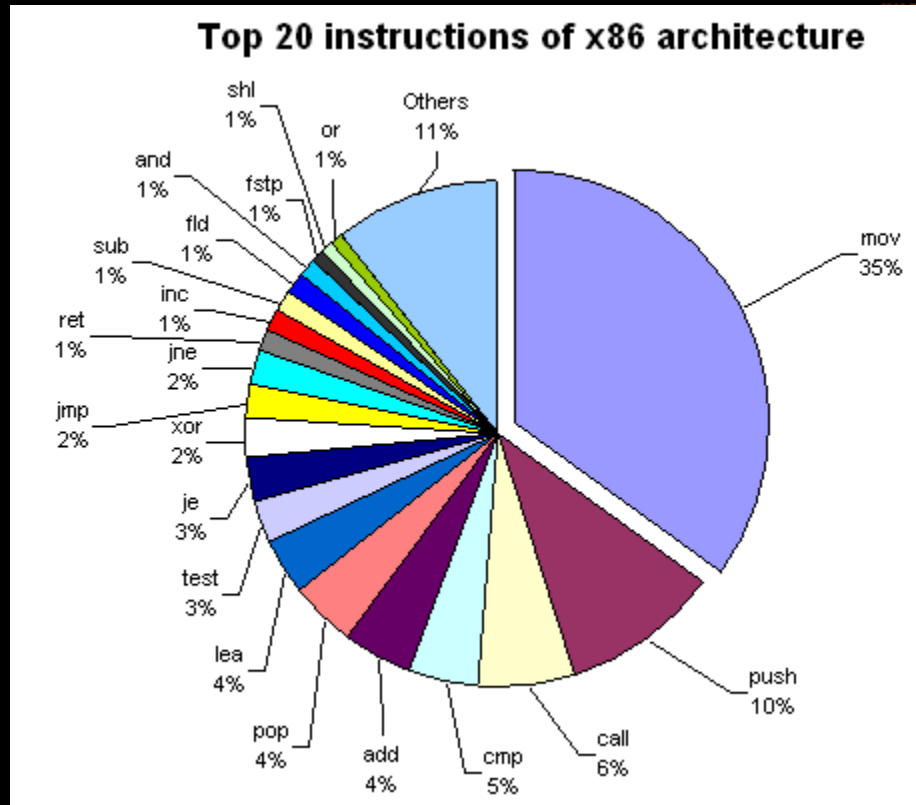
```
mov    [ebp+var_4], eax
```

x86 Misc Notes

- **x86** is like the wild west in computing
 - “it’s like it was designed to be **exploited**”
 - No instruction alignment, and you can jump in the middle of instructions (great for **ROP Gadgets**)
 - Hundreds of instructions, many rarely used

```
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, 100h
push [ebp+arg_4]
push edi
call sub_313070
test eax, eax
jz short loc_31306D
cp [ebp+arg_1], esi
;
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, 0FFFFFFh
or eax, 80070000h
;
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

x86 Instruction Stats



http://www.strchr.com/x86_machine_code_statistics

```
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
eax
edi
[ebp+arg_0], eax
sub_31486A
eax, eax
short loc_31306D
esi
eax, [ebp+arg_0]
eax
esi, 1D0h
esi
[ebp+arg_4]
edi
sub_314623
eax, eax
short loc_31306D
[ebp+arg_0], esi
short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+55
0Dh
sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
sub_3140F3
eax, eax
short loc_31307D
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
```


x86 Misc Notes

- **x86** is like the wild west in computing
 - “it’s like it was designed to be **exploited**”
 - No instruction alignment, and you can jump in the middle of instructions (great for **ROP Gadgets**)
 - Hundreds of instructions, many rarely used
 - Instructions can range from **1** byte long, to **15** bytes long!

```
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, 100h
push [ebp+arg_4]
push edi
call sub_3140F3
test eax, eax
jz short loc_31306D
cmp [ebp+arg_1], esi
jnz short loc_31308F

loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
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, 0FFFFFFh
or eax, 80070000h

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

x86 Long Instructions

```
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
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F
```

lock add DWORD PTR ds:[esi+ecx*4+0x12345678], 0xefcdab89

67 66 f0 3e 81 84 8e 78 56 34 12 89 ab cd ef ; 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
jz short loc_31307D
call sub_3140F3
jmp short loc_31308C
```

(from <http://blog.onlinedisassembler.com/blog/?p=23>)

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
```

x86 Misc Notes

- **x86** is like the wild west in computing
 - “it’s like it was designed to be **exploited**”
 - No instruction alignment, and you can jump in the middle of instructions (great for **ROP Gadgets**)
 - Hundreds of instructions, many rarely used
 - Instructions can range from **1** byte long, to **15** bytes long!

- **It’s the devil’s playground**

```
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, 100h
push [ebp+arg_4]
push edi
call sub_31307D
test eax, eax
jz short loc_31306D
cmp [ebp+arg_1], esi
call sub_31308F
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Lecture Overview

- Architecture Differences
 - x86
 - x86_64
 - ARM
- Platform Differences
 - Windows

```
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
```

x86_64 Overview

- x86_64 is the 64bit successor to 32bit x86
 - Sometimes known as x64, x86_64, AMD64

```
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    [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz     short loc_31306D
lea    esi, [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
```

x86_64 Overview

- **x86_64** is the **64bit** successor to **32bit x86**
 - Sometimes known as **x64**, **x86_64**, **AMD64**
- We're well into the **64bit** era at this point with **32bit x86** machines slowly on their way out

```
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   [ebp+arg_0], eax
call   sub_31486A
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
cmp    [ebp+arg_0], esi
jz     short loc_31308F
```

```
loc_313066:
push   0Dh
call   sub_31411B
```

```
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
```

```
loc_31308C:
mov    [ebp+var_4], eax
```

x86_64 Overview

- **x86_64** is the **64bit** successor to **32bit x86**
 - Sometimes known as **x64**, **x86_64**, **AMD64**
- We're well into the **64bit** era at this point with **32bit x86** machines slowly on their way out
- **x86_64** is Bigger, better, faster... and familiar!

```
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   [ebp+arg_0]
call  sub_31486A
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
cmp   [ebp+arg_0], esi
jz   short loc_31308F
loc_313066:
push  0Dh
call  sub_31411B
```

```
loc_31306D:
; CODE XREF: sub_312FD8
; sub_312FD8+55
```

```
push  0Dh
call  sub_31411B
loc_31307D:
; CODE XREF: sub_312FD8
; sub_312FD8+49
call  sub_314623
and  eax, 0FFFFFFh
or   eax, 80070000h
```

```
loc_31307D:
; CODE XREF: sub_312FD8
; sub_312FD8+49
call  sub_3140F3
```

```
loc_31308C:
; CODE XREF: sub_312FD8
; sub_312FD8+55
mov  [ebp+var_4], eax
```

x86_64 CPU



```
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
eax
edi
[ebp+arg_0], eax
sub_31486A
eax, eax
short loc_31306D
esi
eax, [ebp+arg_0]
eax
esi, 1D0h
esi
[ebp+arg_4]
edi
sub_314623
eax, eax
short loc_31306D
[ebp+arg_0], esi
short loc_31308F

; CODE XREF: sub_312FD8
; sub_312FD8+55

0Dh
sub_31411B

; CODE XREF: sub_312FD8
; sub_312FD8+49

sub_3140F3
eax, eax
short loc_31307D
sub_3140F3
short loc_31308C

; CODE XREF: sub_312FD8

sub_3140F3
and eax, 0FFFFh
or eax, 80070000h
```


x86_64 Registers

- Pretty similar to x86, but with a few upgrades

- General Purpose Registers

- Everything starts with **R** instead of **E** - **RAX, RBX, RCX...**
- GPR's are now **64bit**, not **32bit**
- There is now 8 more GPR's for use - **R8** to **R15**

- More **XMM*** registers (**128 bits**)

```
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
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
push [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
```

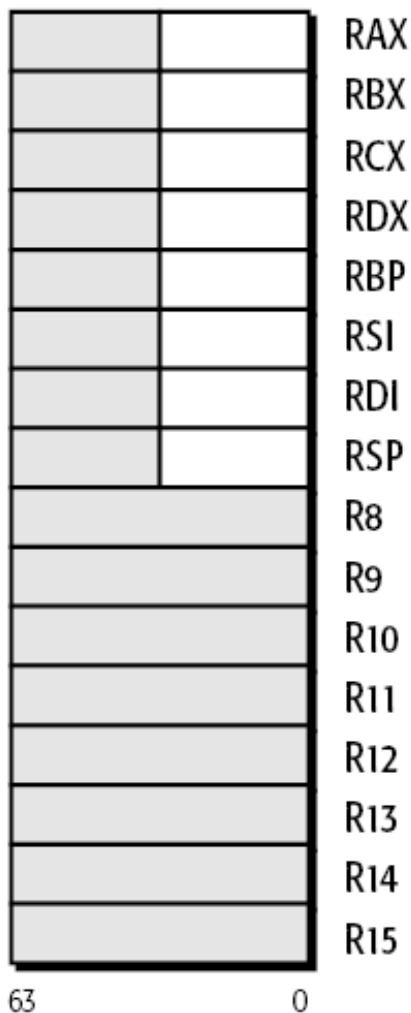
```
loc_31307D: ; CODE XREF: sub_312FD8
```

