

Linux Kernel Exploitation

Modern Binary Exploitation

CSCI 4968 - Spring 2015

Patrick Biernat

```
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    esi
mov    esi, 0Dh
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
```

; CODE XREF: sub_312FD8
; sub_312FD8+59

```
push    0Dh
call    sub_31411B
```

; CODE XREF: sub_312FD8
; sub_312FD8+49

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

```
-----
```

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

; CODE XREF: sub_312FD8

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

; CODE XREF: sub_312FD8

First order of Business

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

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

loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; ----- ; CODE XREF: sub_312FD8
loc_31307D:                                ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h

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

First order of Business

**FILES IN A FOLDER CALLED
TEMPORARY**

ARE TEMPORARY

```
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
```

You probably feel like this

**TELLS YOU TO PUT YOUR
STUFF IN /TMP**

MBE TA's



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

DELETES /TMP

Lecture Overview

1. An Introduction to the Kernel
2. General Exploitation Strategy
3. Kernel-Space Protections
4. Example
5. Conclusion

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

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

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

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

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

Jumping out of the Matrix

So far, we have been exploiting binaries running in userspace.

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

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

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

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

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

Jumping out of the Matrix

So far, we have been exploiting binaries running in userspace.

Userspace is an *abstraction* that runs “on top” of the kernel.

```
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]
or    eax, [ebp+var_84]
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov    [ebp+arg_0], eax
sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
sub_31486B
push    esi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Jumping out of the Matrix

So far, we have been exploiting binaries running in userspace.

Userspace is an *abstraction* that runs “on top” of the kernel.

1. Filesystem I/O
2. Privilege Levels (Per User/Per Group)
3. Syscalls
4. Processes
5. And so much more

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jnz   short loc_313066
je    eax, [ebp+var_70]
je    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    sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
sub    [ebp+arg_0]
push    esi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313066:
push    0Dh
call    sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31306D:
call    sub_3140F3
test    eax, eax
jg    short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8
```

Jumping out of the Matrix

So far, we have been exploiting binaries running in userspace.

Userspace is an *abstraction* that runs “on top” of the kernel.

1. Filesystem I/O
2. Privilege Levels (Per User/Per Group)
3. Syscalls
4. Processes
5. And so much more

These are all “services” provided by the Kernel

```
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]
or    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov    [ebp+arg_0], eax
sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
sub_31486B
push    esi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313066:
push    0Dh
call    sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31306D:
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; -----
call    sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8
```

What's a Kernel?

Low Level code with two major responsibilities

1. Interact with and control hardware components
2. Provide an Environment in which Applications can run

```
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    sub_31462A
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
esi, 1D0h
esi
push    [ebp+arg_4]
sub    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

What's a Kernel?

Low Level code with two major responsibilities

1. Interact with and control hardware components
2. Provide an Environment in which Applications can run

The Kernel is the core of the operating system

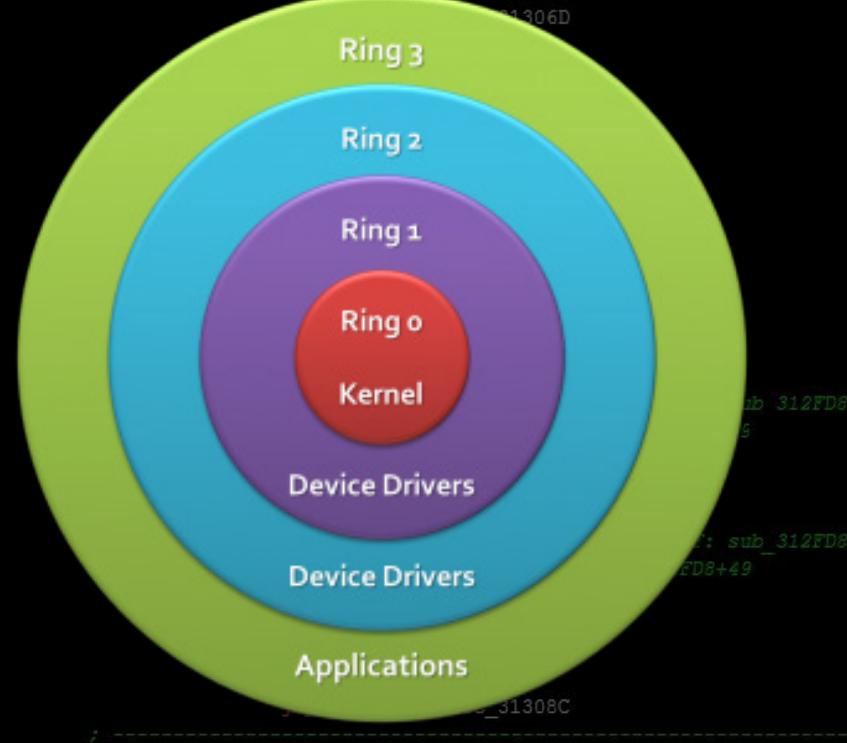
```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jnZ    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov    [ebp+arg_0], eax
call    sub_31462A
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]
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
loc_313066:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
push    0Dh
call    sub_31411B
; -----                                         ; 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
```

What's a Kernel? - Ring Model

Hardware Enforced Model

0: Privileged, Kernelspace

3: Restricted, Userspace



```
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
```

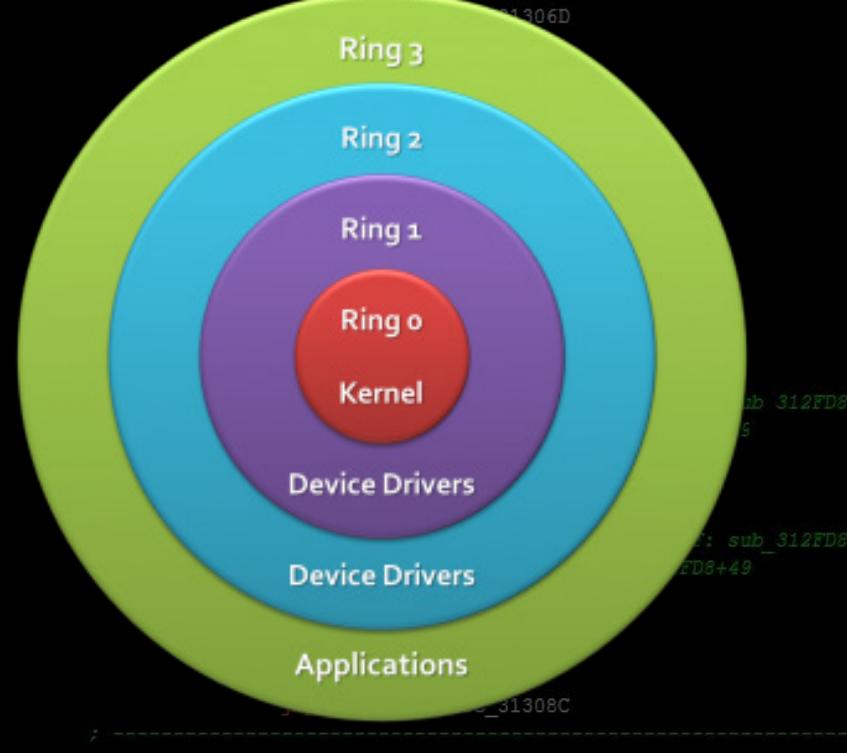
What's a Kernel? - Ring Model

Hardware Enforced Model

0: Privileged, Kernelspace

3: Restricted, Userspace

Ring 1 and Ring 2 are not utilized by most popular/modern Operating Systems (Linux / Windows / OSX)

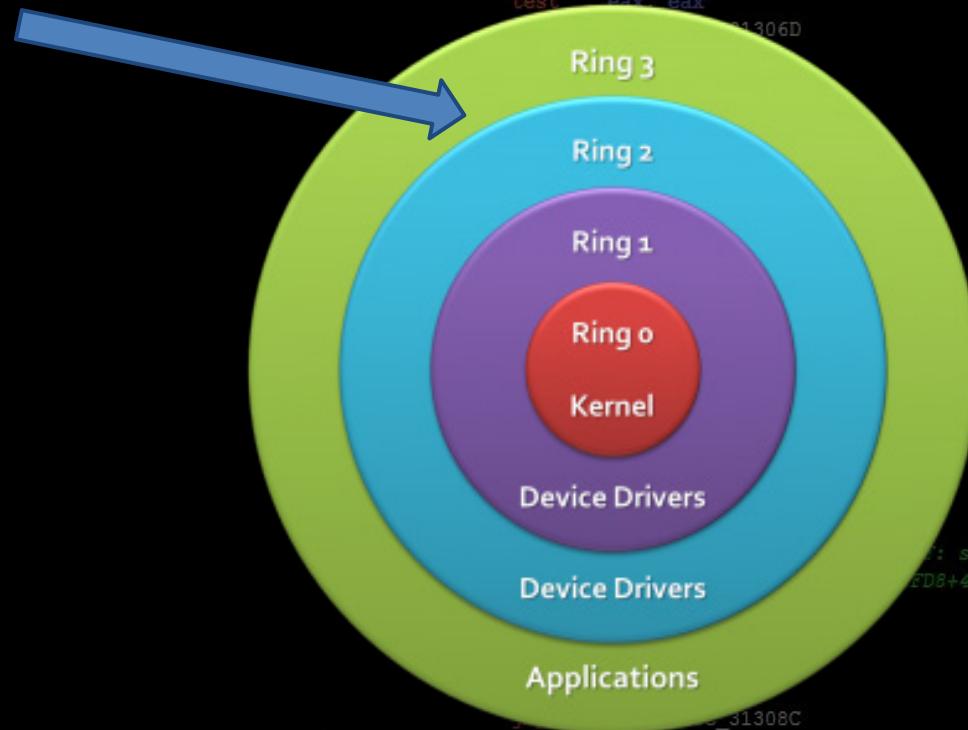


```
loc_31307D: ; CODE XREF: sub_312FD8
    call    sub_3140F3
    and    eax, 0FFFFh
    or     eax, 80070000h

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

What's a Kernel? - Ring Model

We've Been Here



```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_01], ebx
jnz    short loc_313066
or     eax, [var_70]
cmp    [ebp+var_84], ebx
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
```

```
; CODE XREF: sub_31306D
```

```
Ring 2
```

```
Ring 1
```

```
Ring 0
```

```
Kernel
```

```
Device Drivers
```

```
Device Drivers
```

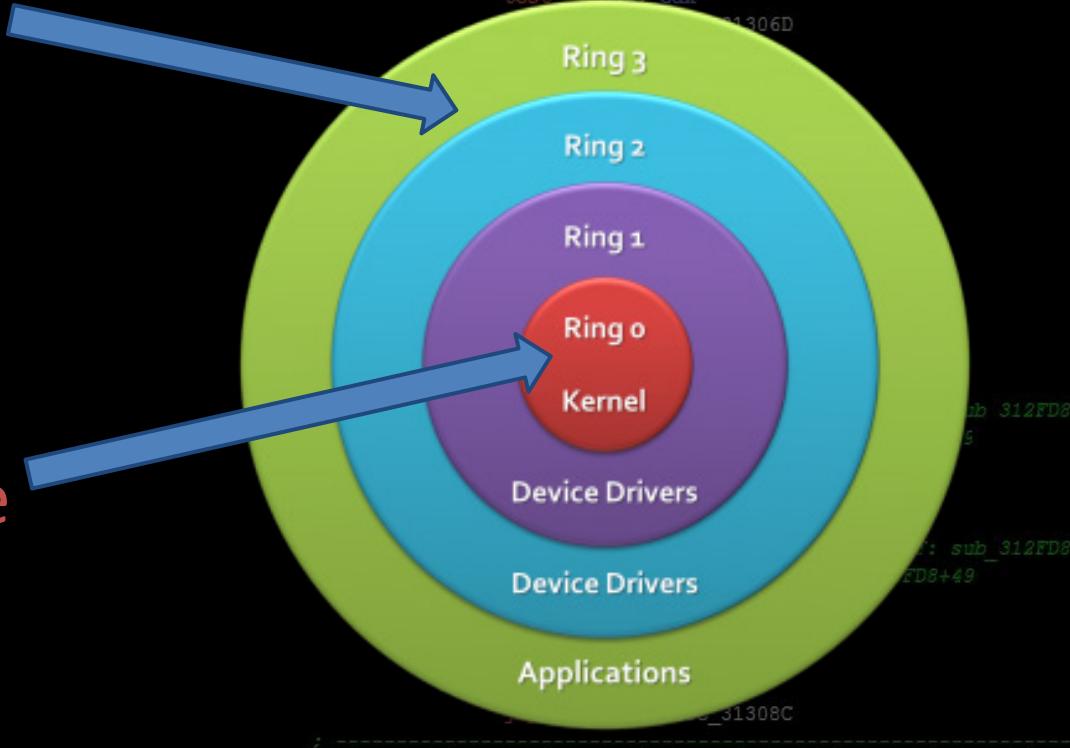
```
Applications
```

```
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
```

What's a Kernel? - Ring Model

We've Been Here



```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_01], ebx
jnz    short loc_313066
or     eax, [ebp+var_70]
cmp    [ebp+var_84], ebx
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
```

```
; CODE XREF: sub_312FD8+5
```

```
loc_31306D:
```

```
call    sub_3140F3
```

```
and    eax, 0FFFFh
```

```
or     eax, 80070000h
```

