

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 eax
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
```

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

```
push 0Dh
call sub_31411B
```

```
loc_313069:                                     ; CODE XREF: sub_312FD8
; sub_312FD8+49
```

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

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

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

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

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

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+55
push    0Dh
call    sub_31411B

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

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

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

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
```

**FILES IN A FOLDER CALLED
TEMPORARY**

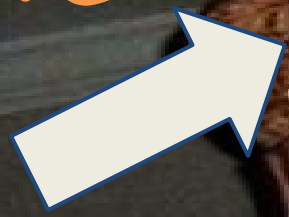
ARE TEMPORARY

You probably feel like this

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

TELLS YOU TO PUT YOUR STUFF IN /TMP

MBETA'S



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

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

```
push   0Dh
call   sub_31411B
```

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

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

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

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

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

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

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

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

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

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

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

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]
call 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
call [ebp+arg_4]
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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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

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
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jnb 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
mov [ebp+arg_4], esi
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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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


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]
call 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
call [ebp+arg_4]
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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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

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
call sub_31465A
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]
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F
```

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

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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

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_31465A
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]
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F
```

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

```
push 0Dh
call sub_31411B
```

```
loc_31306B: ; CODE XREF: sub_312FD8
; sub_312FD8+49
```

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

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

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

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

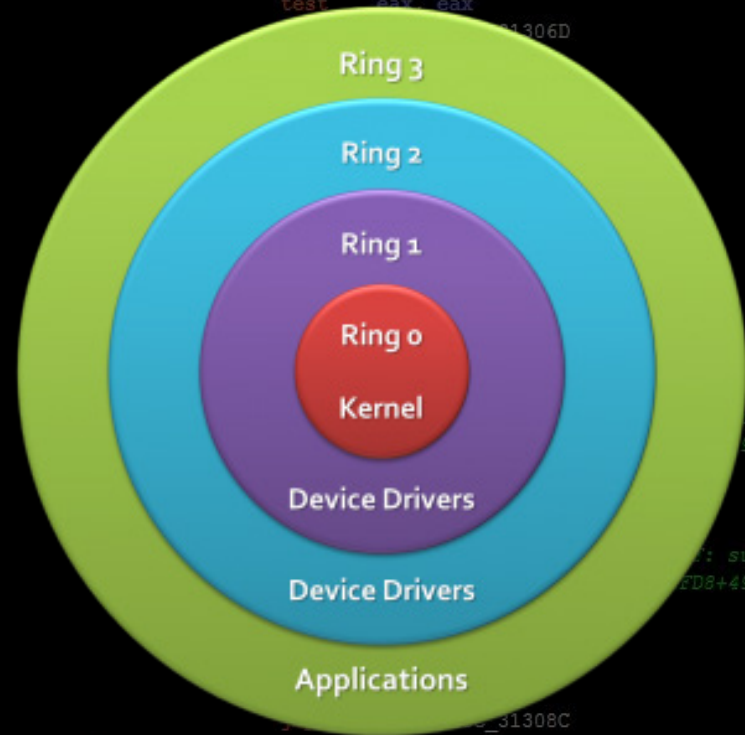
```
mov [ebp+var_4], eax
```

What's a Kernel? - Ring Model

Hardware Enforced Model

0: Privileged, Kernelspace

3: Restricted, Userspace



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

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

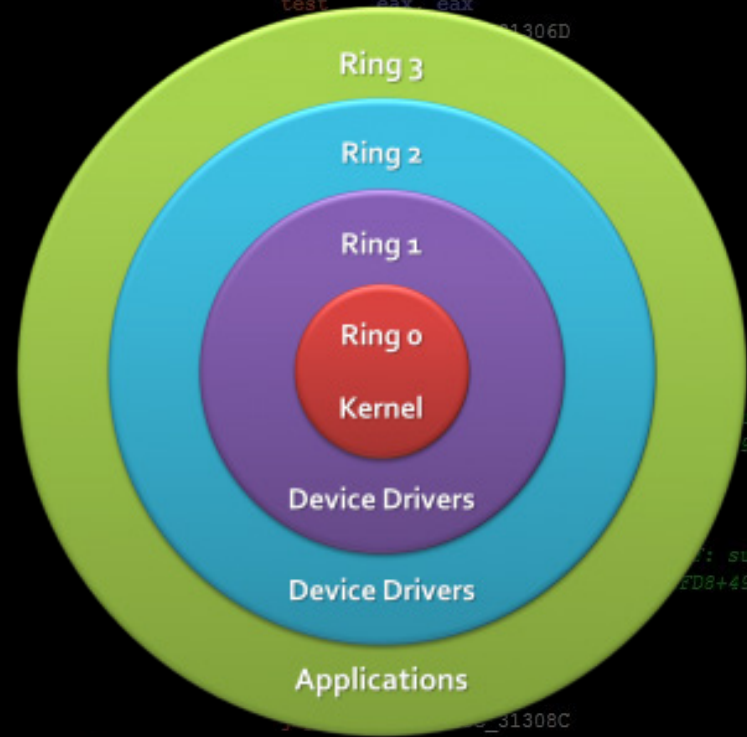
What's a Kernel? - Ring Model

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)