```
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
```

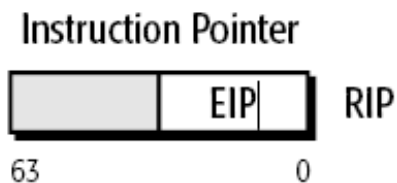
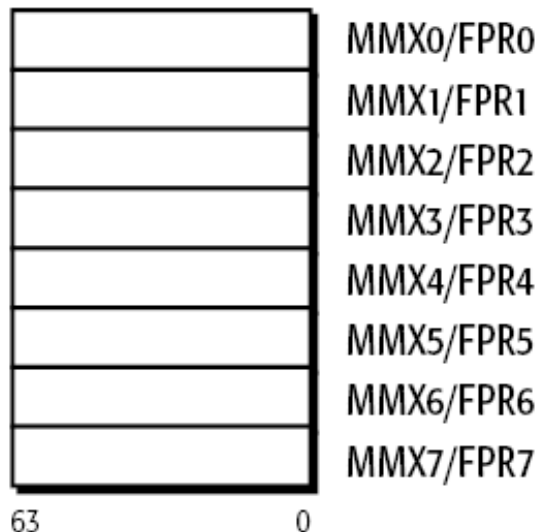
```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

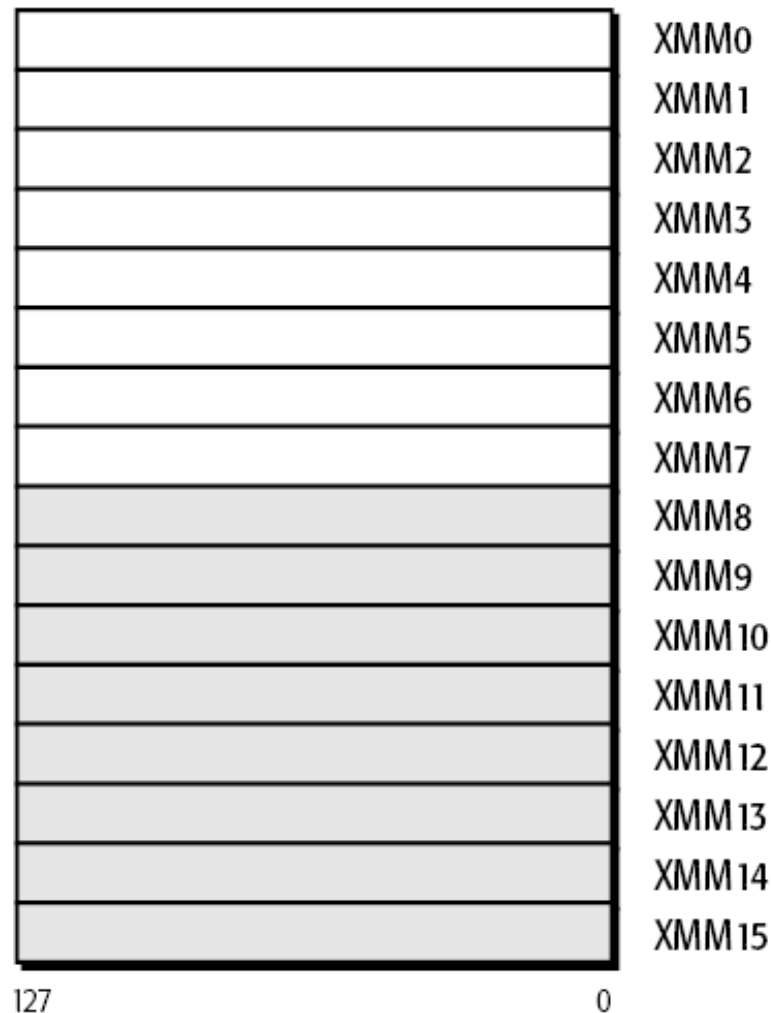
General-Purpose Registers (GPRs)





64-Bit Media and Floating-Point Registers



128-Bit Media Registers



-  Legacy x86 registers, supported in all modes
-  Register extensions, supported in 64-bit mode

Application-programming registers also include the 128-bit media control-and-status register and the x87 tag-word, control-word, and status-word registers

x86_64 Registers

RAX (64bits)

RAX

```
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
```

```
push    0Dh
call   sub_31411B
```

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
```

loc_31308C:

```
mov    [ebp+var_4], eax
```

x86_64 Registers

RAX (64bits)

EAX (32bits)

AX (16bits)

```
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, 100h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
push 0Dh
call sub_31411B
loc_31306D:
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp sub_31307D
loc_31307D:
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
```

x86_64 Registers

```

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

```

Register encoding	Not modified for 8-bit operands				Low 8-bit	16-bit	32-bit	64-bit
	Not modified for 16-bit operands							
	Zero-extended for 32-bit operands							
0			AH†	AL	AX	EAX	RAX	
3			BH†	BL	BX	EBX	RBX	
1			CH†	CL	CX	ECX	RCX	
2			DH†	DL	DX	EDX	RDY	
6				SIL‡	SI	ESI	RSI	
7				DIL‡	DI	EDI	RDI	
5				BPL‡	BP	EBP	RBP	
4				SPL‡	SP	ESP	RSP	
8				R8B	R8W	R8D	R8	
9				R9B	R9W	R9D	R9	
10				R10B	R10W	R10D	R10	
11				R11B	R11W	R11D	R11	
12				R12B	R12W	R12D	R12	
13				R13B	R13W	R13D	R13	
14				R14B	R14W	R14D	R14	
15				R15B	R15W	R15D	R15	

63 32 31 16 15 8 7 0

† Not legal with REX prefix ‡ Requires REX prefix

```

CODE XREF: sub_312FD8
sub_312FD8+55
CODE XREF: sub_312FD8
sub_312FD8+49
CODE XREF: sub_312FD8

```

```

and eax, 0FFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
CODE XREF: sub_312FD8

```

x86_64 Calling Conventions

- The **64bit** calling convention is a lot like **32bit** fastcall where arguments are put into registers

```
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
mov    [ebp+var_70], 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
```

x86_64 Calling Conventions

- The **64bit** calling convention is a lot like **32bit** fastcall where arguments are put into registers
- But **Linux** and **Windows** use different registers for their respective calling conventions

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
mov eax, [ebp+var_70]
call eax, [ebp+var_84]
jb short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
mov eax, [ebp+arg_0]
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
mov eax, [ebp+var_70]
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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

x86_64 Calling Conventions

- The **64bit** calling convention is a lot like **32bit** fastcall where arguments are put into registers
- But **Linux** and **Windows** use different registers for their respective calling conventions
 - **Linux: RDI, RSI, RDX, RCX, R8, R9**
 - **Windows: RCX, RDX, R8, R9**

```
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]
call eax, [ebp+var_84]
jb short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
mov [ebp+var_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
mov [ebp+var_0], eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jnz 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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```


x86_64 Calling Conventions

- The **64bit** calling convention is a lot like **32bit** fastcall where arguments are put into registers
- But **Linux** and **Windows** use different registers for their respective calling conventions

– **Linux: RDI, RSI, RDX, RCX, R8, R9**

– **Windows: RCX, RDX, R8, R9**

(any other arguments are pushed onto the stack)

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
lea eax, [ebp+var_70]
call eax, [ebp+var_84]
jb short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
push [ebp+arg_0]
call sub_31486A
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
cmp [ebp+arg_0], esi
jnz 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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

x86_64 ROP

- Chaining multiple function calls via ROP is way easier on 64bit
 - Why?

```
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    [ebp+arg_0], ebx
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
```

x86_64 ROP

- Chaining multiple function calls via ROP is way easier on 64bit
 - Why?
- You simply load function arguments into registers, they don't need to be on the stack!

```
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 [ebp+arg_0]
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
jmp short loc_31308F
```

```
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0
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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

x86_64 ASLR

- **64bit** address space means better **ASLR**
 - ‘better’ simply means more entropy to bruteforce
 - Bruteforcing **ASLR** on **64bit** is rarely done

```
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   esi
mov    [ebp+arg_0], eax
call   sub_31486A
test   eax, eax
jz     short loc_313066
push   esi
lea   eax, [ebp+arg_0]
push   eax
mov    esi, 1D0h
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
```

x86_64 ASLR

```
doom@upwn64:~$ cat /proc/self/maps
(the same segment after multiple runs)
7f638218c000-7f6382347000 r-xp 00000000 08:01 922887
...
```

```
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, 0FFFFFFh
or     eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov    [ebp+var_4], eax
```

x86_64 ASLR

doom@upwn64:~\$ cat /proc/self/maps
(the same segment after multiple runs)

7f638218c000-7f6382347000 r-xp 00000000 08:01 922887

...

7f6fa368e000-7f6fa3849000 r-xp 00000000 08:01 922887

...