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

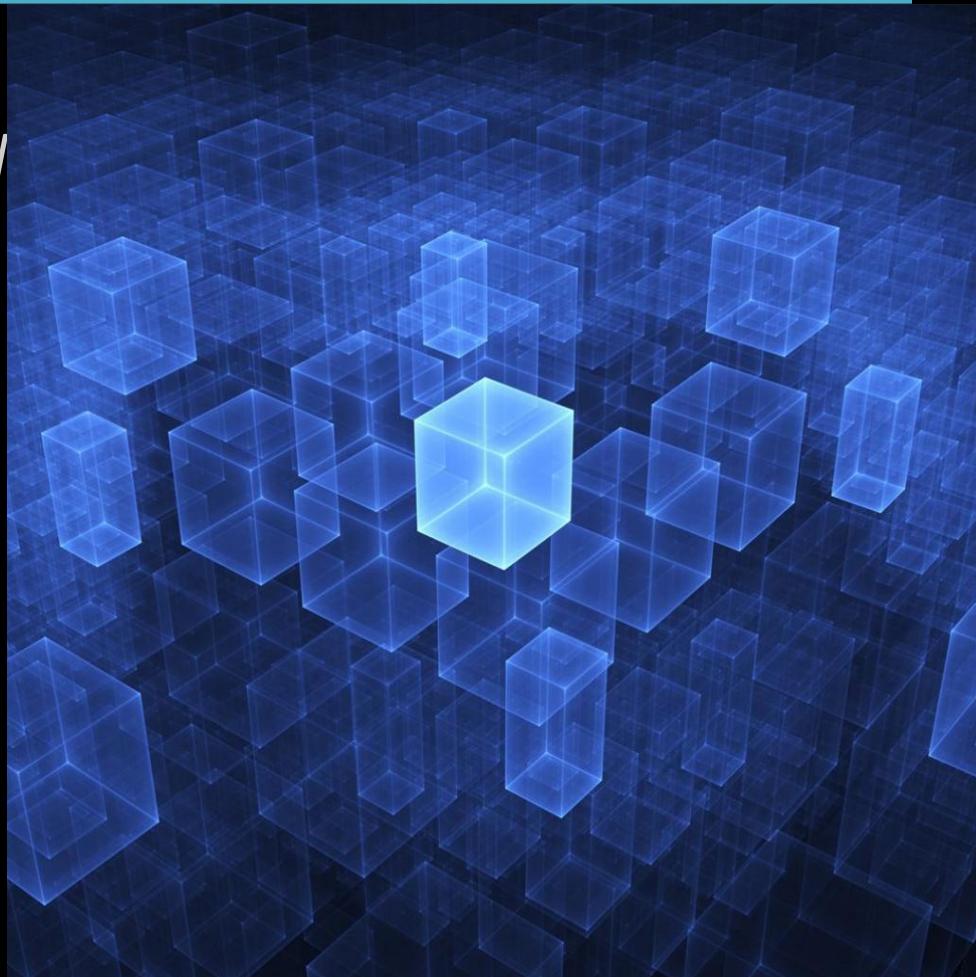
```
loc_31308C:
```

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

```
; CODE XREF: sub_312FD8+5C
```

Obligatory Matrix Analogy

"The Matrix is the world that has been pulled over your eyes to blind you from the truth." - Morpheus



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

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

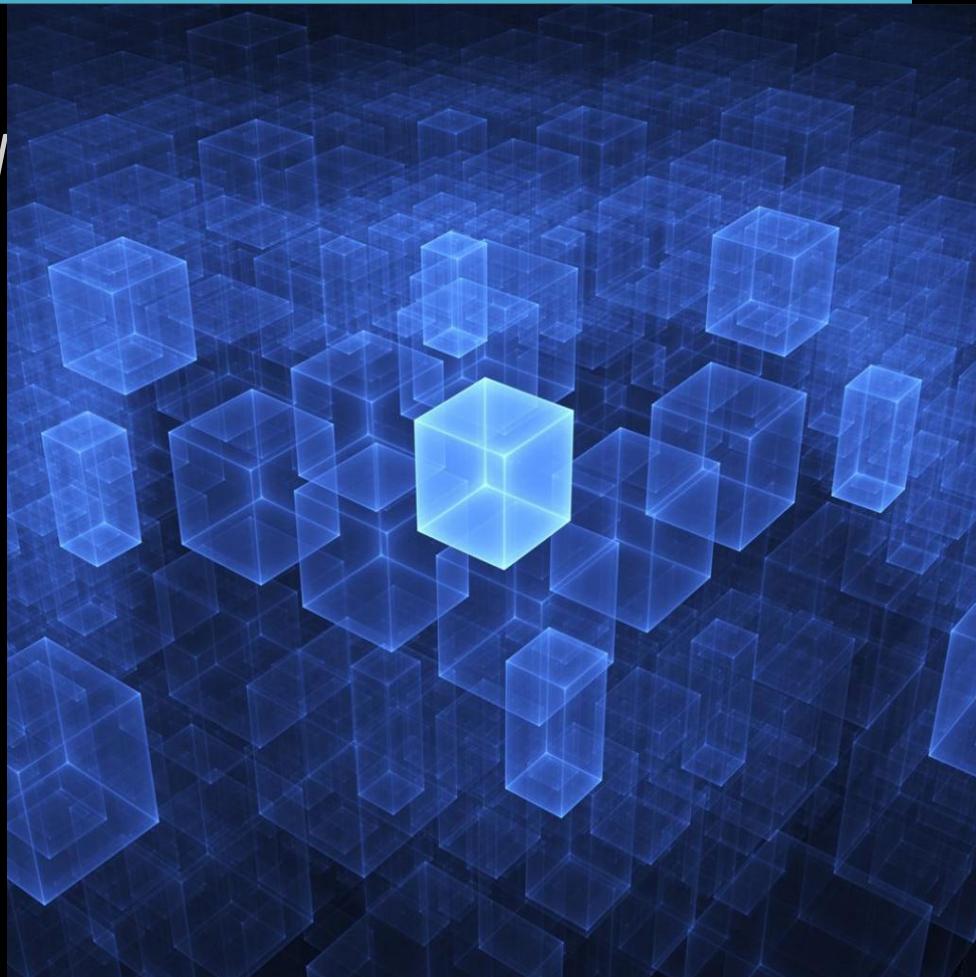
loc_31308C:

```
; CODE XREF: sub_312FDE
mov    [ebp+var_4], eax
```

Obligatory Matrix Analogy

“The Matrix is the world that has been pulled over your eyes to blind you from the truth.” - Morpheus

The kernel provides the “matrix” your programs run in



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

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

loc_31308C:

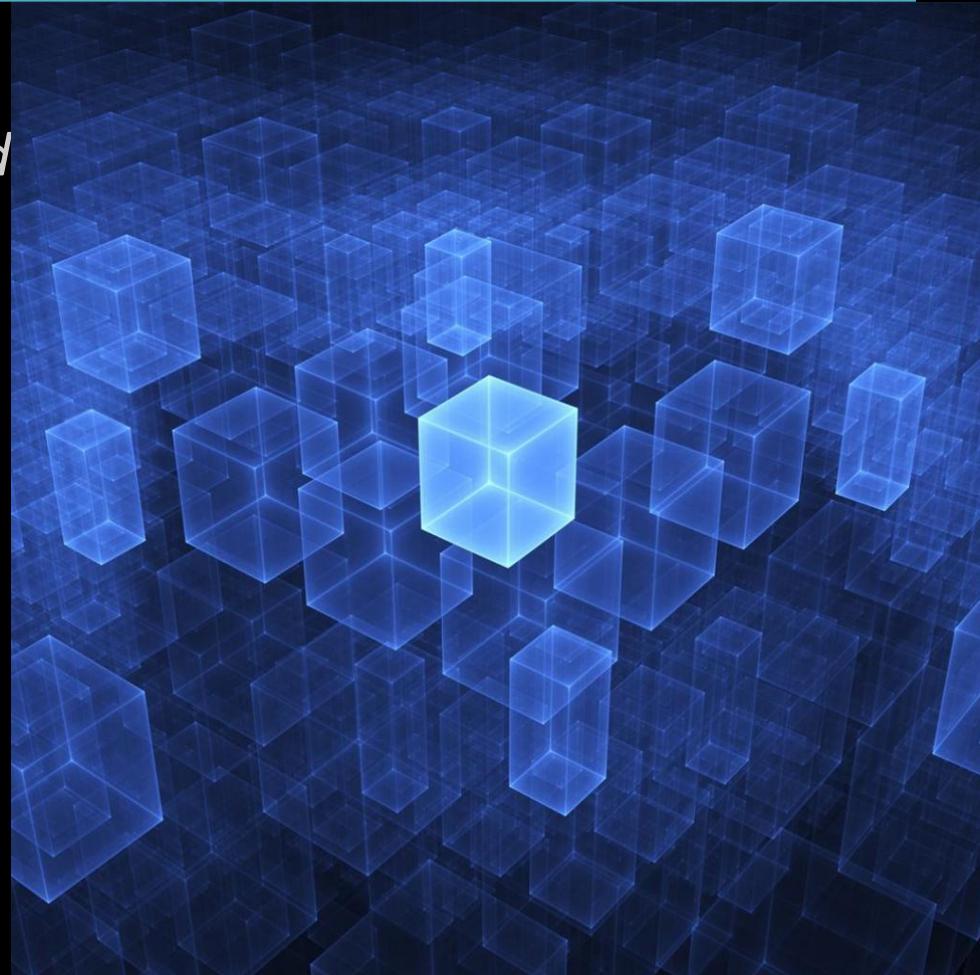
```
; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Obligatory Matrix Analogy

“The Matrix is the world that has been pulled over your eyes to blind you from the truth.” - Morpheus

The kernel provides the “matrix” your programs run in

Break out of the Matrix, and you pwn the entire system



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

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

loc_31308C:

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

Kernel Pwning in Popular Culture

“Jailbreaking” or “rooting” devices often depends on finding and leveraging Kernel bugs



Kernel Pwning in Popular Culture

“Jailbreaking” or “rooting” devices often depends on finding and leveraging Kernel bugs

Remember JailbreakMe?



Kernel Pwning in Popular Culture

“Jailbreaking” or “rooting” devices often depends on finding and leveraging Kernel bugs

Remember JailbreakMe?

It used a **remote code execution** primitive inside Safari to trigger a **kernel-level exploit** to bypass Apple’s code-signing protection



Kernel Basics

Your Kernel is:

Managing your Processes
Managing your Memory
Coordinating your Hardware

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

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

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

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

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

Kernel Basics

Your Kernel is:

Managing your Processes
Managing your Memory
Coordinating your Hardware

A crash oftentimes means a reboot!

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

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

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

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

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

Kernel Basics

Your Kernel is:

Managing your Processes
Managing your Memory
Coordinating your Hardware

A crash oftentimes means a reboot!

In general, we want to spend as little time there as possible.

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

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

loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
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
```

Basic Exploitation Strategy

The Kernel is typically *the most powerful* place we can find bugs

```
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]
mov    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
; CODE XREF: sub_312FD8+180
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Basic Exploitation Strategy

The Kernel is typically *the most powerful* place we can find bugs

But, how do we go from “vulnerability” to “privileged execution” *without bringing down the rest of the system?*

```
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]
mov    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
; CODE XREF: sub_312FD8+180
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    14623
; CODE XREF: sub_312FD8+180
test   ax, eax
short loc_31306D
jz     short loc_31306D
cmp    [ebp+arg_0], esi
; CODE XREF: sub_312FD8+180
; -----
loc_313066:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
push    0Dh
call    sub_31411B
; -----
loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test   eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:                                ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
; -----
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Basic Exploitation Strategy

The Big Picture

1. Find **vulnerability** in kernel code
2. Manipulate it to gain **code execution**
3. Elevate our process's **privilege level**
4. **Survive** the “trip” back to userland
5. Enjoy our **root privileges**

```
push    edi
call   sub_314623
test   eax, eax
jne    loc_31306D
        [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
        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
        [ebp+arg_0], esi
jz    short loc_31308F

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

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

You already know how to find these!

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

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

loc_31306D:                         ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; ----- ; CODE XREF: sub_312FD8
loc_31307D:                         ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h

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

Basic Exploitation Strategy

The Vulnerabilities

You already know how to find these!

Kernel vulnerabilities are almost exactly the same as userland vulnerabilities.

```
push    edi
call   sub_314623
test    eax, eax
jne    loc_31306D
    ebx=00000000, ebx=00000000
jnz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jbe    short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
    [ebp+arg_0], eax
sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
push    esi
push    [ebp+arg_4]
push    edi
call   sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

You already know how to find these!

Kernel vulnerabilities are almost exactly the same as userland vulnerabilities.

1. Buffer Overflows
2. Signedness issues
3. Partial Overwrites
4. Use-After-Free

By now, finding these should be a familiar process

```
push edi
call sub_314623
test eax, eax
jne loc_31306D
    ebz+var_0 ebx
    jnz short loc_313066
    mov eax, [ebp+var_70]
    cmp eax, [ebp+var_84]
    jbe loc_313066
    sub eax, [ebp+var_84]
    push esi
    push esi
    push eax
    push edi
    xor [ebp+arg_0], eax
    call sub_31486A
    test eax, eax
    jz short loc_31306D
    push esi
    lea eax, [ebp+arg_0]
    push eax
    mpx [edi], 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+59
    push 0Dh
    call sub_31411B

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

The most common place to find vulnerabilities is inside of Loadable Kernel Modules (LKMs).

```
push edi
call sub_314623
test eax, eax
jne loc_31306D
    ebz+var_0 ebx
    short loc_313066
    mov eax, [ebp+var_70]
    cmp eax, [ebp+var_84]
    jbe loc_313066
    sub eax, [ebp+var_84]
    push esi
    push esi
    push eax
    push edi
    call sub_314624
    test eax, eax
    jz short loc_31306D
    push esi
    lea eax, [ebp+arg_0]
    push eax
    mov esi, 1D0h
    push esi
    push [ebp+arg_4]
    push edi
    call sub_314623
    test eax, eax
    jz short loc_31306D
    cmp [ebp+arg_0], esi
    jz short loc_31308F

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

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

loc_31307D: ; CODE XREF: sub_312FD8+59
    call sub_3140F3
    and eax, 0FFFFh
    or eax, 80070000h

loc_31308C: ; CODE XREF: sub_312FD8+59
    mov [ebp+var_4], eax
```

Basic Exploitation Strategy

The Vulnerabilities

The most common place to find vulnerabilities is inside of Loadable Kernel Modules (LKMs).

LKMs are like executables that run in Kernel Space.
A few common uses are listed below:

- > Device Drivers
- > Filesystem Drivers
- > Networking Drivers
- > Executable Interpreters
- > Kernel Extensions
- > (rootkits :P)

```
push edi
call sub_314623
test eax, eax
jne loc_31306D
    ebz+var_0 ebx
    jnz short loc_313066
    mov eax, [ebp+var_70]
    cmp eax, [ebp+var_84]
    jbe loc_313066
    sub eax, [ebp+var_84]
    push esi
    push esi
    push eax
    push edi
    call sub_314623
    test eax, eax
    jz short loc_31306D
    push esi
    lea eax, [ebp+arg_0]
    push eax
    mov esi, 1D0h
    push edi
    call sub_314623
    test eax, eax
    jz short loc_31306D
    cmp [ebp+arg_0], esi
    jz short loc_31308F

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

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

LKMs are just binary blobs like your familiar ELF's, EXE's and MACH-O's. (On Linux, they even use the ELF format)

```
push    edi
call    sub_314623
test    eax, eax
jne    loc_31306D
    ebp+var_0, ebx
jnz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov    [ebp+arg_0], eax
call    sub_31462A
test    eax, eax
jne    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+59
push    0Dh
call    sub_31411B

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

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

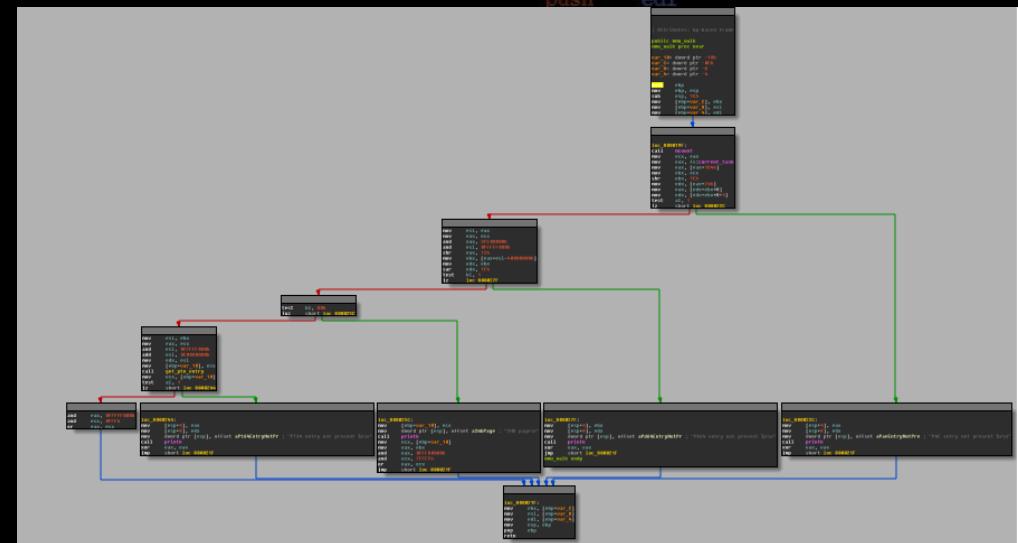
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Basic Exploitation Strategy

The Vulnerabilities

LKM's are just binary blobs like your familiar ELF's, EXE's and MACH-O's. (On Linux, they even use the ELF format)

You can drop them into IDA and reverse-engineer them like you're used to already.



Basic Exploitation Strategy

The Vulnerabilities

There's a few useful commands that deal with LKMs on Linux.