```
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
push 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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

x86_64 ASLR

```
doom@upwn64:~$ cat /proc/self/maps  
(the same segment after multiple runs)
```

```
7f638218c000-7f6382347000 r-xp 00000000 08:01 922887  
...  
7f6fa368e000-7f6fa3849000 r-xp 00000000 08:01 922887  
...  
7f974db38000-7f974dcf3000 r-xp 00000000 08:01 922887  
...
```

```
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  
push esi, 1D0h  
push esi  
push [ebp+arg_4]  
push edi  
push 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, 0FFFFFFh  
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

x86_64 ASLR

doom@upwn64:~\$ cat /proc/self/maps
(the same segment after multiple runs)

7f638218c000-7f6382347000 r-xp 00000000 08:01 922887

...

7f6fa368e000-7f6fa3849000 r-xp 00000000 08:01 922887

...

7f974db38000-7f974dcf3000 r-xp 00000000 08:01 922887

...

```
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
push 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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```


x86_64 ASLR

```
doom@upwn64:~$ cat /proc/self/maps  
(the same segment after multiple runs)
```

```
7f638218c000-7f6382347000 r-xp 00000000 08:01 922887
```

```
...
```

```
7f6fa368e000-7f6fa3849000 r-xp 00000000 08:01 922887
```

```
...
```

```
7f974db38000-7f974dcf3000 r-xp 00000000 08:01 922887
```

```
...
```

```
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  
push 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
```

At least 7 nibbles of libc is changing per run on Ubuntu 14.04 x64

7 (nibbles) * 4 (bits) = 28

2^{28} bruteforce

0.00000000037% exploit reliability!

```
loc_313066: ; CODE XREF: sub_312FD8  
push edi  
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, 0FFFFFFh  
or eax, 80070000h  
loc_31308C: ; CODE XREF: sub_312FD8  
mov [ebp+var_4], eax
```

x86_64 Addresses

- **64bit** addresses almost always have a **NULL** upper byte, meaning **ROP chains** and string functions (eg **strncpy**) don't get along

```
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
push   [ebp+arg_1], eax
call   sub_31486A
test   eax, eax
jz     short loc_31306D
lea   esi, [ebp+arg_0]
lea   eax, [ebp+arg_0]
push   eax
mov    eax, [ebp+var_70]
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, 0FFFFFFh
or     eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov    [ebp+var_4], eax
```

x86_64 Addresses

```
doom@upwn64:~$ cat /proc/self/maps
```

```
00400000-0040b000 r-xp 00000000 08:01 790596 /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 790596 /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 790596 /bin/cat
...
7fc6a4788000-7fc6a4943000 r-xp 00000000 08:01 922887 libc-2.19.so
7fc6a4943000-7fc6a4b42000 ---p 001bb000 08:01 922887 libc-2.19.so
7fc6a4b42000-7fc6a4b46000 r--p 001ba000 08:01 922887 libc-2.19.so
7fc6a4b46000-7fc6a4b48000 rw-p 001be000 08:01 922887 libc-2.19.so
...
```

```
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
sub_314486A
test eax, eax
jz short loc_31306D
push esi
mov [ebp+arg_0], esi
push eax
mov esi, 1D0h
push esi
push edi
mov [ebp+arg_0], edi
sub_314486A
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jnz 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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

x86_64 Addresses

```
doom@upwn64:~$ cat /proc/self/maps
```

```
00400000-0040b000 r-xp 00000000 08:01 790596 /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 790596 /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 790596 /bin/cat
...
7fc6a4788000-7fc6a4943000 r-xp 00000000 08:01 922887 libc-2.19.so
7fc6a4943000-7fc6a4b42000 ---p 001bb000 08:01 922887 libc-2.19.so
7fc6a4b42000-7fc6a4b46000 r--p 001ba000 08:01 922887 libc-2.19.so
7fc6a4b46000-7fc6a4b48000 rw-p 001be000 08:01 922887 libc-2.19.so
...
```

```
0x0000000000400000 - 0x000000000040b000
0x00007fc6a4788000 - 0x00007fc6a4943000
```

These are 64bit addresses, so yes there's plenty of space for nulls

x86_64 Syscalls

- The syscall numbers in 32bit vs 64bit Linux are different, so be sure you're looking at the respective table when writing your payloads

```
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+var_10], eax
call    sub_31486A
test    eax, eax
jz      short loc_31306D
lea    eax, [ebp+arg_0]
push    eax
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, 0FFFFFFh
or      eax, 80070000h

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

x86_64 Syscalls

- The syscall numbers in **32bit** vs **64bit Linux** are different, so be sure you're looking at the respective table when writing your payloads

exec syscall on **32bit**: **0x0b**

exec syscall on **64bit**: **0x3b**

```
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+var_70], eax
call sub_31486A
test eax, eax
jz short loc_31306D
lea eax, [ebp+arg_0]
push eax
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 00h
call sub_3141F3
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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Lecture Overview

- Architecture Differences
 - x86
 - x86_64
 - **ARM**
- Platform Differences
 - Windows

```
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
```

ARM Overview

- ARM is a 32bit RISC instruction set built for low power devices
 - Has a '16bit' THUMB mode

The ARM logo is displayed in a large, bold, blue font. It consists of the letters 'ARM' with a registered trademark symbol (®) to the upper right of the 'M'. The logo is centered horizontally and partially overlaps the assembly code on the right side of the slide.

```
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 esi, [ebp+arg_0]
push eax
mov [ebp+var_10], eax
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, 0FFFFFFh
or eax, 80070000h

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


ARM Overview

- **ARM** is a 32bit RISC instruction set built for low power devices
 - Has a '16bit' **THUMB** mode
- Used on your phone, tablet, raspberry pi, other small or mobile devices
 - 'low power'

ARM[®]



ARM Registers

Register
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 'sp'
r14 'lr'
r15 'pc'

Scratch Registers: r0-r3, r12 r0-r3 used to pass parameters r12 intra-procedure scratch will be overwritten by subroutines
Preserved Registers: r4-r11 stack before using restore before returning
Stack Pointer: not much use on the stack
Link Register: set by BL or BLX on entry of routine overwritten by further use of BL or BLX
Program Counter

Register Use in the ARM Procedure Call Standard

```

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+55
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

```

```

; CODE XREF: sub_312FD8
call    sub_3140F3
and     eax, 0FFFFFFh
or      eax, 80070000h
mov     [ebp+var_4], eax

```

ARM Calling Convention

- Calling convention is basically like `fastcall`
 - r0-r3 hold your function arguments

```
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    esi
mov     esi, [ebp+var_70]
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, 0FFFFFFh
or      eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov     [ebp+var_4], eax
```

ARM Assembly

- Some **ARM/THUMB** instructions can operate on multiple registers at once

pop {r4, r5, r6, 1r}

...

```
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_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
```

Instruction Alignment

- **ARM** mode has 4 byte instruction alignment
 - Can't jump in the middle of instructions
- **THUMB** mode has 2 byte instruction alignment
 - When **ROPing** there's usually more **THUMB** gadgets that will be of use due to the 2 byte alignment

```
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+var_70], 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
sub [ebp+var_70], eax
cmp [ebp+var_70], esi
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31306D
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push esi
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
```

An Interesting Bit

- Because of 2 & 4 byte instruction alignment, the lowest bit of the program counter (eg r15) will never be set

0x080462B0

00001000000001000110001010110000

```
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_31486A
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
cmp [ebp+arg_0], esi
jz short loc_31308F
```

```
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
```

```
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
```

```
loc_31307D: ; CODE XREF: sub_312FD8
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

An Interesting Bit

- Because of 2 & 4 byte instruction alignment, the lowest bit of the program counter (eg r15) will never be set

0x080462B0

000010000000010001100010110000

This bit is re-purposed to tell the processor if we are in THUMB mode or ARM mode