```
push    edi
call   sub_314623
test   eax, eax
jne    loc_31306D
jne    [ebp+var_0], ebx
jnz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
[...]
call   sub_31462A
test   eax, eax
jz    short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
jz    short loc_31308F

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

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

There's a few useful commands that deal with LKMs on Linux.

- insmod ---> Insert a module into the running kernel
- rmmod ---> Remove a module from the running kernel
- lsmod ---> List currently loaded modules

```
push    edi
call   sub_314623
test   eax, eax
jne    loc_31306D
jne    [ebp+var_0], ebx
jnz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
short loc_313066
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
[...]
call   sub_31460A
test   eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    [ebp+arg_4]
[...]
push    [ebp+arg_2]
call   sub_314623
test   eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

There's a few useful commands that deal with LKMs on Linux.

- insmod ---> Insert a module into the running kernel
- rmmod ---> Remove a module from the running kernel
- lsmod ---> List currently loaded modules

A general familiarity with these is helpful

```
push edi
call sub_314623
test eax, eax
jne loc_31306D
    ebx=00000000 ebx
jnz short loc_313066
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
push edi
[...]
call sub_31460A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push [ebp+arg_4]
[...]
sub_314622
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F
```

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

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

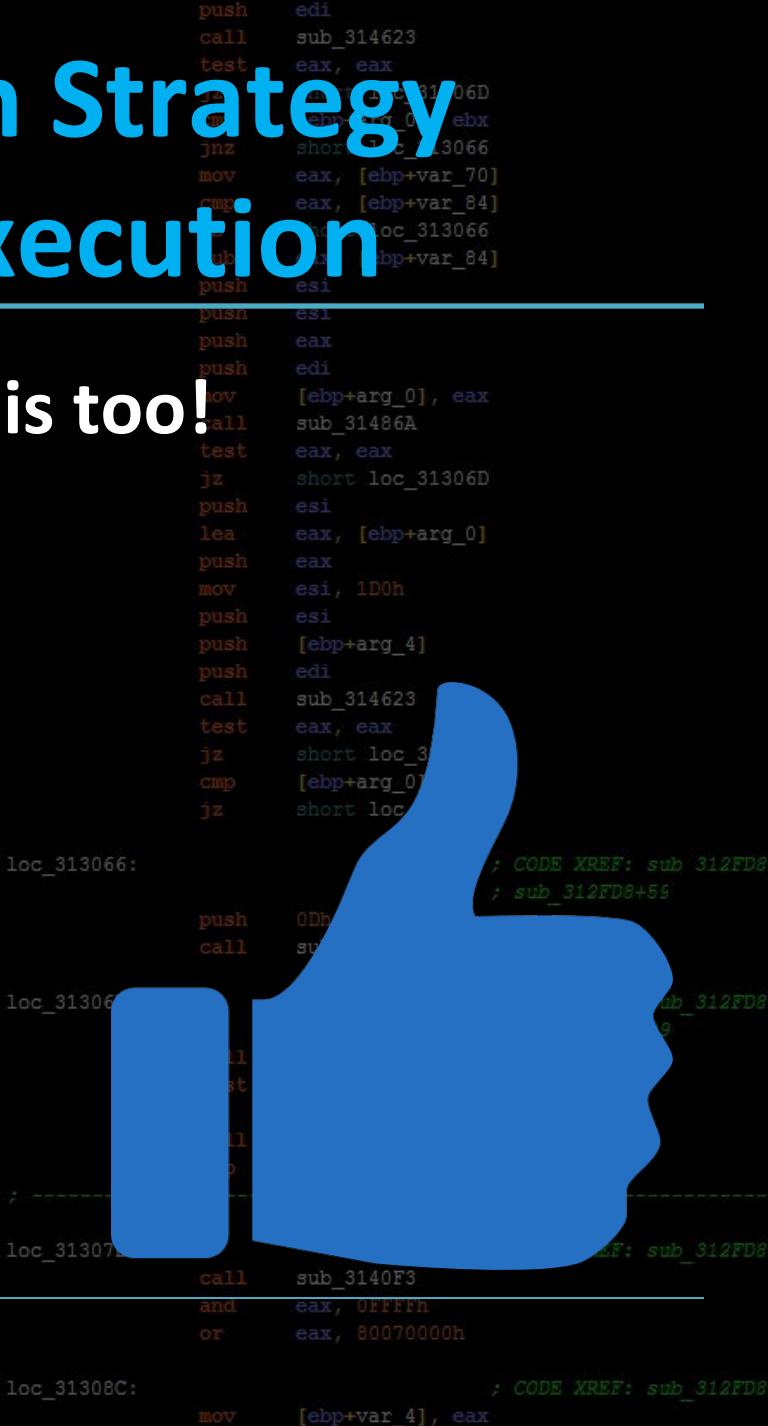
```
; -----
loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFh
or eax, 80070000h
```

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

Basic Exploitation Strategy

Gaining Code Execution

You already know how to do this too!



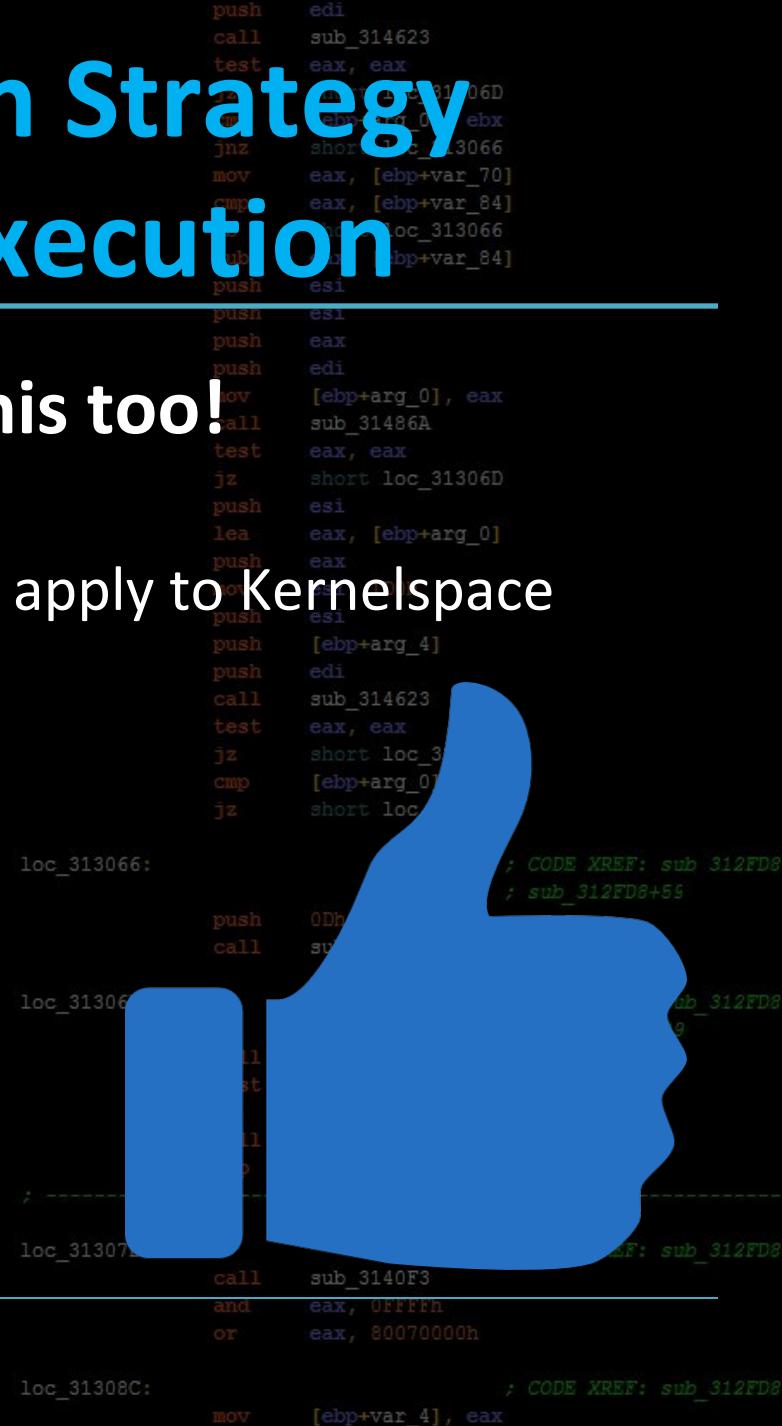
```
push edi
call sub_314623
test eax, eax
jne loc_31306D
    push ebx
    shr eax, ebx
    mov eax, [ebp+var_70]
    imul eax, [ebp+var_84]
    add eax, loc_313066
    add eax, [ebp+var_84]
    push esi
    push esi
    push eax
    push edi
    mov [ebp+arg_0], eax
    call sub_31486A
    test eax, 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_3
    cmp [ebp+arg_0]
    jz short loc_313066:
; CODE XREF: sub_312FD8
; sub_312FD8+59
    push 0Dh
    call sub_312FD8
loc_313066:
    push 0Dh
    call sub_312FD8
loc_313067:
    push 0Dh
    call sub_312FD8
; -----
loc_313071:
    push 0Dh
    call sub_312FD8
loc_313075:
    push 0Dh
    call sub_312FD8
loc_313079:
    push 0Dh
    call sub_312FD8
; CODE XREF: sub_312FD8
; sub_312FD8+59
    and eax, 0FFFFh
    or eax, 80070000h
loc_31308C:
    mov [ebp+var_4], eax
; CODE XREF: sub_312FD8
```

Basic Exploitation Strategy

Gaining Code Execution

You already know how to do this too!

The same basic exploitation techniques apply to Kernelspace
(After all, it's just x86 code!)



```
push edi
call sub_314623
test eax, eax
jne loc_31306D
    ebx
    short loc_3066
    mov eax, [ebp+var_70]
    mov eax, [ebp+var_84]
    add eax, loc_313066
    add 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
    or eax, 0Dh
    push esi
    push [ebp+arg_4]
    push edi
    call sub_314623
    test eax, eax
    jz short loc_31306D
    cmp [ebp+arg_0], 0
    jz short loc_31306D
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313066:
    push 0Dh
    call sub_3140F3
; -----
loc_313067:
    call sub_3140F3
    test eax, eax
    jne loc_31307E
    and eax, 0FFFFh
    or eax, 80070000h
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_31307E:
    call sub_3140F3
    test eax, eax
    jne loc_31308C
    and eax, 0FFFFh
    or eax, 80070000h
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_31308C:
    mov [ebp+var_4], eax
    ; CODE XREF: sub_312FD8
    ; sub_312FD8+59
```

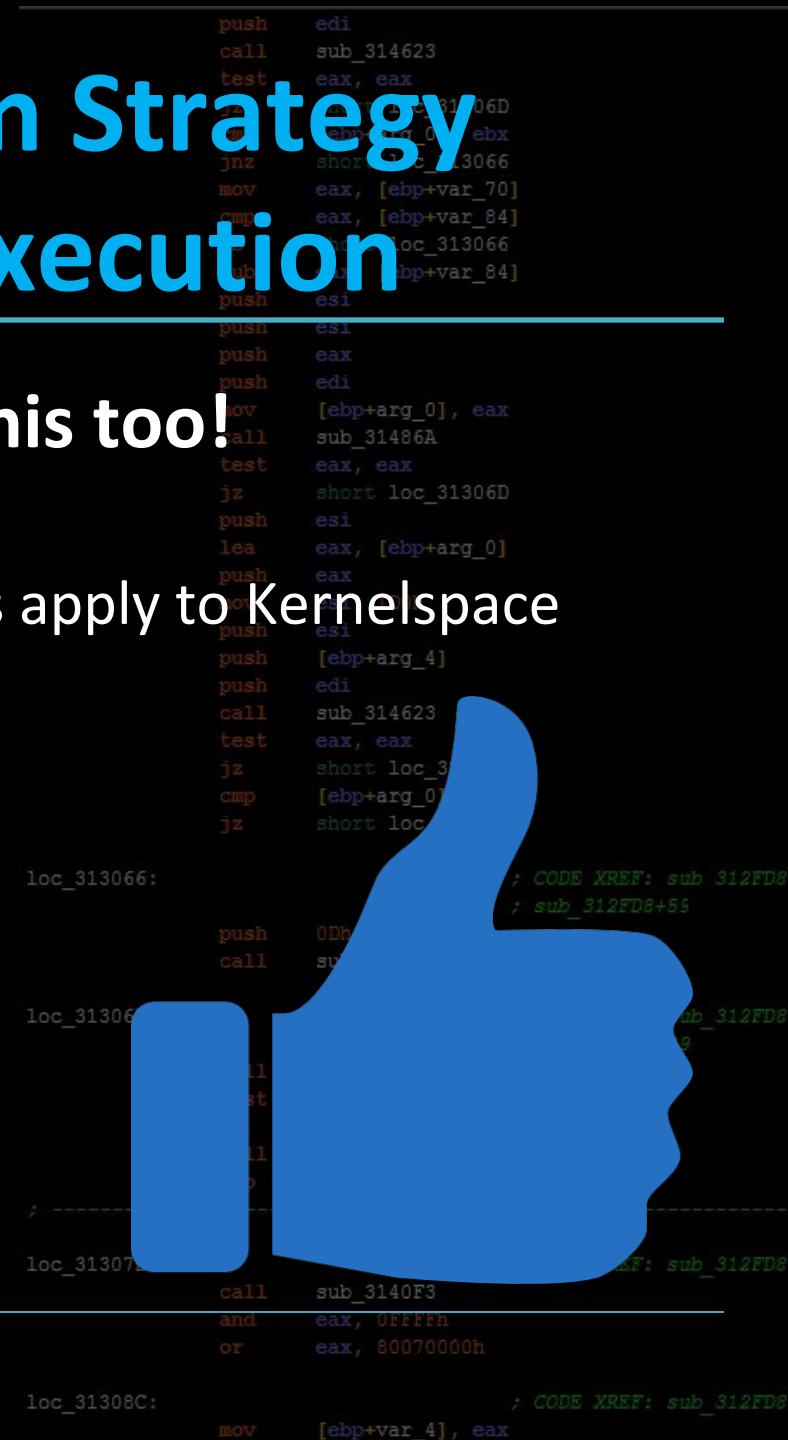
Basic Exploitation Strategy

Gaining Code Execution

You already know how to do this too!

The same basic exploitation techniques apply to Kernelspace
(After all, it's just x86 code!)

Shellcoding, ROP, Pointer Overwrites,
Type Confusion, etc can all be used to
execute code in Kernel Land.



```
push    edi
call    sub_314623
test    eax, eax
jne    short loc_31306D
push    ebx
shor    ebx, ebx
mov     eax, [ebp+var_70]
mov     eax, [ebp+var_84]
add    eax, 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_3
cmp    [ebp+arg_0]
jz     short loc_313066:
loc_313066:
push    0Dh
call    sub_312FD8
; CODE XREF: sub_312FD8+59
; sub_312FD8+59
loc_313067:
loc_313070:
loc_313073:
loc_313076:
loc_313079:
loc_313082:
loc_313085:
loc_313088:
loc_31308B:
loc_31308C:
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
loc_31308C:                                ; CODE XREF: sub_312FD8+59
mov     [ebp+var_4], eax
```

Basic Exploitation Strategy

Gaining Code Execution

You already know how to do this too!

The same basic exploitation techniques apply to Kernelspace
(After all, it's just x86 code!)

Shellcoding, ROP, Pointer Overwrites,
Type Confusion, etc can all be used to
execute code in Kernel Land.

Typically, you won't have to deal with ASLR!

```
push    edi
call    sub_314623
test    eax, eax
jne    short loc_31306D
push    ebx
shor    ebx, ebx
mov     eax, [ebp+var_70]
mov     eax, [ebp+var_84]
add    eax, 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_3
cmp    [ebp+arg_0]
jz     short loc_312FD8+59
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313066:
push    0Dh
call    sub_314623
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313067:
push    00h
call    sub_314623
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313071:
push    00h
test    eax, eax
jne    short loc_313071
and    eax, 0FFFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_31308C:
mov     [ebp+var_4], eax
; CODE XREF: sub_312FD8
; sub_312FD8+59
```


Basic Exploitation Strategy

Gaining Code Execution

Common Library calls are sometimes *different*, so there is a slight learning curve involved.

printf() --->
memcpy() --->
malloc() --->

printk()
copy_from_user()/copy_to_user()
kmalloc()

```
push    edi
call   sub_314623
test   eax, eax
jne    loc_31306D
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
jnz    loc_313066
mov    eax, [ebp+var_70]
imov   eax, [ebp+var_84]
add    eax, loc_313066
sub    eax, [ebp+var_84]
push   esi
push   esi
push   eax
push   edi
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
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
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
jz     short loc_31306F
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
; CODE XREF: sub_312FD8+59
; sub_312FD8+59
push   0Dh
call   sub_31411B
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
loc_31306D:
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
; 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:
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
call   sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
loc_31308C:
        ebx=00000000, eax=00000000, ebx=00000000, esp=00000000, ebp=00000000, esi=00000000, edi=00000000
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
```

Basic Exploitation Strategy

Gaining Code Execution

Common Library calls are sometimes *different*, so there is a slight learning curve involved.

printf() --->
memcpy() --->
malloc() --->

printf()
copy_from_user()/copy_to_user()
kmalloc()

Typically, whatever you want to know is a quick google-search or man page away.

```
push    edi  
call   sub_314623  
test   eax, eax  
jne    short loc_31306D  
push   ebx  
shor   [ebp+var_70]  
mov    eax, [ebp+var_84]  
add    eax, loc_313066  
sub    eax, [ebp+var_84]  
push   esi  
push   esi  
push   eax  
push   edi  
sub   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  
push   esi  
push   esi  
sub   eax  
push   0Dh  
call  sub_31411B  
; CODE XREF: sub_312FD8  
; sub_312FD8+59  
;-----  
loc_313066:  
call   sub_3140F3  
test   eax, eax  
jg    short loc_31307D  
call   sub_3140F3  
jmp   short loc_31308C  
;-----  
loc_31307D:  
call   sub_3140F3  
and    eax, 0FFFFh  
or     eax, 80070000h  
; CODE XREF: sub_312FD8  
;-----  
loc_31308C:  
mov    [ebp+var_4], eax  
; CODE XREF: sub_312FD8
```

Basic Exploitation Strategy

Gaining Code Execution

Debugging kernel code can be difficult

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

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

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

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

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

Basic Exploitation Strategy

Gaining Code Execution

Debugging kernel code can be difficult

We can't just run the kernel in gdb

```
push    edi  
call    sub_314623  
test    eax, eax  
jne    short loc_31306D  
jne    short loc_31306D  
push    ebx  
shor    ebx  
mov     eax, [ebp+var_70]  
mov     eax, [ebp+var_84]  
add    eax, [loc_313066]  
sub    eax, [ebp+var_84]  
  
push    esi  
push    esi  
push    eax  
push    edi  
mov     [ebp+arg_0], eax  
call    sub_31486A  
test    eax, eax  
jz     short loc_31306D  
push    esi  
lea     eax, [ebp+arg_0]  
push    eax  
mov     esi, 1D0h  
push    esi  
push    [ebp+arg_4]  
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_31306D  
cmp    [ebp+arg_0], esi  
jz     short loc_31308F  
  
loc_313066:           ; CODE XREF: sub_312FD8+  
                      ; sub_312FD8+59  
push    0Dh  
call    sub_31411B  
  
loc_31306D:           ; CODE XREF: sub_312FD8+  
                      ; sub_312FD8+49  
call    sub_3140F3  
test    eax, eax  
jg     short loc_31307D  
call    sub_3140F3  
jmp    short loc_31308C  
;  
  
loc_31307D:           ; CODE XREF: sub_312FD8+  
call    sub_3140F3  
and    eax, 0FFFFh  
or     eax, 80070000h  
  
loc_31308C:           ; CODE XREF: sub_312FD8+  
mov     [ebp+var_4], eax
```

Basic Exploitation Strategy

Gaining Code Execution

Debugging kernel code can be difficult

We can't just run the kernel in gdb

You will often have to rely on stack dumps, error messages, and other “black box” techniques to infer what's going on inside the kernel.

```
push    edi  
call    sub_314623  
test    eax, eax  
jne    short loc_31306D  
push    ebx  
shor    ebx, ebx  
mov     eax, [ebp+var_70]  
cmpl   eax, [ebp+var_84]  
jne    short loc_313066  
push    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  
test    eax, eax  
jz     short loc_31306F  
push    esi  
test    eax, eax  
jz     short loc_31306D  
push    esi  
test    eax, eax  
jz     short loc_31308F
```

```
loc_313066: ; CODE XREF: sub_312FD8  
; sub_312FD8+59  
push    0Dh  
call    sub_31411B  
  
loc_31306D: ; CODE XREF: sub_312FD8  
; sub_312FD8+49  
call    sub_3140F3  
test    eax, eax  
jg     short loc_31307D  
call    sub_3140F3  
jmp    short loc_31308C  
  
loc_31307D: ; CODE XREF: sub_312FD8  
call    sub_3140F3  
and    eax, 0FFFh  
or     eax, 80070000h  
  
loc_31308C: ; CODE XREF: sub_312FD8  
mov     [ebp+var_4], eax
```

Basic Exploitation Strategy

Gaining Code Execution

This is an example of what you might see if you get a crash in the kernel.

```
push    edi
call    sub_314623
test    eax, eax
jne    loc_314623+06D
        ebpf=0000000000000000 ebx
        jnz    loc_3066
        mov    eax, [ebp+var_70]
        cmov   eax, [ebp+var_84]
        mov    eax, loc_313066
        add    eax, [ebp+var_84]
        push   esi
        push   exit
        push   cs:loc_313066
        push   [ebp+var_84]

PID: 0      TASK: ffff81121ff987b0  CPU: 2  COMMAND: "swapper"
#0 [ffff81011fff3b80] crash_kexec at ffffffff800b1287
#1 [ffff81011fff3c40] __die at ffffffff80065137
#2 [ffff81011fff3c80] do_page_fault at ffffffff8006741e
#3 [ffff81011fff3d70] error_exit at ffffffff8005ddf9
[exception RIP: uhci_scan_schedule+162]
RIP: ffffffff880218ee RSP: ffff81011fff3e20 RFLAGS: 00010007
RAX: 0000002019105000 RBX: 0000002019105000 RCX: ffff81121ff8cb68
RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffff81091fe27950
RBP: ffff81011fff3ed0 R8: 0000000000000000 R9: ffff81012b4f7df8
R10: 0000000000000001 R11: 00000000af482de4 R12: ffff81091fe27950
R13: 0000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG_RAX: ffffffffffffdfffff CS: 0010 SS: 0018
#4 [ffff81011fff3e98] uhci_hub_status_data at ffffffff880232da [uhci_hcd]
#5 [ffff81011fff3ec8] usb_hcd_poll_rh_status at ffffffff80200275
#6 [ffff81011fff3f08] run_timer_softirq at ffffffff8009a819
#7 [ffff81011fff3f58] __do_softirq at ffffffff800125a9
#8 [ffff81011fff3f88] call_softirq at ffffffff8005e30c
#9 [ffff81011fff3fa0] do_softirq at ffffffff8006d630
#10 [ffff81011fff3fb0] apic_timer_interrupt at ffffffff8005dc9e
--- <IRQ stack> ---
#11 [ffff81011ffefdf8] apic_timer_interrupt at ffffffff8005dc9e
[exception RIP: acpi_safe_halt+37]
RIP: ffffffff801a62ab RSP: ffff81011ffefea0 RFLAGS: 00000246
RAX: 0000000000000000 RBX: ffff81121ff1f8a8 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 0000000000000001 RDI: 0000000000000001
RBP: ffff81011ffefee8 R8: ffff81011fffee000 R9: 000000000000003f
R10: ffff81091fdc4008 R11: 00000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 0000000000000000 R15: ffff81118d6700c0
ORIG_RAX: ffffffffffffdfffff10 CS: 0010 SS: 0018
#12 [ffff81011ffefea0] acpi_processor_idle_simple at ffffffff801a6b29

        and    eax, 0FFFFh
        or     eax, 80070000h

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

Basic Exploitation Strategy

Gaining Code Execution

This is an example of what you might see if you get a crash in the kernel.

Call Trace

Register Dump

Stack Dump

```
push    edi
call    sub_314623
test    eax, eax
jne    loc_314623+06D
        ebpf=0000000000000000 ebx
        jnz    loc_3066
        mov    eax, [ebp+var_70]
        cmov   eax, [ebp+var_84]
        add    eax, loc_313066
        sub    eax, [ebp+var_84]
        push   esi
        push   exit
        ; CODE XREF: sub_312FD8

PID: 0      TASK: ffff81121ff987b0  CPU: 2  COMMAND: "swapper"
#0 [ffff81011fff3b80] crash_kexec at ffffffff800b1287
#1 [ffff81011fff3c40] __die at ffffffff80065137
#2 [ffff81011fff3c80] do_page_fault at ffffffff8006741e
#3 [ffff81011fff3d70] error_exit at ffffffff8005ddf9
[exception RIP: uhci_scan_schedule+162]
RIP: ffffffff880218ee RSP: ffff81011fff3e20 RFLAGS: 00010007
RAX: 0000002019105000 RBX: 0000002019105000 RCX: ffff81121ff8cb68
RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffff81091fe27950
RBP: ffff81011fff3ed0 R8: 0000000000000000 R9: ffff81012b4f7df8
R10: 0000000000000001 R11: 00000000af482de4 R12: ffff81091fe27950
R13: 0000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG_RAX: ffffffffffffffff CS: 0010 SS: 0018
#4 [ffff81011fff3e98] uhci_hub_status_data at ffffffff880232da [uhci_hcd]
#5 [ffff81011fff3ec8] usb_hcd_poll_rh_status at ffffffff80200275
#6 [ffff81011fff3f08] run_timer_softirq at ffffffff8009a819
#7 [ffff81011fff3f58] __do_softirq at ffffffff800125a9
#8 [ffff81011fff3f88] call_softirq at ffffffff8005e30c
#9 [ffff81011fff3fa0] do_softirq at ffffffff8006d630
#10 [ffff81011fff3fb0] apic_timer_interrupt at ffffffff8005dc9e
--- <IRQ stack> ---
#11 [ffff81011ffefdf8] apic_timer_interrupt at ffffffff8005dc9e
[exception RIP: acpi_safe_halt+37]
RIP: ffffffff801a62ab RSP: ffff81011ffefea0 RFLAGS: 00000246
RAX: 0000000000000000 RBX: ffff81121ff1f8a8 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 0000000000000001 RDI: 0000000000000001
RBP: ffff81011ffefee8 R8: ffff81011fffee000 R9: 000000000000003f
R10: ffff81091fdc4008 R11: 00000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 0000000000000000 R15: ffff81118d6700c0
ORIG_RAX: ffffffffffff10 CS: 0010 SS: 0018
#12 [ffff81011ffefea0] acpi_processor_idle_simple at ffffffff801a6b29
        and    eax, 0FFFFh
        or     eax, 80070000h
        ; CODE XREF: sub_312FD8

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

Basic Exploitation Strategy

Gaining Code Execution

This is an example of what you might see if you get a crash in the kernel.

Call Trace

Register Dump

Stack Dump

You might be able to see this with `dmesg` if the crash is not fatal.