```
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_31486A
test eax, eax
sub eax, [ebp+var_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
cmp [ebp+arg_0], esi
jz short loc_31308F
```

```
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
```

```
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
```

```
call sub_3140F3
test eax, eax
sub eax, [ebp+var_31306D]
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
```

An Interesting Bit

r15 = 0x080462B0

= 000010000000001000110001010110000

Interpret bytes at 0x080462B0 as ARM

r15 = 0x080462B1

= 000010000000001000110001010110001

Interpret bytes at 0x080462B0 as THUMB

```
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
mov [ebp+arg_0], eax
push eax
mov esi, 1D0h
push esi
push [ebp+var_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
jnz 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
```


Caching

- In **x86** the processor will invalidate icache lines if the line is written to

```
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
```

Caching

- In **x86** the processor will invalidate icache lines if the line is written to
- With **ARM** you have to request manual cache flushes, or do large memory operations to flush the cache naturally

```
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
jnz short loc_313066
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
```

Caching

- In **x86** the processor will invalidate icache lines if the line is written to
- With **ARM** you have to request manual cache flushes, or do large memory operations to flush the cache naturally
 - Can get annoying in exploitation
 - ‘what you seez, may not beez what it iz’

```
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
jnz short loc_313066
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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Lecture Overview

- Architecture Differences
 - x86
 - x86_64
 - ARM
- Platform Differences
 - Windows

```
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
```

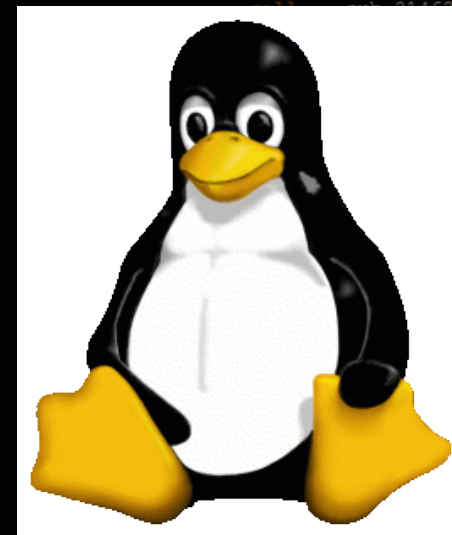
```
call sub_3140F3
and eax, 0FFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

Windows vs Linux

- Almost all the vulnerability classes and exploitation techniques you have learned in this course will apply directly to **Windows**



```
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
lea eax, [ebp+arg_0]
push esi
push esi
push [ebp+arg_4]
push edi
```

```
31306D
], esi
31308F
; CODE XREF: sub_312FD8
; sub_312FD8+55
; CODE XREF: sub_312FD8
; sub_312FD8+49
31307D
31308C
loc_31307D:
; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Windows Basics

- The executable format on Windows is obviously **.EXE's** instead of **Linux ELF'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
mov     [ebp+arg_0], eax
call    sub_31486A
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
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, 0FFFFFFh
or      eax, 80070000h
```

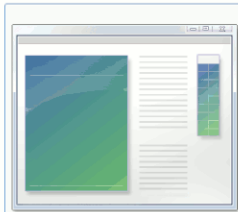
```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov     [ebp+var_4], eax
```

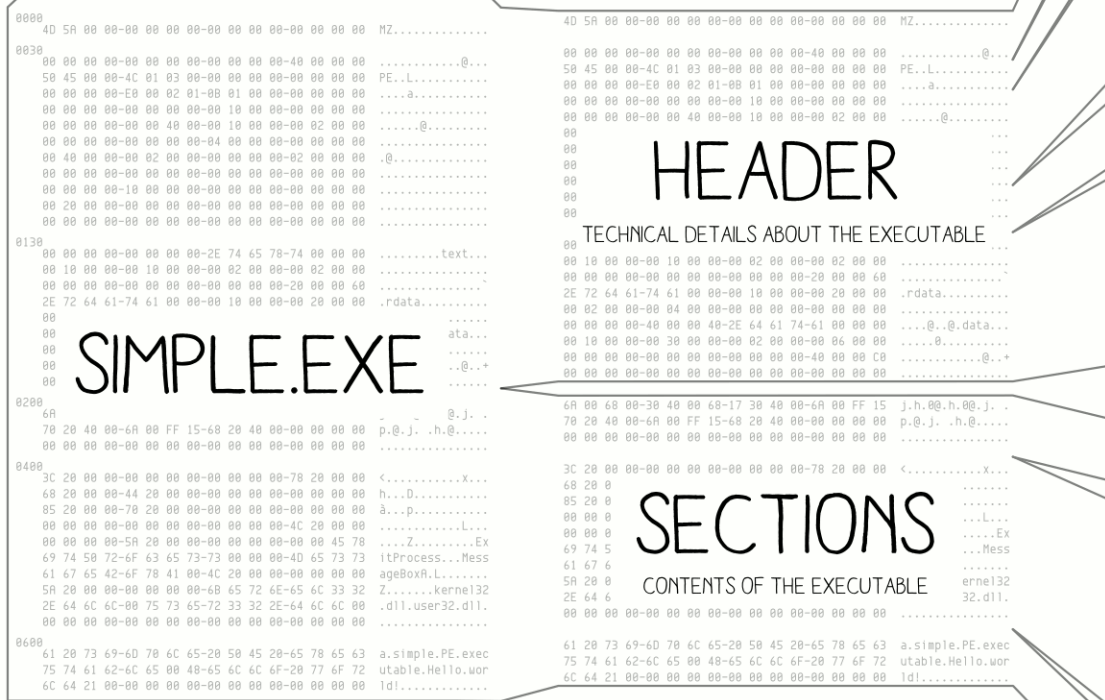
```

push edi
call sub_314623
test eax, eax

```



simple



SIMPLE.EXE

HEADER

TECHNICAL DETAILS ABOUT THE EXECUTABLE

SECTIONS

CONTENTS OF THE EXECUTABLE

DOS HEADER

4D 5A 00 00-00 00 00 MZ.....

PE HEADER

50 45 00 00-4C 01 03 PE..L.....

OPTIONAL HEADER

01-00 01 00 00-00 00 00 00
00 00 00 00-00 00 00 00-00 10 00 00-00 00 00 00
00 00 00 00-00@.....
00 40 00 00-00 EXECUTABLE INFORMATION
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00 00 00 00-10 00 00 00

DATA DIRECTORIES

00 20 00 00- POINTERS TO EXTRA STRUCTURES (EXPORTS, IMPORTS,...)

SECTIONS TABLE

2E 72 64 61-74 61 00 00-00 10 00 00-00 20 00 60
00 00 00 00-40 DEFINES HOW THE FILE IS LOADED IN MEMORY
00 10 00 00-00 30 00 00-00 02 00 00-00 06 00 00
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00 C0
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00

CODE

6A 00 68 00-30 40 00 68- FF 15 j.h.@.h.@.j. .12FD8
70 20 40 00-6A 00 FF 15- p.@.j. .h.@.....
00 00 00 00-00 00 00 00-00 00 00 00 00

IMPORTS

3C 20 00 00-00 00 00 00-00 00 00 00-78 20 00 00 <.....X...
68 20 00 00-44 20 00 00-00 00 00 00-00 00 00 00 h...D.....
85 20 00 00-70 20 00 00-00 00 00 00-00 00 00 00 a...p.....12FD8
00 00 00 0
69 74 50 7 LINK BETWEEN THE EXECUTABLE AND (WINDOWS) LIBRARIES ess...Mess
61 67 65 42-6F 78 41 00-4C 20 00 00-00 00 00 00 00 ageBoxA.L.....
5A 20 00 00-00 00 00 00-68 65 72 6E-65 6C 33 32 Z.....kernel32
2E 64 6C 6C-00 75 73 65-72 33 32 2E-64 6C 6C 00 .dll.user32.dll.

DATA

61 20 73 69-6D 70 6C 65-20 50 45 20-65 78 65 63 a.simple.PE.exec
75 74 61 62-6C 65 00 48-65 6C 6C 6F-20 77 6F 72 utable.Hello.wor
6C 64 21 00-00 00 00 00-00 00 00 00-00 00 00 00 ld!.....12FD8

```

call sub_3140F3

```

```

and eax, 0FFFFFFFh
or eax, 80070000h

```

```

loc_31308C:
mov [ebp+var 4], eax

```

Windows Basics

- The executable format on Windows is obviously **.EXE's** instead of **Linux ELF's**
- Libraries are **.DLL's**, like **Linux .so's**
 - eg: **MSVCRT.dll** is like **libc**
 - Microsoft Visual C(++) Common Runtime