```
push    edi
call    sub_314623
test    eax, eax
jne    loc_31306D
jne    [ebp+var_0], ebx
jnz    loc_3066
mov    eax, [ebp+var_70]
mov    eax, [ebp+var_84]
add    eax, loc_313066
sub    eax, [ebp+var_84]
push    esi
push    ebx
; CODE XREF: sub_312FD8
PID: 0      TASK: ffff81121ff987b0  CPU: 2  COMMAND: "swapper"
#0 [ffff81011fff3b80] crash_kexec at ffffffff800b1287
#1 [ffff81011fff3c40] __die at ffffffff80065137
#2 [ffff81011fff3c80] do_page_fault at ffffffff8006741e
#3 [ffff81011fff3d70] error_exit at ffffffff8005ddf9
[exception RIP: uhci_scan_schedule+162]
RIP: ffffffff880218ee RSP: ffff81011fff3e20 RFLAGS: 00010007
RAX: 0000002019105000 RBX: 0000002019105000 RCX: ffff81121ff8cb68
RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffff81091fe27950
RBP: ffff81011fff3ed0 R8: 0000000000000000 R9: ffff81012b4f7df8
R10: 0000000000000001 R11: 00000000af482de4 R12: ffff81091fe27950
R13: 0000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG_RAX: ffffffffffffdffff CS: 0010 SS: 0018
#4 [ffff81011fff3e98] uhci_hub_status_data at ffffffff880232da [uhci_hcd]
#5 [ffff81011fff3ec8] usb_hcd_poll_rh_status at ffffffff80200275
#6 [ffff81011fff3f08] run_timer_softirq at ffffffff8009a819
#7 [ffff81011fff3f58] __do_softirq at ffffffff800125a9
#8 [ffff81011fff3f88] call_softirq at ffffffff8005e30c
#9 [ffff81011fff3fa0] do_softirq at ffffffff8006d630
#10 [ffff81011fff3fb0] apic_timer_interrupt at ffffffff8005dc9e
--- <IRQ stack> ---
#11 [ffff81011ffefdf8] apic_timer_interrupt at ffffffff8005dc9e
[exception RIP: acpi_safe_halt+37]
RIP: ffffffff801a62ab RSP: ffff81011ffefea0 RFLAGS: 00000246
RAX: 0000000000000000 RBX: ffff81121ff1f8a8 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 0000000000000001 RDI: 0000000000000001
RBP: ffff81011ffefee8 R8: ffff81011fffee000 R9: 000000000000003f
R10: ffff81091fdc4008 R11: 00000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 0000000000000000 R15: ffff81118d6700c0
ORIG_RAX: ffffffffffffdffff CS: 0010 SS: 0018
#12 [ffff81011ffefea0] acpi_processor_idle_simple at ffffffff801a6b29
and    eax, 0FFFFh
or     eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
        mov    [ebp+var_4], eax
```

Basic Exploitation Strategy

Elevate Privileges

Remember: The Kernel manages running processes

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

Remember: The Kernel manages running processes

Therefore: The Kernel keeps track of permissions

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

Remember: The Kernel manages running processes

Therefore: The Kernel keeps track of permissions

```
struct task_struct {  
    ...  
    /* process credentials */  
    const struct cred __rcu *real_cred;  
    const struct cred __rcu *cred;  
    char comm[TASK_COMM_LEN];  
};
```

linux/include/linux/sched.h

```
push    edi  
call   sub_314623  
test   eax, eax  
jz    short loc_31306D  
mov    eax, [ebp+var_70]  
cmp    eax, [ebp+var_84]  
short loc_313066  
push    eax, [ebp+var_84]  
push    esi  
push    esi  
push    eax  
push    edi  
pop    eax  
call   sub_31486A  
test   eax, eax  
jz    short loc_31306D  
push    esi  
lea    eax, [ebp+arg_0]  
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+59  
push    0Dh  
call   sub_31411B  
  
loc_31306D: ; CODE XREF: sub_312FD8+  
            ; sub_312FD8+49  
call   sub_3140F3  
test   eax, eax  
jg    short loc_31307D  
call   sub_3140F3  
jmp    short loc_31308C  
  
loc_31307D: ; CODE XREF: sub_312FD8+  
            ; sub_312FD8+49  
call   sub_3140F3  
and    eax, 0FFFFh  
or     eax, 80070000h  
  
loc_31308C: ; CODE XREF: sub_312FD8+  
            ; sub_312FD8+49  
mov    [ebp+var_4], eax
```

Basic Exploitation Strategy

Elevate Privileges

Conveniently, the Linux Kernel has a wrapper for updating process credentials!

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

Conveniently, the Linux Kernel has a wrapper for updating process credentials!

```
int commit_creds(struct cred *new) {  
    ...  
}
```

```
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_313066  
mov    eax, [ebp+var_70]  
cmp    eax, [ebp+var_84]  
short loc_313066  
push    eax, [ebp+var_84]  
push    esi  
push    esi  
push    eax  
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_31306D  
push    esi  
lea     eax, [ebp+arg_0]  
push    eax  
mov    esi, 1D0h  
push    esi  
push    [ebp+arg_4]  
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_31306D  
cmp    [ebp+arg_0], esi  
jz     short loc_31308F  
  
loc_313066:          ; CODE XREF: sub_312FD8+  
                     ; sub_312FD8+59  
push    0Dh  
call    sub_31411B  
  
loc_31306D:          ; CODE XREF: sub_312FD8+  
                     ; sub_312FD8+49  
call    sub_3140F3  
test    eax, eax  
jg     short loc_31307D  
call    sub_3140F3  
jmp    short loc_31308C  
-----  
loc_31307D:          ; CODE XREF: sub_312FD8+  
call    sub_3140F3  
and    eax, 0FFFh  
or     eax, 80070000h  
  
loc_31308C:          ; CODE XREF: sub_312FD8+  
mov    [ebp+var_4], eax
```

Basic Exploitation Strategy

Elevate Privileges

Conveniently, the Linux Kernel has a wrapper for updating process credentials!

```
int commit_creds(struct cred *new) {  
    ...  
}
```

We just need to create a valid cred struct!

```
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_313066  
mov    eax, [ebp+var_70]  
cmp    eax, [ebp+var_84]  
short loc_313066  
push    eax, [ebp+var_84]  
push    esi  
push    esi  
push    eax  
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_31306D  
push    esi  
lea    eax, [ebp+arg_0]  
push    eax  
mov    esi, 1D0h  
push    esi  
push    [ebp+arg_4]  
push    edi  
call    sub_314623  
test    eax, eax  
jz     short loc_31306D  
cmp    [ebp+arg_0], esi  
jz     short loc_31308F  
  
loc_313066:                                ; CODE XREF: sub_312FD8  
; sub_312FD8+59  
push    0Dh  
call    sub_31411B  
  
loc_31307D:                                ; CODE XREF: sub_312FD8  
; sub_312FD8+49  
call    sub_3140F3  
test    eax, eax  
jg     short loc_31307D  
call    sub_3140F3  
jmp    short loc_31308C  
  
loc_31307D:                                ; CODE XREF: sub_312FD8  
call    sub_3140F3  
and    eax, 0FFFh  
or     eax, 80070000h  
  
loc_31308C:                                ; CODE XREF: sub_312FD8  
mov    [ebp+var_4], eax
```

Basic Exploitation Strategy

Elevate Privileges

The kernel is helpful again!

```
struct cred *prepare_kernel_cred(struct task_struct *daemon) {  
    ...  
}
```

```
    push    edi  
    call    sub_314623  
    test    eax, eax  
    jz     short loc_313066  
    mov     eax, [ebp+var_70]  
    cmp     eax, [ebp+var_84]  
    short loc_313066  
    mov     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     ax, [ebp+arg_0]  
    mov     eax, 1D0h  
    push    esi  
    push    [ebp+arg_4]  
    push    edi  
    call    sub_314623  
    test    eax, eax  
    jz     short loc_31306D  
    cmp     [ebp+arg_0], esi  
    jz     short loc_31308F  
  
loc_313066:                                ; CODE XREF: sub_312FD8  
                                                ; sub_312FD8+59  
    push    0Dh  
    call    sub_31411B  
  
loc_31306D:                                ; CODE XREF: sub_312FD8  
                                                ; sub_312FD8+49  
    call    sub_3140F3  
    test    eax, eax  
    jg     short loc_31307D  
    call    sub_3140F3  
    jmp     short loc_31308C  
; -----  
  
loc_31307D:                                ; CODE XREF: sub_312FD8  
    call    sub_3140F3  
    and    eax, 0FFFFh  
    or     eax, 80070000h  
  
loc_31308C:                                ; CODE XREF: sub_312FD8  
    mov     [ebp+var_4], eax
```

Basic Exploitation Strategy

Elevate Privileges

The kernel is helpful again!

```
struct cred *prepare_kernel_cred(struct task_struct *daemon) {
```

```
    ...
```

```
}
```

```
"
```

If @daemon is supplied, then the security data will be derived from that; otherwise they'll be set to 0 and no groups, full capabilities and no keys.

```
"
```

- *source/kernel/cred.c*

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

```
loc_313066:
```

; CODE XREF: sub_312FD8
; sub_312FD8+59

```
push    0Dh
call    sub_31411B
```

```
loc_31306D:
```

; CODE XREF: sub_312FD8
; sub_312FD8+49

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

```
loc_31307D:
```

; CODE XREF: sub_312FD8

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

```
loc_31308C:
```

; CODE XREF: sub_312FD8

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

Basic Exploitation Strategy

Elevate Privileges

Great! Now we can map out what we need to do

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

Great! Now we can map out what we need to do

1. Create a “root” “**struct creds**” by calling
`prepare_kernel_cred(NULL);`
2. Call `commit_creds(root cred *);`

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

Great! Now we can map out what we need to do

1. Create a “root” “**struct creds**” by calling **prepare_kernel_cred(NULL);**
2. Call **commit_creds(root cred *);**
3. Enjoy our new root privileges!



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

; CODE XREF: sub_312FD8
; sub_312FD8+59

loc_31306D:
push    0Dh
call    sub_3140F3
; CODE XREF: sub_312FD8
; sub_312FD8+49

loc_31307B:
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C

; CODE XREF: sub_312FD8

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

; CODE XREF: sub_312FD8

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

Basic Exploitation Strategy Returning To UserSpace

Why bother returning to Userspace?

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
mov     eax, [ebp+var_70]
jmp     eax, [ebp+var_84]
b     short loc_31306D
push    esi
push    esi
push    eax
push    edi
mov     [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp     [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Basic Exploitation Strategy Returning To UserSpace

Why bother returning to Userspace?

Most useful things we want to do are *much* easier from userland.

```
push    edi
call    sub_314623
test    eax, eax
jnz    short loc_31306D
mov    eax, [ebp+var_70]
mov    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]
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Basic Exploitation Strategy Returning To UserSpace

Why bother returning to Userspace?

Most useful things we want to do are *much* easier from userland.

In KernelSpace, there's no easy way to:

- > Modify the filesystem
- > Create a new process
- > Create network connections

```
push    edi
call   sub_314623
test   eax, eax
jnz    loc_31306C
mov    eax, [ebp+var_70]
mov    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]
mov    esi, 1D0h
push   esi
push   [ebp+arg_4]
push   edi
call   sub_314623
test   eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Basic Exploitation Strategy Returning To UserSpace

How does the kernel do it?

```
push    edi
call    sub_314623
test    eax, eax
jz      loc_31306D
mov     eax, [ebp+var_70]
jmp     eax, [ebp+var_84]
b      loc_31306D
push    esi
push    esi
push    eax
push    edi
mov     [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz      short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], esi
jz      short loc_31308F

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

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

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

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

Basic Exploitation Strategy Returning To UserSpace

How does the kernel do it?

```
push    $SS_USER_VALUE
push    $USERLAND_STACK
push    $USERLAND_EFLAGS
push    $CS_USER_VALUE
push    $USERLAND_FUNCTION_ADDRESS
swapgs
iretq
```

```
push    edi
call    sub_314623
test    eax, eax
jz     loc_31306D
mov    eax, [ebp+var_70]
mov    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov    [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
loc_313066:          ; CODE XREF: sub_312FD8
; sub_312FD8+59
push    0Dh
call    sub_31411B
loc_31306D:          ; CODE XREF: sub_312FD8
; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
;
loc_31307D:          ; CODE XREF: sub_312FD8
call    sub_3140F3
and     eax, 0FFFFh
or      eax, 80070000h
loc_31308C:          ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Basic Exploitation Strategy Returning To UserSpace

How does the kernel do it?

```
push    $SS_USER_VALUE
push    $USERLAND_STACK
push    $USERLAND_EFLAGS
push    $CS_USER_VALUE
push    $USERLAND_FUNCTION_ADDRESS
swapgs
iretq
```

This *will usually* get you out of “Kernel Mode” safely.

```
push    edi
call    sub_314623
test    eax, eax
jnz    short loc_31306D
mov    eax, [ebp+var_70]
mov    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+59
; sub_312FD8+59
loc_31306D:
push    0Dh
call    sub_31411B
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
loc_31307D:
test    eax, 3140F3
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:
call    sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
```

Basic Exploitation Strategy Returning To UserSpace

For exploitation, the easiest strategy is highjacking execution, and letting the kernel return by itself.

```
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
mov     eax, [ebp+var_70]
mov     eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
[ebp+arg_0], eax
call    sub_31462A
test    eax, eax
short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], esi
jz      short loc_31308F

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

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

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

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

Basic Exploitation Strategy Returning To UserSpace

For exploitation, the easiest strategy is **highjacking** execution, and letting the kernel return by itself.

- > Function Pointer Overwrites
- > Syscall Table Highjacking
- > Use-After-Free

```
push    edi
call    sub_314623
test    eax, eax
jnz    short loc_31306D
mov    eax, [ebp+var_70]
eax, [ebp+var 84]
push    esi
push    esi
push    eax
push    edi
[ebp+arg_0], eax
call    sub_31462A
test    eax, eax
short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
jz    short loc_31308F

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

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

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

loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var 4], eax
```

Basic Exploitation Strategy Returning To UserSpace

For exploitation, the easiest strategy is **highjacking** execution, and letting the kernel return by itself.

- > Function Pointer Overwrites
- > Syscall Table Highjacking
- > Use-After-Free

You need to be very careful about destroying Kernel state.

A **segfault** probably means a **reboot!**

```
push    edi
call    sub_314623
test    eax, eax
jnz    short loc_31306F
mov    eax, [ebp+var_70]
eax, [ebp+var 84]
push    esi
push    esi
push    eax
push    edi
[ebp+arg_0], eax
call    sub_31462A
test    eax, eax
short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
jz    short loc_31308F

loc_313066:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
call    sub_3140F3
test    eax, eax
jg    short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; -----
loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
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
```

Basic Exploitation Strategy

Enjoying our Root Privils

If we make it back to userland, our process should be running with **root** privileges.

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

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

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

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

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

Basic Exploitation Strategy

Enjoying our Root Privils

If we make it back to userland, our process should be running with **root** privileges.