```
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
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
```

```
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
```


Windows Basics

Loaded DLL's

Dependency Walker - [Regshot-x86-Unicode.exe]

File Edit View Options Profile Window Help

REGSHOT-X86-UNICODE.EXE

- KERNEL32.DLL
- USER32.DLL
- COMDLG32.DLL
- ADVAPI32.DLL
- SHELL32.DLL

PI	Ordinal ^	Hint	Function
C	N/A	68 (0x0044)	CloseHandle
C	N/A	82 (0x0052)	CompareFileTime
C	N/A	121 (0x0079)	CreateFileA
C	N/A	128 (0x0080)	CreateFileW
C	N/A	191 (0x00BF)	DeleteCriticalSection
C	N/A	218 (0x00DA)	EnterCriticalSection
C	N/A	261 (0x0105)	ExitProcess
C	N/A	282 (0x011A)	FindClose
C	N/A	293 (0x0125)	FindFirstFileW
C	N/A	305 (0x0131)	FindNextFileW
C	N/A	322 (0x0142)	FlushFileBuffers
C	N/A	332 (0x014C)	FreeEnvironmentStringsW
C	N/A	339 (0x0153)	GetACP
C	N/A	348 (0x015C)	GetCPInfo
C	N/A	369 (0x0171)	GetCommandLineW

```
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]
```

```
loc_31307D:
call sub_3140F3
and eax, 0FFFFh
or eax, 80070000h

loc_31308C:
mov [ebp+var_4], eax
```

Windows Basics

- The executable format on Windows is obviously **.EXE's** instead of **Linux ELF's**

- Libraries are **.DLL's**, like **Linux .so's**

- eg: **MSVCRT.dll** is like **libc**

- Microsoft Visual C(++) Common Runtime

- A process usually loads lots of libs (dll'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
mov [ebp+arg_0], eax
call sub_31486A
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
cmp [ebp+arg_0], esi
jz short loc_31308F
```

```
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
```

```
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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

Windows Debuggers

- If you're going to get rolling on Windows, try to pick up skills debugging with WinDbg EARLY

```
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+var_0], eax
call    sub_31486A
test    eax, eax
jnz     short loc_31306E
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
```

Windows Debuggers

- If you're going to get rolling on Windows, try to pick up skills debugging with WinDbg EARLY
- WinDBG is Microsoft's debugger
 - Basically GDB with different command mappings
 - Not as convenient as OllyDBG, but way less sketchy
 - Best 64bit debugger

```
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+var_0], eax
call sub_31486A
test eax, eax
jnz short loc_31306F
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+55
push 0Dh
call sub_314623
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, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

WinDbg

```

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

```

Kernel 'com:port=\\.\pipe\com_1_pipe' - WinDbg:6.8.0004.0

File Edit View Debug Window Help

Command - Kernel 'com:port=\\.\pipe\com_1_pipe' - WinDbg:6.8.0004.0

```

kd> !drvobj atapi 2 2
Driver object (817d2a68) is for:
  \Driver\atapi
DriverEntry: f99b7974 atapi!DriverEntry
DriverStartIo: f99ac02e atapi!IdePortStartIo
DriverUnload: f99b4c5a atapi!IdePortUnload
AddDevice: f99b2fac atapi!ChannelAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE f99ae86c atapi!IdePortAlwaysStatusSuccessIrp
[01] IRP_MJ_CREATE_NAMED_PIPE 8050086f nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE f99ae86c atapi!IdePortAlwaysStatusSuccessIrp
[03] IRP_MJ_READ 8050086f nt!IopInvalidDeviceRequest
[04] IRP_MJ_WRITE 8050086f nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION 8050086f nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION 8050086f nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA 8050086f nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA 8050086f nt!IopInvalidDeviceRequest
[09] IRP_MJ_FLUSH_BUFFERS 8050086f nt!IopInvalidDeviceRequest
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION 8050086f nt!IopInvalidDeviceRequest
[0b] IRP_MJ_SET_VOLUME_INFORMATION 8050086f nt!IopInvalidDeviceRequest
[0c] IRP_MJ_DIRECTORY_CONTROL 8050086f nt!IopInvalidDeviceRequest
[0d] IRP_MJ_FILE_SYSTEM_CONTROL 8050086f nt!IopInvalidDeviceRequest
[0e] IRP_MJ_DEVICE_CONTROL 8050086f nt!IopInvalidDeviceRequest

```

Memory

Virtual	Symbol
817d2a68	00a80004
817d2a6c	81791d98
817d2a70	00000012
817d2a74	f99a5000 atapi!InitDeviceGeome
817d2a78	00015380
817d2a7c	817f3c28
817d2a80	817d2b10
817d2a84	001a001a
817d2a88	e131bd00
817d2a8c	806700c8 nt!CmRegistryMachineH
817d2a90	00000000
817d2a94	f99b7974 atapi!DriverEntry
817d2a98	f99ac02e atapi!IdePortStartIo
817d2a9c	f99b4c5a atapi!IdePortUnload
817d2aa0	f99ae86c atapi!IdePortAlwaysSt
817d2aa4	8050086f nt!IopInvalidDeviceRe
817d2aa8	f99ae86c atapi!IdePortAlwaysSt
817d2aac	8050086f nt!IopInvalidDeviceRe
817d2ab0	8050086f nt!IopInvalidDeviceRe
817d2ab4	8050086f nt!IopInvalidDeviceRe
817d2ab8	8050086f nt!IopInvalidDeviceRe
817d2abc	8050086f nt!IopInvalidDeviceRe
817d2ac0	8050086f nt!IopInvalidDeviceRe

Disassembly - Kernel 'com:port=\\.\pipe\com_1_pipe' - WinDbg:6.8.0004.0

Offset	Disassembly
f99ac025	33c0 xor eax, eax
f99ac027	5b pop ebx
f99ac028	5f pop edi
f99ac029	5e pop esi
f99ac02a	c9 leave
f99ac02b	c20c00 ret 0Ch
f99ac02e	55 atapi!IdePortStartIo:
f99ac02f	8bec push ebp
f99ac031	51 mov ebp, esp
f99ac032	51 push ecx
f99ac033	8b4d08 mov ecx, dword ptr [ebp+8]
f99ac036	8b450c mov eax, dword ptr [ebp+0Ch]

Memory

Virtual	Display format	Value
0x`ffffffff80695759	Byte	80695759 00 53 00 79 00 73 00 74 00 65 00 6d .S.y.s.t.e.m
		80695765 00 5c 00 4c 00 61 00 73 00 74 00 4b .\L.a.s.t.K
		80695771 00 6e 00 6f 00 77 00 6e 00 47 00 6f .n.o.w.n.G.o
		8069577d 00 6f 00 64 00 52 00 65 00 63 00 6f .o.d.R.e.c.o
		80695789 00 76 00 65 00 72 00 79 00 5c 00 4c .v.e.r.y.\L
		80695795 00 61 00 73 00 74 00 47 00 6f 00 6f .a.s.t.G.o.o
		806957a1 00 64 00 00 00 5c 00 53 00 79 00 73 .d.\.S.y.s
		806957ad 00 74 00 65 00 6d 00 52 00 6f 00 6f .t.e.m.R.o.o
		806957b9 00 74 00 5c 00 4c 00 61 00 73 00 74 .\L.a.s.t
		806957c5 00 47 00 6f 00 6f 00 64 00 2e 00 54 .G.o.o.d...T
		806957d1 00 6d 00 70 00 00 00 00 00 00 00 .m.p.....

Registers - Kernel 'com:port=\\.\pipe\com_1_pipe' - WinDbg:6.8.0004.0

Reg	Value
gs	0
fs	30
es	23
ds	23
edi	1502b92
esi	0
ebx	fffff980
edx	3f8
ecx	8054af9c
eax	1
ebp	f9ede178
eip	805183fa
cs	8
efl	202
esp	f9ede168
ss	10
dr0	0
dr1	0
dr2	0
dr3	0
dr6	ffff0ff0

Memory

Virtual	Display format	Value
817d2a68	Pointer and Symbol	00a80004

Disassembly - Kernel 'com:port=\\.\pipe\com_1_pipe' - WinDbg:6.8.0004.0

Offset	Disassembly
loc_31308C	and eax, 0ffffh
loc_31308C	or eax, 80070000h
loc_31308C	mov [ebp+var_4], eax

Registers - Kernel 'com:port=\\.\pipe\com_1_pipe' - WinDbg:6.8.0004.0

Reg	Value
eax	80070000h

MBE - 05/08/2015

x64, ARM, Windows

69

Windows Exploitation Basics

- Raw syscalls are virtually never seen in native windows applications or libraries

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

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

Windows Exploitation Basics

- Raw syscalls are virtually never seen in native windows applications or libraries
 - No more ``int 0x80`` shellcode

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
lea 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
lea [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, 0FFFFFFh
or eax, 80070000h

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

Windows Exploitation Basics

- Raw syscalls are virtually never seen in native windows applications or libraries
 - No more `int 0x80` shellcode
 - Why?