We can do whatever we want!

```
push    edi
call    sub_314623
test    eax, eax
jz      loc_313066
mov     eax, [ebp+var_70]
eax, [ebp+var_84]
add    eax, 1
sub    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
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      loc_31306D
cmp     [ebp+arg_0], esi
jz      loc_31308F

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

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

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

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

Kernel Space Protections

By now, you're familiar with the alphabet soup of exploit mitigations

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jz     short loc_31306D
mov    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call    sub_31462A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
; ----- ; CODE XREF: sub_312FD8
loc_31307D:                                ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h

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

Kernel Space Protections

By now, you're familiar with the alphabet soup of exploit mitigations

DEP

ASLR

PIE

Canaries

RELRO

etc...

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jz     short loc_31306D
mov    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
loc_313066:          ; CODE XREF: sub_312FD8
; sub_312FD8+59
push    0Dh
call    sub_31411B
loc_31306D:          ; CODE XREF: sub_312FD8
; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp    short loc_31308C
;
loc_31307D:          ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
loc_31308C:          ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Kernel Space Protections

By now, you're familiar with the alphabet soup of exploit mitigations

DEP

ASLR

PIE

Canaries

RELRO

etc...

Green: Present in Kernel Space

Yellow: Present, with caveats

Red: Not directly applicable

```
loc_313066: ; CODE XREF: sub_312FD8+59  
; sub_312FD8+5F  
    push 0Dh  
    call sub_31411B  
  
loc_31306D: ; CODE XREF: sub_312FD8+49  
; sub_312FD8+4F  
    call sub_3140F3  
    test eax, eax  
    jg short loc_31307D  
    call sub_3140F3  
    jmp short loc_31308C  
; -----  
  
loc_31307D: ; CODE XREF: sub_312FD8+4F  
; sub_312FD8+59  
    call sub_3140F3  
    and eax, 0FFFFh  
    or  eax, 80070000h  
  
loc_31308C: ; CODE XREF: sub_312FD8+4F  
; sub_312FD8+59  
    mov [ebp+var_4], eax
```

Kernel Space Protections

By now, you're familiar with the alphabet soup of exploit mitigations

DEP

ASLR

PIE

Canaries

RELRO

etc...

Green: Present in Kernel Space

Yellow: Present, with caveats

Red: Not directly applicable

There's a whole new alphabet soup for Kernel Mitigations!

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jz     short loc_31306C
mov    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call    sub_31466A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    edi
push    eax
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F
loc_313066:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
push    0Dh
call    sub_31411B
loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
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
```

Kernel Space Protections

Some new words in our soup

MMAP_MIN_ADDR
KALLSYMS
RANDSTACK
STACKLEAK
SMEP / SMAP

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jz     short loc_31306F
mov    eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
mov    [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Kernel Space Protections

Some new words in our soup (There's plenty more...)

MMAP_MIN_ADDR
KALLSYMS
RANDSTACK
STACKLEAK
SMEP / SMAP

Most of these will be off for the labs!

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

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

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

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

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

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.

```
push    edi
call    sub_314623
test    eax, eax
jz      short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call    sub_31462A
test    eax, eax
jz      short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], esi
jz      short loc_31308F

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

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

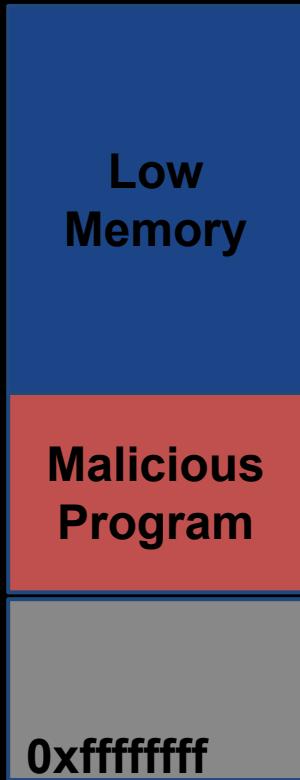
loc_31307D:          ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h

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

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.



```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call   sub_31462A
test   eax, eax
jz    short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
jz    short loc_31308F

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

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

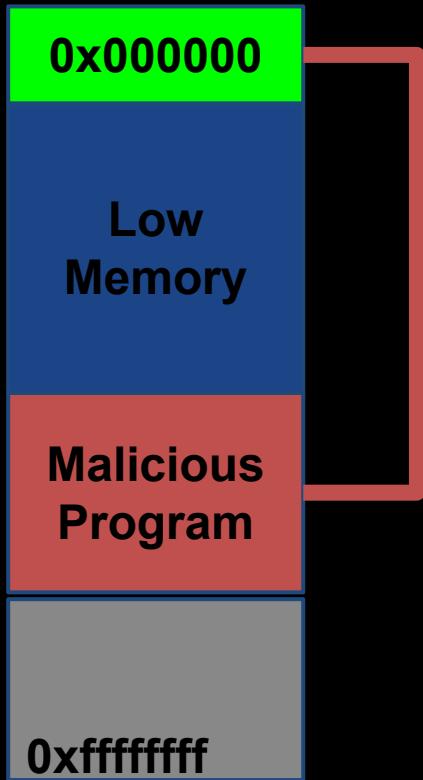
loc_31307D:                                ; CODE XREF: sub_312FD8
call   sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h

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

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.



Program does mmap(0,...)

```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call   sub_31462A
test   eax, eax
jz    short loc_31306D
push    esi
push    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
jz    short loc_31308F

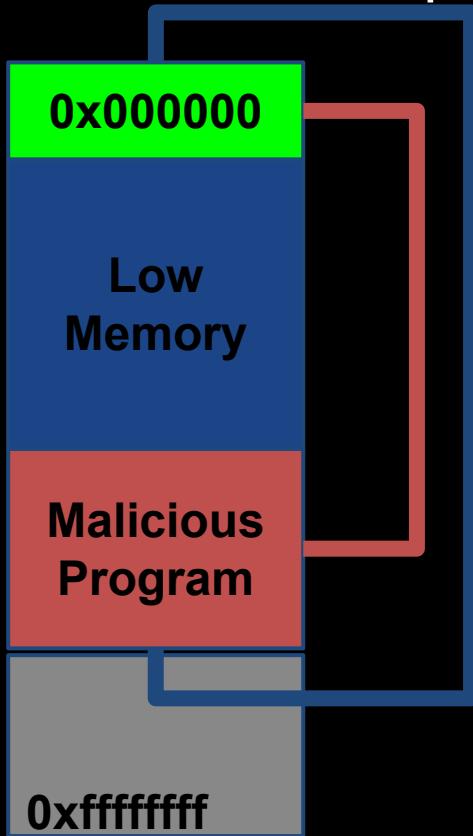
loc_313066:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
push    0Dh
call   sub_31411B

loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call   sub_3140F3
test   eax, eax
jg    short loc_31307D
call   sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:                                ; CODE XREF: sub_312FD8
call   sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
; -----
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.



Program does mmap(0,...)

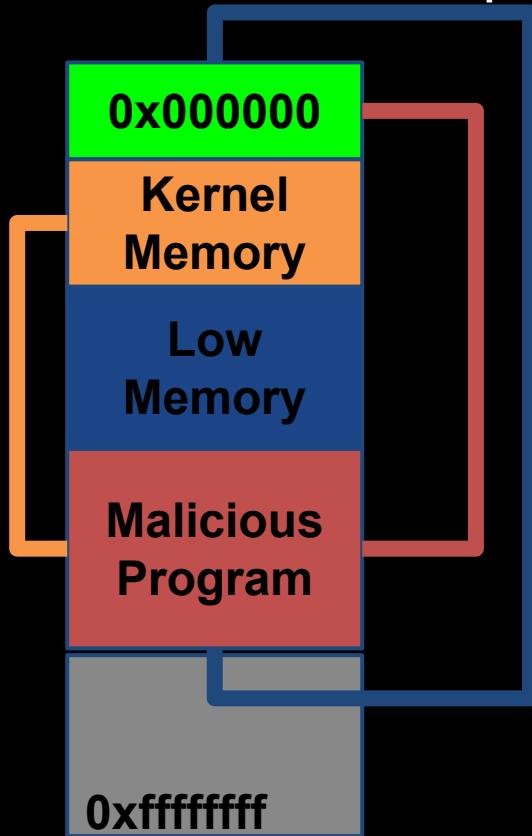
Program writes malicious Code

```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
or    eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call   sub_31466A
test   eax, eax
jz    short loc_31306D
push    esi
push    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
jz    short loc_31308F
loc_313066:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
push    0Dh
call   sub_31411B
loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call   sub_3140F3
test   eax, eax
jg    short loc_31307D
call   sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:                                ; CODE XREF: sub_312FD8
call   sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
loc_31308C:                                ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.



Program does `mmap(0,...)`

Program writes malicious Code

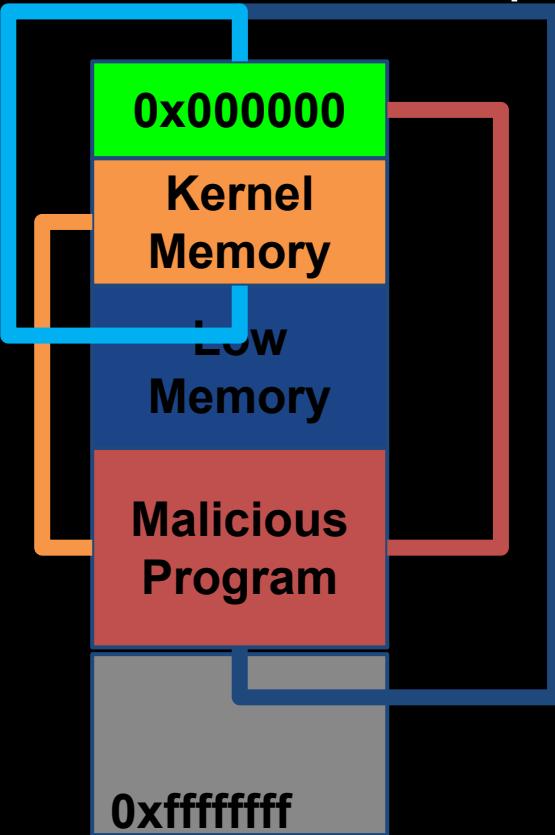
Program triggers Kernel Bug

```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
mov    eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call   sub_31466A
test   eax, eax
jz    short loc_31306D
push    esi
push    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+59
loc_313066:
push    0Dh
call   sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31306D:
call   sub_3140F3
test   eax, eax
jg    short loc_31307D
call   sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:
call   sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8
```

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.



Program does `mmap(0,...)`

Program writes malicious Code

Program triggers Kernel Bug

Kernel starts executing malicious Code

```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
mov    eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call   sub_31462A
test   eax, eax
jz    short loc_31306D
push    esi
push    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    0Dh
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
short loc_31308F
```

; CODE XREF: sub_312FD8
; sub_312FD8+59

```
loc_313066:
push    0Dh
sub_312FD8
; CODE XREF: sub_312FD8
; sub_312FD8+49
```

```
loc_31306D:
push    0Dh
sub_312FD8
; CODE XREF: sub_312FD8
; sub_312FD8+49
```

```
loc_31307D:
call   sub_3140F3
test   eax, eax
jg    short loc_31307D
call   sub_3140F3
jmp    short loc_31308C
;
```

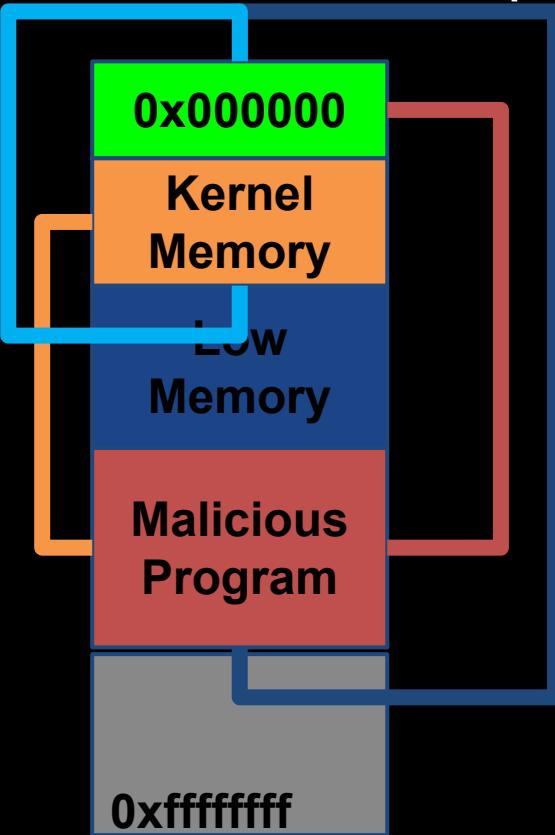
```
loc_31307D:
call   sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
;
```

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

Kernel Space Protections

mmap_min_addr

This makes exploiting **NULL** pointer dereferences harder.



mmap_min_addr disallows programs from allocating low memory.

Makes it much more difficult to exploit a simple **NULL** pointer dereference in the kernel.

```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
or    eax, [ebp+var_84]
short loc_313066
eax, [ebp+var_84]
push    esi
push    esi
push    eax
push    edi
call   sub_31462A
test   eax, eax
jz    short loc_31306D
push    esi
push    [ebp+arg_0]
push    eax
push    edi
push    [ebp+arg_4]
push    edi
call   sub_314623
test   eax, eax
jz    short loc_31306D
cmp    [ebp+arg_0], esi
push    0Fh
push    31411B
; CODE XREF: sub_312FD8+59
; sub_312FD8+59
loc_31306D:
push    0Fh
push    31411B
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
call   sub_3140F3
test   eax, eax
jg    short loc_31307D
call   sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:
call   sub_3140F3
and    eax, 0FFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8+49
; sub_312FD8+49
```

Kernel Space Protections

kallsyms

/proc/kallsyms gives the address of all symbols in the kernel.

We need this information to write reliable exploits without an info-leak!

```
push    edi
call    sub_314623
test    eax, eax
jz     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
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp     [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Kernel Space Protections

kallsyms

/proc/kallsyms gives the address of all symbols in the kernel.

We need this information to write reliable exploits without an info-leak!

```
$: cat /proc/kallsyms | grep commit_creds
```

ffffffff810908c0 T commit_creds

ffffffff81b01390 R __ksymtab_commit_creds

ffffffff81b1cf38 r __kcrctab_commit_creds

ffffffff81b2c33b r __kstrtab_commit_creds

```
push edi
call sub_314623
test eax, eax
jz 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
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+59
push 0Dh
call sub_31411B
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C
;
loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Kernel Space Protections

kallsyms

kallsyms used to be world-readable.