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

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```


Windows Exploitation Basics

- Raw syscalls are virtually never seen in native windows applications or libraries
 - No more `int 0x80` shellcode
 - Why?
- Syscall numbers tend to change from version to version of Windows and would be hard or unreliable to code into an exploit

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
lea 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
lea [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+55
; sub_312FD8+55
```

```
push 0h
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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

ntdll.dll and kernel32.dll

- ntdll.dll – the ‘Native API’
 - Wraps all the syscalls for the given version of Windows, is pretty low level stuff

- kernel32.dll – the ‘Win32 API’

– More familiar high level stuff

- **OpenFile(), ReadFile(), CreateProcess(), LoadLibrary(), GetProcAddress(),**

```
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
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
```

```
call sub_3140F3 ; CODE XREF: sub_312FD8
; sub_312FD8+49
```

```
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
```

Windows Fun Facts

- Most people think `kernel32.dll` is required by every windows process, but `ntdll.dll` is infact the only one that **MUST** be loaded

```
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_4], eax
call    sub_31486A
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
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
```

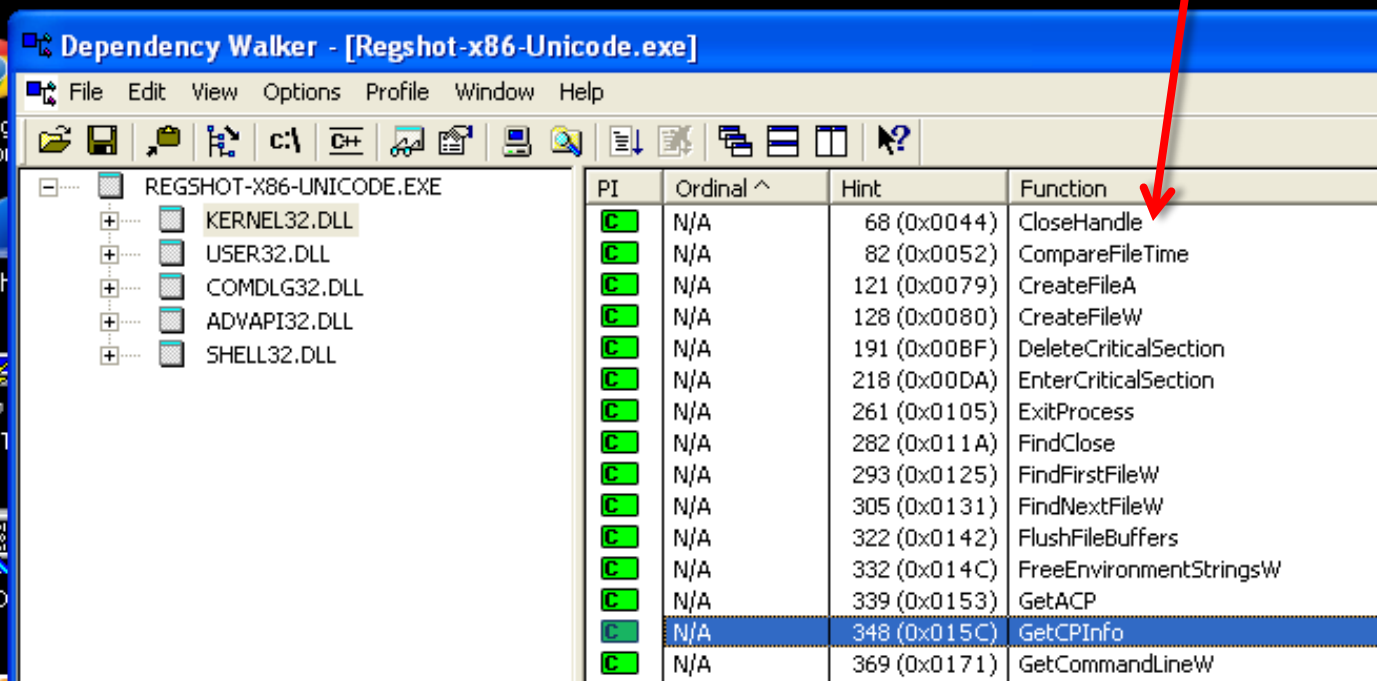
```
call    sub_3140F3
and     eax, 0FFFFh
or      eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov     [ebp+var_4], eax
```

Windows Exploitation Basics

- So instead of using syscalls, an exploit will almost always use existing **imported functions**



Windows Exploitation Basics

- If a function of interest is not imported by a loaded **DLL**, an exploit payload will usually do what is known as **‘walking the IAT’**
 - It resolves the function location manually

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
lea 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_31486A
call sub_31486A
test eax, eax
jnz short loc_313066
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
```

Windows Exploitation Basics

- If a function of interest is not imported by a loaded **DLL**, an exploit payload will usually do what is known as **‘walking the IAT’**
 - It resolves the function location manually
- If **GetProcAddress()** is imported from **kernel32.dll**, you can easily lookup functions
 - Same as **dlsym()** on **Linux**

```
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_31486A
test eax, eax
jnz short loc_313066
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
```

```
loc_313066: ; CODE XREF: sub_312FD8 ; sub_312FD8+55
```

```
push 1Dh
call sub_31411B
```

```
loc_31306E: ; 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
```

Windows Exploitation Basics

GetProcAddress(k32h, "CreateProcess");

...

Looking up the CreateProcess function

```
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_1]
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 1Dh
call sub_3140F3
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, 0FFFFFFh
or eax, 80070000h
```

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

Windows XP Security

- Windows XP SP2 marked the start of the modern security era (Summer 2004)

```
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
```

Protection – Bypass – ???

```
loc_31307D:                                     ; CODE XREF: sub_312FD8
```

```
call    sub_3140F3
and     eax, 0FFFFFFh
or      eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov     [ebp+var_4], eax
```


Windows XP Security

- Windows XP SP2 marked the start of the modern security era (Summer 2004)
 - Hardware Enforced DEP – ROP

```
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
```

Protection – Bypass – ???

```
loc_31307D:                                     ; CODE XREF: sub_312FD8
```

```
call    sub_3140F3
and     eax, 0FFFFFFh
or      eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov     [ebp+var_4], eax
```

Windows XP Security

- Windows XP SP2 marked the start of the modern security era (Summer 2004)
 - Hardware Enforced DEP – ROP
 - Stack Cookies (GS) – Leak & replace, write past, SEH

```
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_1], 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
```

Protection – Bypass – ???

```
loc_31307D:                                     ; CODE XREF: sub_312FD8
```

```
call   sub_3140F3
and    eax, 0FFFFFFh
or     eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov    [ebp+var_4], eax
```

Windows XP Security

- Windows XP SP2 marked the start of the modern security era (Summer 2004)
 - Hardware Enforced DEP – ROP
 - Stack Cookies (GS) – Leak & replace, write past, SEH
 - Safe heap unlinking – Heap metadata exploits

```
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
```

```
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
```

Protection – Bypass – ???

```
loc_31307D: ; CODE XREF: sub_312FD8
```

```
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

Windows XP Security

- Windows XP SP2 marked the start of the modern security era (Summer 2004)
 - Hardware Enforced DEP – ROP
 - Stack Cookies (GS) – Leak & replace, write past, SEH
 - Safe heap unlinking – Heap metadata exploits
 - SafeSEH – ?

```
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
jnz short loc_313066
```

```
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, 0FFFFFFh
or eax, 80070000h
```

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

Protection – Bypass – ???

Windows XP Security

- Windows XP SP2 marked the start of the modern security era (Summer 2004)
 - Hardware Enforced DEP – ROP
 - Stack Cookies (GS) – Leak & replace, write past, SEH
 - Safe heap unlinking – Heap metadata exploits
 - SafeSEH – ? What is SEH/SafeSEH ?

Protection – Bypass – ???

```
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
```

```
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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

Structured Exception Handling

- Structured Exception Handling is a lot like assigning signal handlers on Linux

```
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]
jg     eax, [ebp+var_84]
jb     short loc_313066
sub     eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    esi
mov     [ebp+var_4], 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
```

Structured Exception Handling

- Structured Exception Handling is a lot like assigning signal handlers on Linux
- You simply register an exception handler, and if something bad like a segfault happens, code flow is redirected to the handler
 - Print an error message, exit semi-gracefully, etc...