Now, it returns 0's for unprivileged users

```
$: cat /proc/kallsyms | grep commit_creds
```

```
0000000000000000 T commit_creds
```

```
0000000000000000 R __ksymtab_commit_creds
```

```
0000000000000000 r __kcrctab_commit_creds
```

```
0000000000000000 r __kstrtab_commit_creds
```

Can still be a useful source of information on older systems

```
push edi  
call sub_314623  
test eax, eax  
jz loc_313066  
mov eax, [ebp+var_70]  
cmp eax, [ebp+var_84]  
jb short loc_313066  
sub eax, [ebp+var_84]  
push esi  
push esi  
push eax  
push edi  
mov [ebp+arg_0], eax  
call sub_31486A  
test eax, eax  
jz short loc_31306D  
push esi  
lea eax, [ebp+arg_0]  
push eax  
mov esi, 1D0h  
push esi  
push [ebp+arg_4]  
push edi  
call sub_314623  
test eax, eax  
jz short loc_31306D  
cmp [ebp+arg_0], esi  
jz short loc_31308F  
  
loc_313066:  
; CODE XREF: sub_312FD8  
; sub_312FD8+59  
push 0Dh  
call sub_31411B  
  
loc_31306D:  
; CODE XREF: sub_312FD8  
; sub_312FD8+49  
call sub_3140F3  
test eax, eax  
jg short loc_31307D  
call sub_3140F3  
; CODE XREF: sub_312FD8  
; sub_312FD8+1C  
  
loc_31307D:  
; CODE XREF: sub_312FD8  
call sub_3140F3  
and eax, 0FFFh  
or eax, 80070000h  
  
loc_31308C:  
; CODE XREF: sub_312FD8  
mov [ebp+var_4], eax
```

Kernel Space Protections

SMEP / SMAP

SMEP: Supervisor Mode Execution Protection

Introduced in Intel IvyBridge

SMAP: Supervisor Mode Access Protection

Introduced in Intel Haswell

```
push    edi
call    sub_314623
test    eax, eax
jz     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    sub_31486A
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp     [ebp+arg_0], esi
jz     short loc_31308F

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

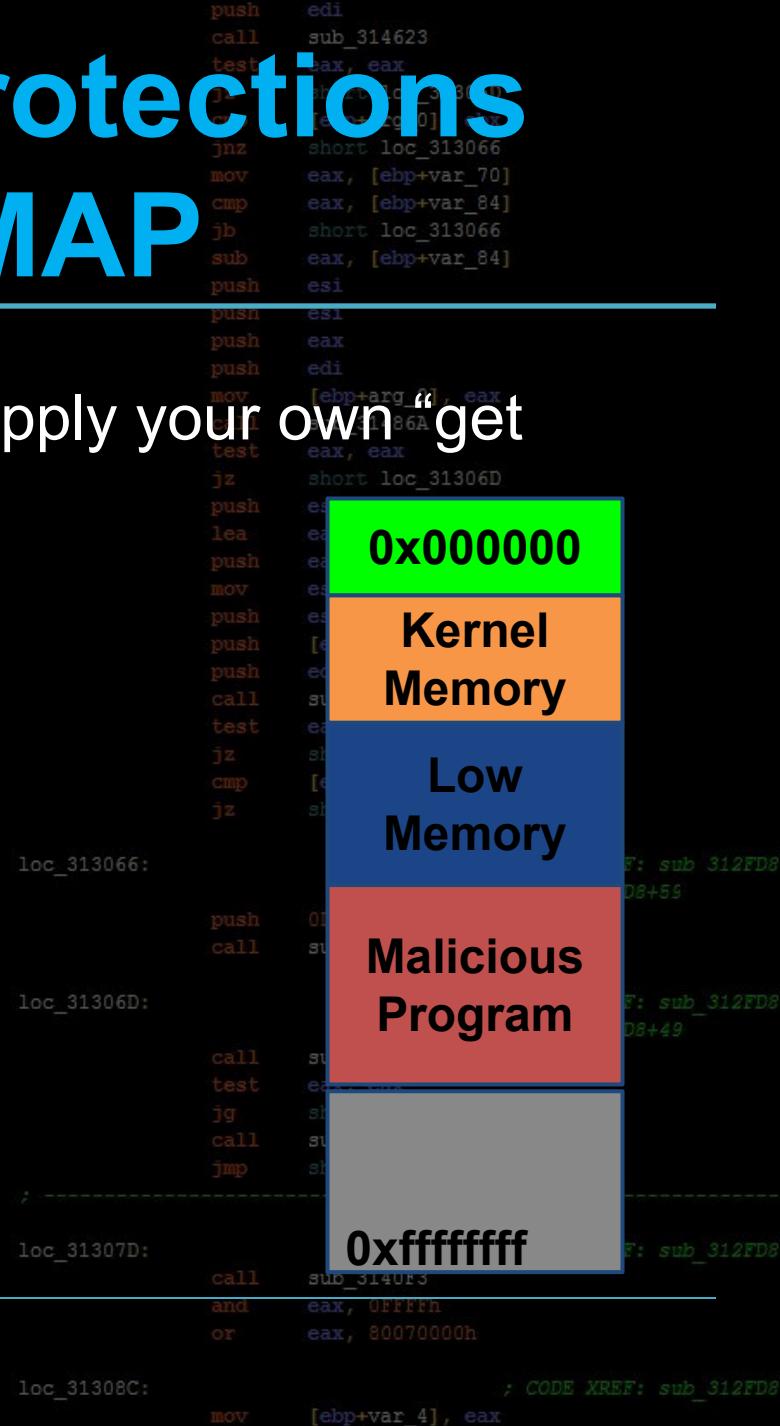
loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
; ----

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

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

Kernel Space Protections SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

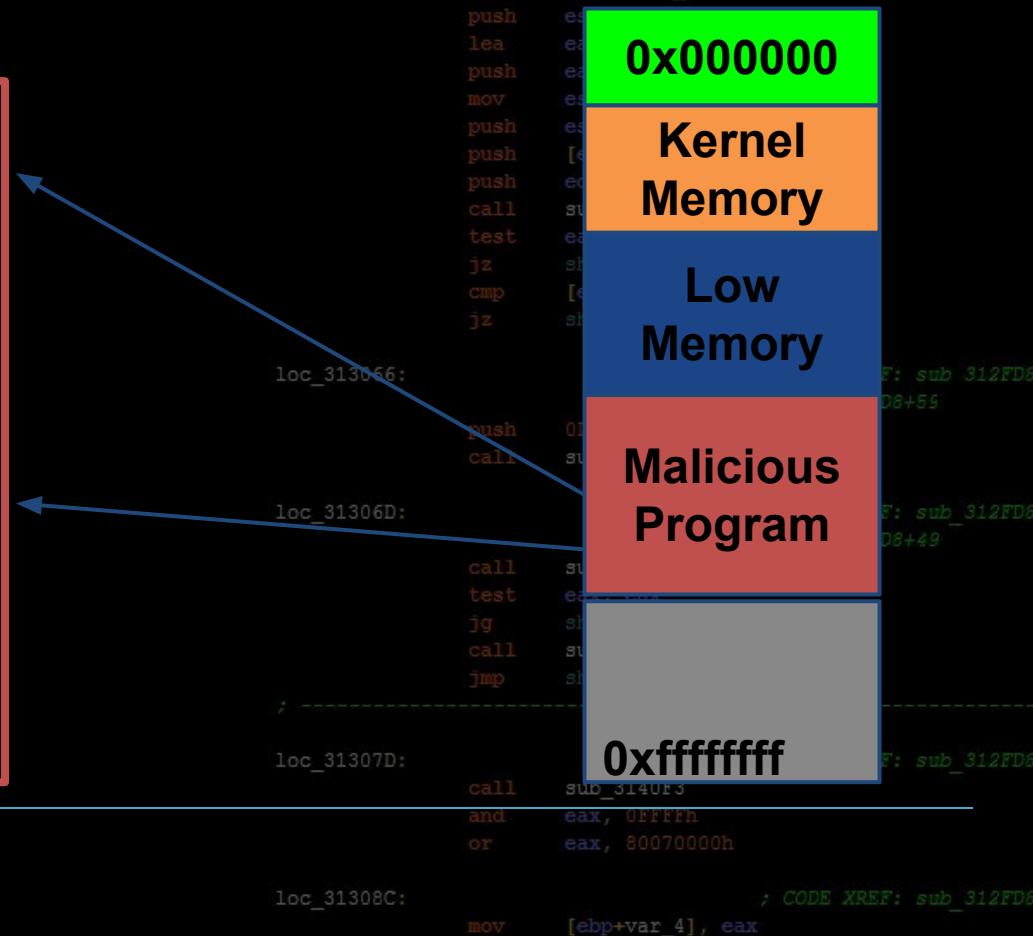


Kernel Space Protections

SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

```
void get_r00t() {  
    commit_creds(prepare_kernel_cred(0));  
}  
  
int main(int argc, char * argv) {  
    ...  
    trigger_fp_overwrite(&get_r00t);  
    ...  
    //trigger fp use  
    trigger_vuln_fp();  
    // Kernel Executes get_r00t  
    ...  
    // Now we have root  
    system("/bin/sh");  
}
```

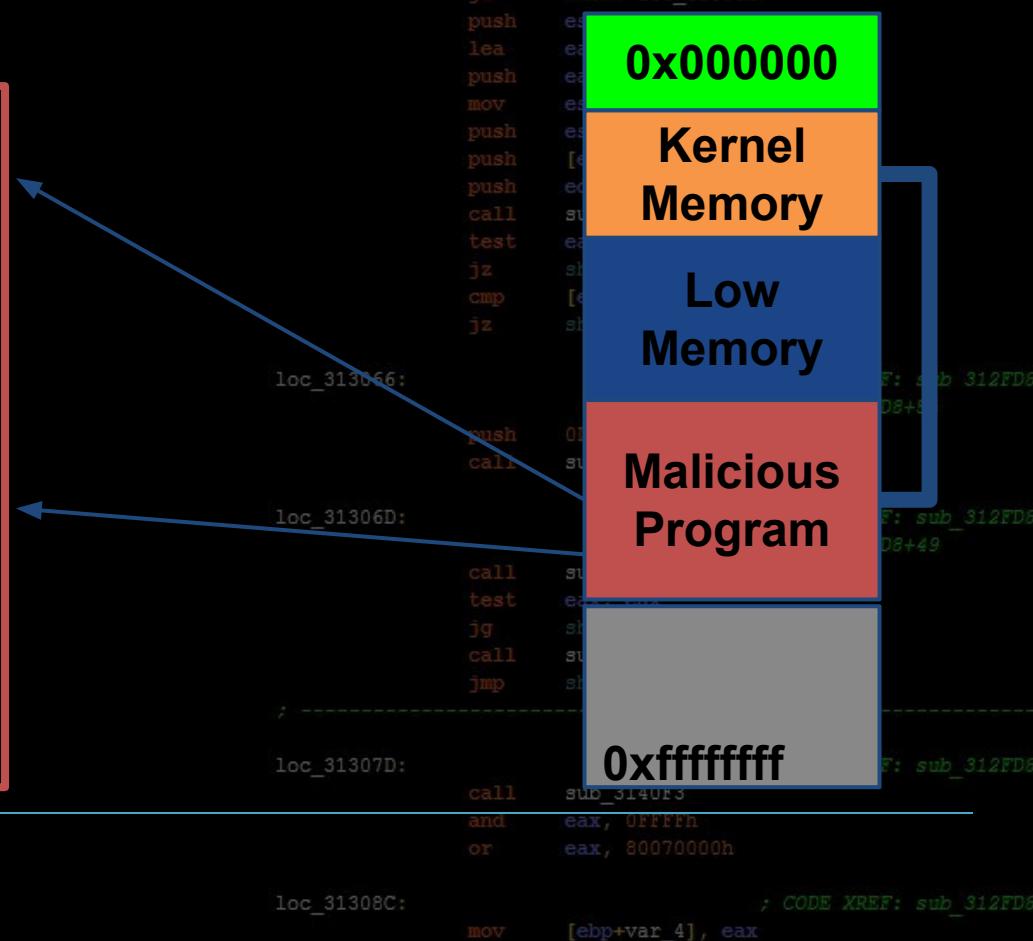


Kernel Space Protections

SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

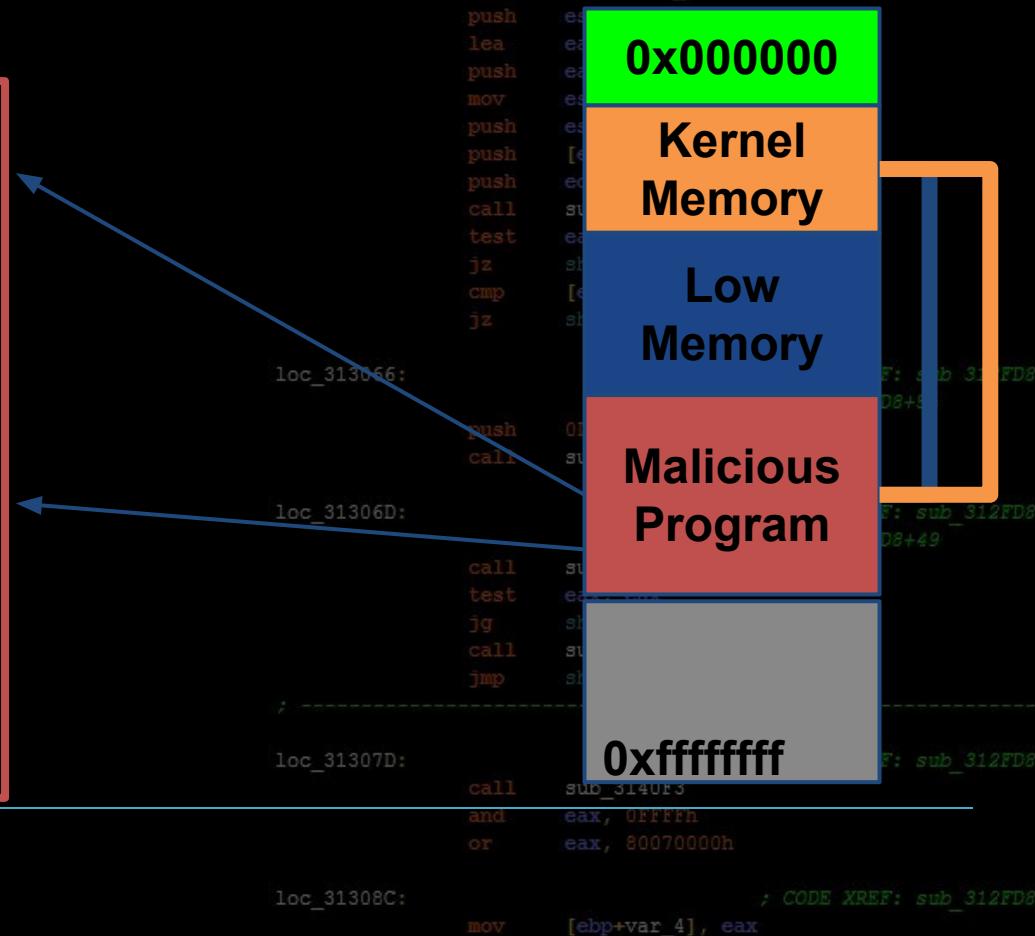
```
void get_r00t() {  
    commit_creds(prepare_kernel_cred(0));  
}  
  