```
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], ebx
mov     eax, [ebp+var_70]
jnb     short loc_313066
sub     eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    esi
mov     [ebp+var_84], 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
; CODE XREF: sub_312FD8
; sub_312FD8+55
push    0Dh
call    sub_31411B

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, 0FFFFFFh
or      eax, 80070000h

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

Exploiting SEH

- Exception records are placed on the **stack**, so they're relatively easy to corrupt

0012BF1C	0012C1C8	!+⊕. Pointer to next SEH record
0012BF20	7C90E900	.0E! SE handler
0012BF24	7C91C348	H æ! ntdll.7C91C348
0012BF28	00000002	0...
0012BF2C	0012C1D8	+!⊕.
0012BF30	7C916351	0cæ! RETURN to ntdll.7C916351 from ntdll.7C91C2D5
0012BF34	00000000
0012BF38	C0150008	0.3^
0012BF3C	0012CE8C	{}⊕.
0012BF40	00000000
0012BF44	0012BF9C	æ^⊕.
0012BF48	00252A60	'*%.
0012BF4C	001310A0	â>!! ASCII "S=Hd,"
0012BF50	0012BFA4	æ^⊕. UNICODE "\\\.\C2CAD972#4079#4fd3#A68D#AD34CC121074\N\max++.00.w86"
0012BF54	0012BFA0	æ^⊕.
0012BF58	7FFDF000	.E^â
0012BF5C	00252BE8	æ+%.
0012BF60	0012BFF8	"^⊕. UNICODE "\\\.\C2CAD972#4079#4fd3#A68D#AD34CC121074\N\max++.00.w86"

```
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
push [ebp+var_10]
call sub_31486A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
```

XREF: sub_312FD8+55
XREF: sub_312FD8+49

```
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C

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

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


Exploiting SEH

- Because you only have one **gadget** of execution through an overwritten **SEH record**, you usually have to use it to **stack pivot**

```
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    [ebp+var_04], esi
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, 0FFFFFFh
or     eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov    [ebp+var_4], eax
```

Exploiting SEH

- Because you only have one **gadget** of execution through an overwritten **SEH record**, you usually have to use it to **stack pivot**
- Classically you could use a **'pop pop ret'** **gadget** to easily return onto the smashed stack (**assumes executable stack**) as a pointer to your overwritten **SEH record** is nearby

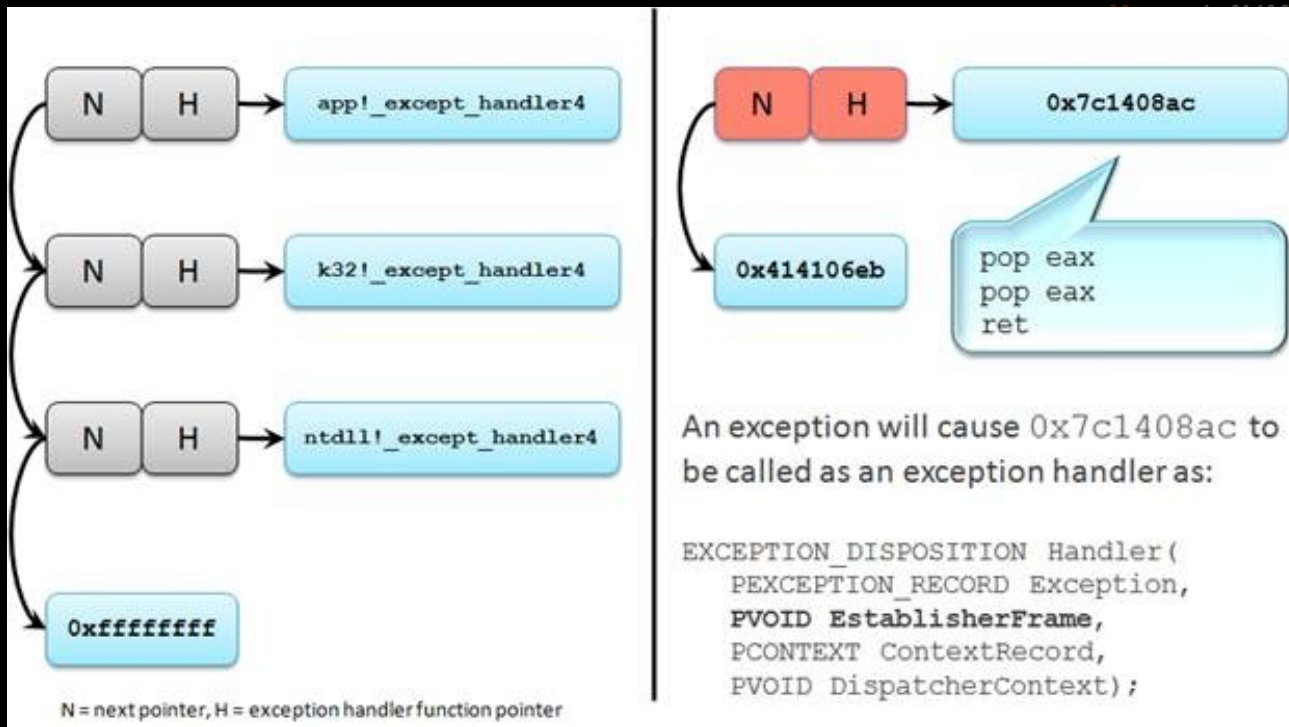
```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jb short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
push edi
mov [ebp+arg_0], eax
call sub_31486A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov [ebp+var_84], ebx
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jnz short loc_31308F
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
call sub_31411B
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jnz short loc_31307D
call sub_3140F3
jmp short loc_31308C
loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Exploiting SEH

```

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

```



An exception will cause 0x7c1408ac to be called as an exception handler as:

```

pop eax
pop eax
ret

```

```

31306D
arg_0]
31306D
], esi
31308F
; CODE XREF: sub_312FD8
; sub_312FD8+55
31306D
; CODE XREF: sub_312FD8
; sub_312FD8+49
31307D
31307D
call sub_3140F3
jmp short loc_31308C
; -----
loc_31307D:
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
; CODE XREF: sub_312FD8

```

SafeSEH

- **SafeSEH** is an additional set of checks made to ensure that a registered exception handler has not been corrupted
- You can enable it using the `/SAFESEH` flag at compile time

```
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_31486A
test   eax, eax
jz     short loc_313066
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
jnz   short loc_313066
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
```

Bypassing SafeSEH

- With **SafeSEH**, an exception record is invalid if:
 - The exception handler is pointing onto the stack
 - The exception handler does not match the list of registered exception handlers in module it is pointing into

```
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
lea   eax, [ebp+arg_0]
push  eax
mov    eax, 1D0h
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
```

Windows Vista Security

- Windows Vista was marred by instability and performance issues, but made good progress in terms of security

```
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 [ebp+arg_0]
call sub_314866
test eax, eax
jz short loc_313066
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, 0FFFFFFh
or eax, 80070000h

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

Protection – Bypass – ???

Windows Vista Security

- Windows Vista was marred by instability and performance issues, but made good progress in terms of security
 - ASLR – Info leaks, partial overwrites, non aslr'd code

```
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 [ebp+arg_0]
call sub_314866
test eax, eax
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_3140F3
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, 0FFFFFFh
or eax, 80070000h

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

Protection – Bypass – ???