int main(int argc, char * argv) {  
    ...  
    trigger_fp_overwrite(&get_r00t);  
    ...  
    //trigger fp use  
    trigger_vuln_fp();  
    // Kernel Executes get_r00t()  
    ...  
    // Now we have root  
    system("/bin/sh");  
}
```



Kernel Space Protections SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

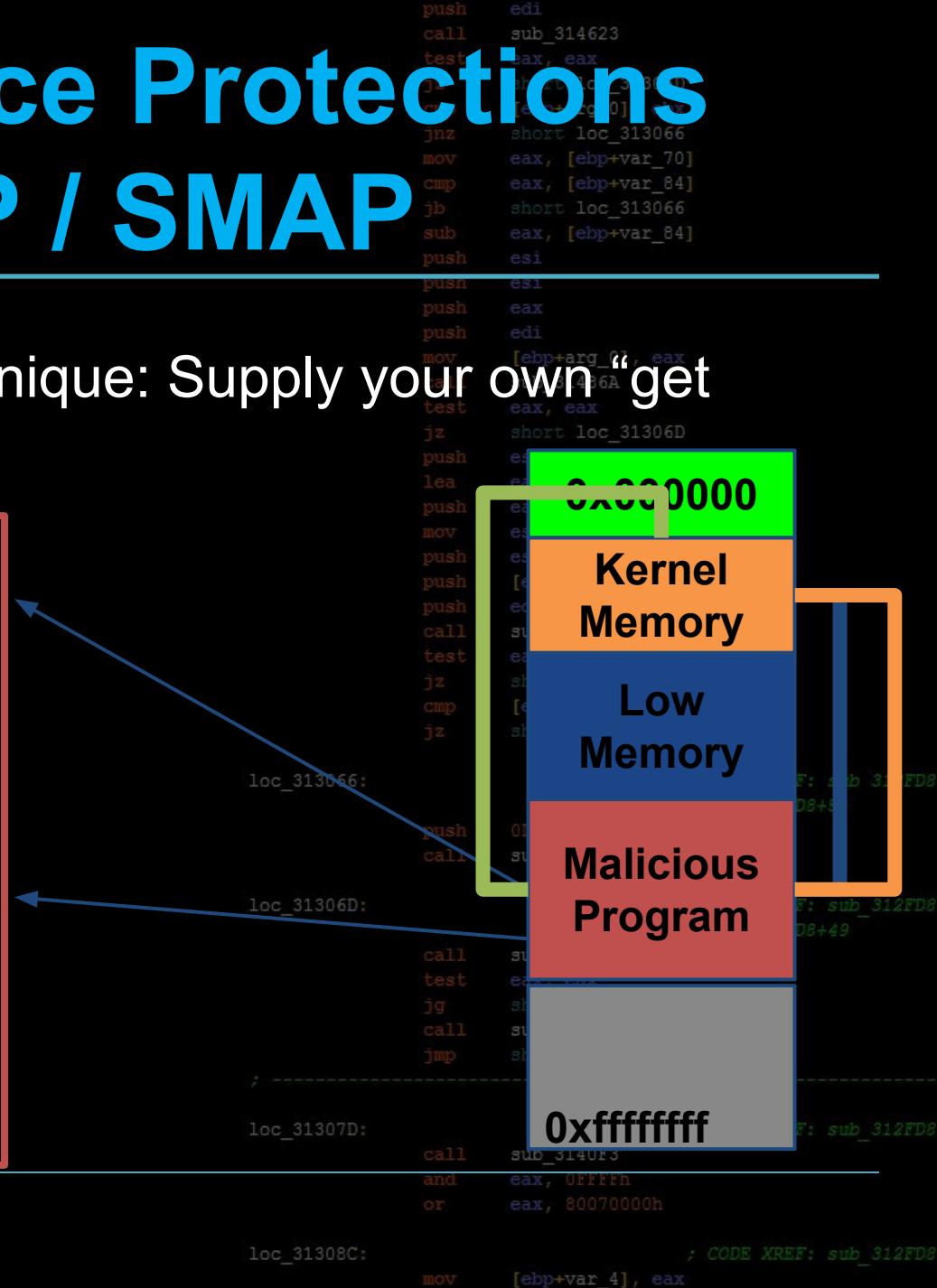
```
void get_r00t() {  
    commit_creds(prepare_kernel_cred(0));  
}  
  
int main(int argc, char * argv) {  
    ...  
    trigger_fp_overwrite(&get_r00t);  
    ...  
    //trigger fp use  
    trigger_vuln_fp();  
    // Kernel Executes get_r00t()  
    ...  
    // Now we have root  
    system("/bin/sh");  
}
```



Kernel Space Protections SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

```
void get_r00t() {  
    commit_creds(prepare_kernel_cred(0));  
}  
  
int main(int argc, char * argv) {  
    ...  
    trigger_fp_overwrite(&get_r00t);  
    ...  
    //trigger fp use  
    trigger_vuln_fp();  
    // Kernel Executes get_r00t()  
    ...  
    // Now we have root  
    system("/bin/sh");  
}
```

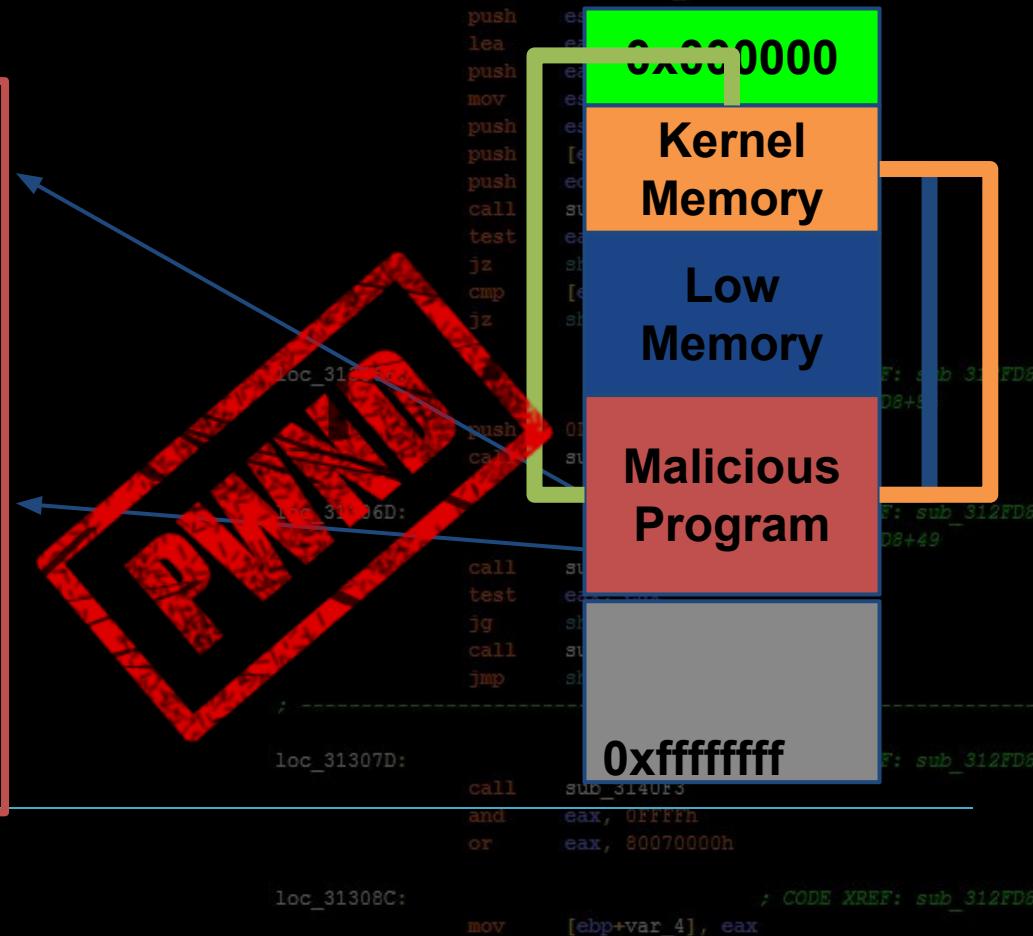


Kernel Space Protections

SMEP / SMAP

Common Exploitation Technique: Supply your own “get root” code.

```
void get_r00t() {  
    commit_creds(prepare_kernel_cred(0));  
}  
  
int main(int argc, char * argv) {  
    ...  
    trigger_fp_overwrite(&get_r00t);  
    ...  
    //trigger fp use  
    trigger_vuln_fp();  
    // Kernel Executes get_r00t()  
    ...  
    // Now we have root  
    system("/bin/sh");  
}
```



Kernel Space Protections

SMEP / SMAP

SMEP prevents this type of attack by triggering a page fault if the processor tries to execute memory that has the “user” bit set while in “ring 0”.

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov     [ebp+arg_0], eax
test    eax, 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+59
push    0Dh
call    sub_31411B

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

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

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

Kernel Space Protections

SMEP / SMAP

SMEP prevents this type of attack by triggering a page fault if the processor tries to execute memory that has the “user” bit set while in “ring 0”.

SMAP works similarly, but for data access in general

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov     [ebp+arg_0], eax
test    eax, eax
jz     short loc_31306D
push    eax
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
je     short loc_31306D
jz     short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+59
push    0Dh
call    sub_31411B
loc_31306D:
; CODE XREF: sub_312FD8
; sub_312FD8+49
call    sub_3140F3
test    eax, eax
jg     short loc_31307D
call    sub_3140F3
jmp     short loc_31308C
; -----
loc_31307D:
; CODE XREF: sub_312FD8
call    sub_3140F3
and     eax, 0FFFFh
or      eax, 80070000h
loc_31308C:
; CODE XREF: sub_312FD8
mov     [ebp+var_4], eax
```

Kernel Space Protections

SMEP / SMAP

SMEP prevents this type of attack by triggering a page fault if the processor tries to execute memory that has the “user” bit set while in “ring 0”.

SMAP works similarly, but for data access in general

This doesn’t *prevent* vulnerabilities, but it adds considerable work to developing a working exploit

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_313066
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jb      short loc_313066
sub     eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov     [ebp+arg_0], eax
test    eax, eax
jz     short loc_31306D
push    eax
pushn   esi
push    [ebp+arg_4]
push    edi
call    sub_314623
je     short loc_31306D
jz     short loc_31306D
cmp     [ebp+arg_0], esi
jz     short loc_31308F
```

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

```
loc_31306D:                                ; CODE XREF: sub_312FD8
                                                ; 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
```

Kernel Space Protections

SMEP / SMAP

SMEP prevents this type of attack by triggering a **page fault** if the processor tries to execute memory that has the “user” bit set while in “ring 0”.

SMAP works similarly, but for data access in general

This doesn’t *prevent* vulnerabilities, but it adds considerable work to developing a working exploit

We need to use **ROP**, or somehow get **executable code** into kernel memory.

```
push    edi
call   sub_314623
test   eax, eax
jz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jb    short loc_313066
sub    eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov    [ebp+arg_0], eax
test   eax, eax
short loc_31306D
push    eax
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call   sub_314623
je    short loc_31306D
[jp]  [ebp+arg_0], esi
jz    short loc_31308F
; CODE XREF: sub_312FD8+59
loc_313066:
push    0Dh
call   sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31306D:
call   sub_3140F3
test   eax, eax
short loc_313070
call   sub_3140F3
jmp    short loc_31308C
; CODE XREF: sub_312FD8+59
loc_31307D:
call   sub_3140F3
and    eax, 0FFFFh
or     eax, 80070000h
; CODE XREF: sub_312FD8
loc_31308C:
mov    [ebp+var_4], eax
; CODE XREF: sub_312FD8+59
```

Example

We'll walk through a short example of a backdoored LKM to get a feel for dealing with the kernel.

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

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

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

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

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

Conclusion

Kernel Exploitation is *weird*, but *extremely powerful*

```
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    sub_1000
test    eax, eax
jz     short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Conclusion

Kernel Exploitation is *weird*, but *extremely powerful*

As userland exploit-dev becomes more challenging and more expensive, kernelspace is becoming a more attractive target.

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jnZ    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov    [ebp+arg_0], eax
sub    sub_1f6d
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
sub    sub_1f6d
push    [ebp+arg_4]
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

Conclusion

Kernel Exploitation is *weird*, but *extremely powerful*

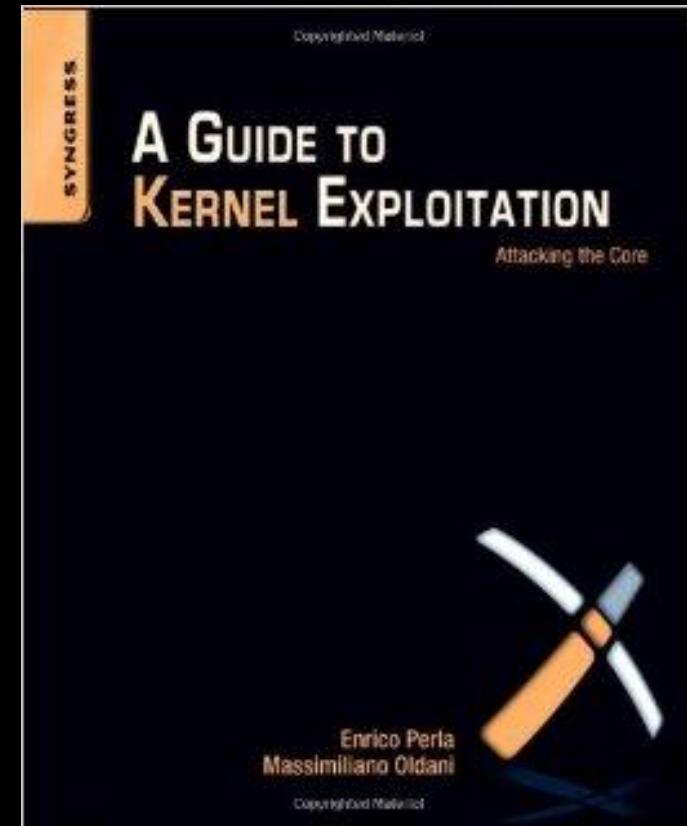
As userland exploit-dev becomes more challenging and more expensive, kernelspace is becoming a more attractive target.

A single bug can be used to bypass sandboxes, and gain root privileges, which may otherwise be impossible

```
push    edi
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], ebx
jnZ    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jb     short loc_313066
sub    eax, [ebp+var_84]
push    esi
pushn   esi
push    eax
push    edi
mov    [ebp+arg_0], eax
sub    sub_1f6d
test    eax, eax
jz     short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
sub    sub_1f6d
push    [ebp+arg_4]
call    sub_314623
test    eax, eax
jz     short loc_31306D
cmp    [ebp+arg_0], esi
jZ     short loc_31308F
; CODE XREF: sub_312FD8
; sub_312FD8+59
push    0Dh
call    sub_31411B
loc_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
```

Conclusion

The book on Kernel 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
```

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

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

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