Windows Vista Security

- Windows Vista was marred by instability and performance issues, but made good progress in terms of security
 - ASLR – Info leaks, partial overwrites, non aslr'd code
 - SEHOP – ?

```
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 [ebp+arg_0]
call sub_314866
test eax, eax
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_3140F3
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
```

Protection – Bypass – ???

```
loc_31307D: ; CODE XREF: sub_312FD8
```

```
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```


SEH Overwrite Protection

- SEH Overwrite Protection (SEHOP) is the second attempt Microsoft made to mitigate SEH 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
mov     [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jnz     short loc_313066
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
```

SEH Overwrite Protection

- SEH Overwrite Protection (SEHOP) is the second attempt Microsoft made to mitigate SEH exploitation
- When an exception is triggered, the SEH dispatcher attempts to walk the SEH chain to a symbolic 'terminating' record
 - If this record cannot be reached, the chain is bad

```
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
jnz short loc_313066
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
loc_31308F: jmp short loc_31308F
```

```
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0
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, 0FFFFFFh
or eax, 80070000h
```

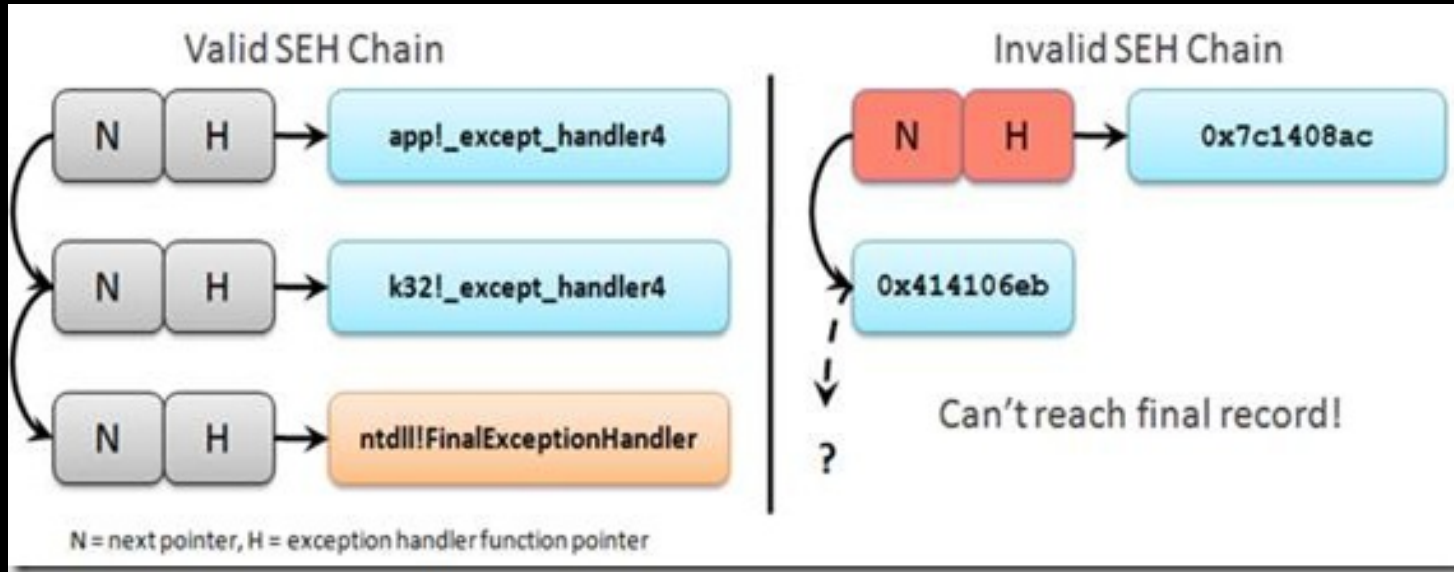
```
loc_31308C: ; CODE XREF: sub_312FD8
```

SEH Overwrite Protection

```

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

```



```

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
```

Bypassing SEHOP

- Bypassing **SEHOP** is pretty painful and basically involves faking a chain to the terminating record

```
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, 0FFFFFFh
or     eax, 80070000h
```

```
loc_31308C:                                     ; CODE XREF: sub_312FD8
```

```
mov    [ebp+var_4], eax
```

Windows Vista Security

- Windows Vista was marred by instability and performance issues, but made good progress in terms of security
 - ASLR – Info leaks, partial overwrites, non aslr'd code
 - SEHOP – Faking SEH Chains
 - Heap Hardening – More heap metadata checks

Protection – Bypass – ???

```
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 [ebp+arg_0]
call sub_314864
test eax, eax
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_3140F3
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
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, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8
```

```
mov [ebp+var_4], eax
```

Windows 7 Security

- I don't think much new stuff happened with **Windows 7** in terms mitigation technologies
- Mostly cleaning up stability issues from Vista

```
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_31486A
call    sub_31486A
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
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
```

Windows 8 Security

- Windows 8/8.1 took a big step forward in security
 - Enhanced GS (Stack Cookies)
 - VTGuard – Like a Vtable Canary
 - Heap Hardening
 - Allocation order randomization – Non-deterministic alloc. order
 - Guard pages – A bit like canaries between heap pages
 - ASLR Entropy Improvements – More entropy all around
 - PatchGuard – Prevent the kernel from being live patched
 - Secure Boot – Eliminate root/boot kits with chain of trust
 - Control Flow Guard – Whitelist indirect calls

```
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 edi
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_313066
cmp [ebp+arg_0], esi
jg short loc_313066
```

```
loc_313066: ; CODE XREF: sub_312FD8+113
push esi
call sub_31411B
loc_31306A: ; CODE XREF: sub_312FD8+117
call sub_3140F3
; sub_312FD8+49
```

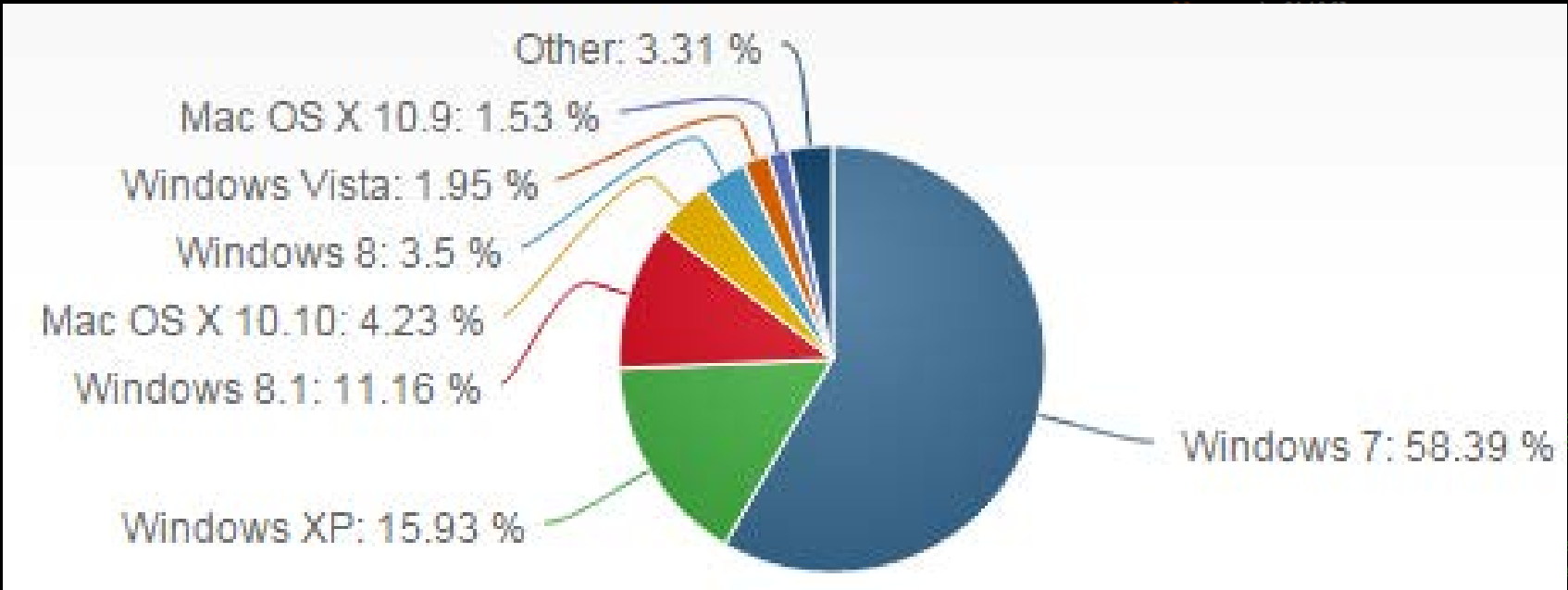
```
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
cmp [ebp+arg_0], loc_31308C
```

```
loc_31307D: ; CODE XREF: sub_312FD8+11F
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
```

```
loc_31308C: ; CODE XREF: sub_312FD8+123
mov [ebp+var_4], eax
```

Desktop Market Share, May 2015

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
short loc_313086
eax [ebp+var_70]
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
```



Windows market share, **~90.93%**

```
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, 0FFFFFFh
or eax, 80070000h

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


Windows Summary

- In the end, **Windows** based exploitation isn't too different from **Linux**, but it's quickly getting harder

```
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
call sub_31486A
test eax, eax
jnz 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
```

Windows Summary

- In the end, **Windows** based exploitation isn't too different from **Linux**, but it's quickly getting harder
- **Some main takeaways**
 - Differing 64bit calling convention
 - Syscalls aren't really a thing on Windows
 - New class of vulnerabilities, SEH Exploitation
 - New protections, SafeSEH, SEHOP
 - Better ASLR & Heap internals
 - Its mitigation technologies are rapidly evolving

```
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
call sub_31486A
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:
call sub_3140F3
push esi
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, 0FFFFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
```