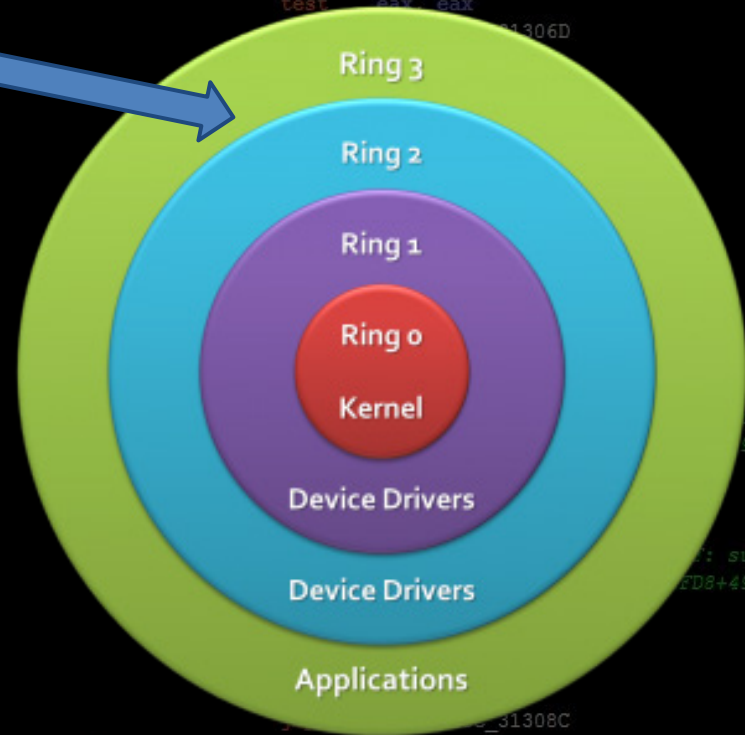


```
loc_31307D:          call     sub_3140F3          ; CODE XREF: sub_312FD8
                    and     eax, 0FFFFFFh
                    or      eax, 80070000h

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

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_0], ebx
nz short loc_313066
xor eax, eax
mov [ebp+var_70], eax
mov [ebp+var_84], eax
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
```

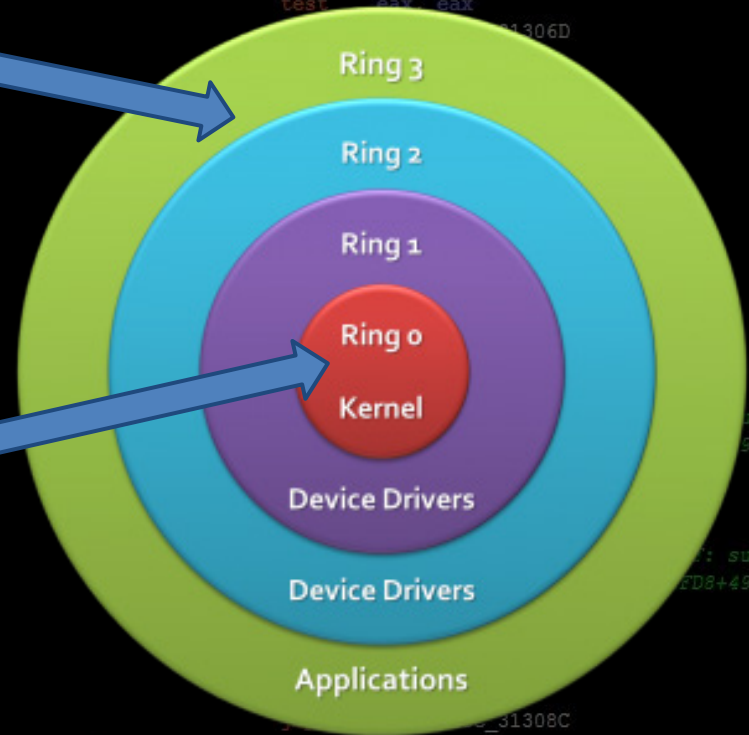
```
loc_31307D: call sub_3140F3 ; CODE XREF: sub_312FD8
and eax, 0FFFFFFh
or eax, 80070000h

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

What's a Kernel? - Ring Model

We've Been Here

We're Going Here



```
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 [ebp+var_84], eax
jnb 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
```

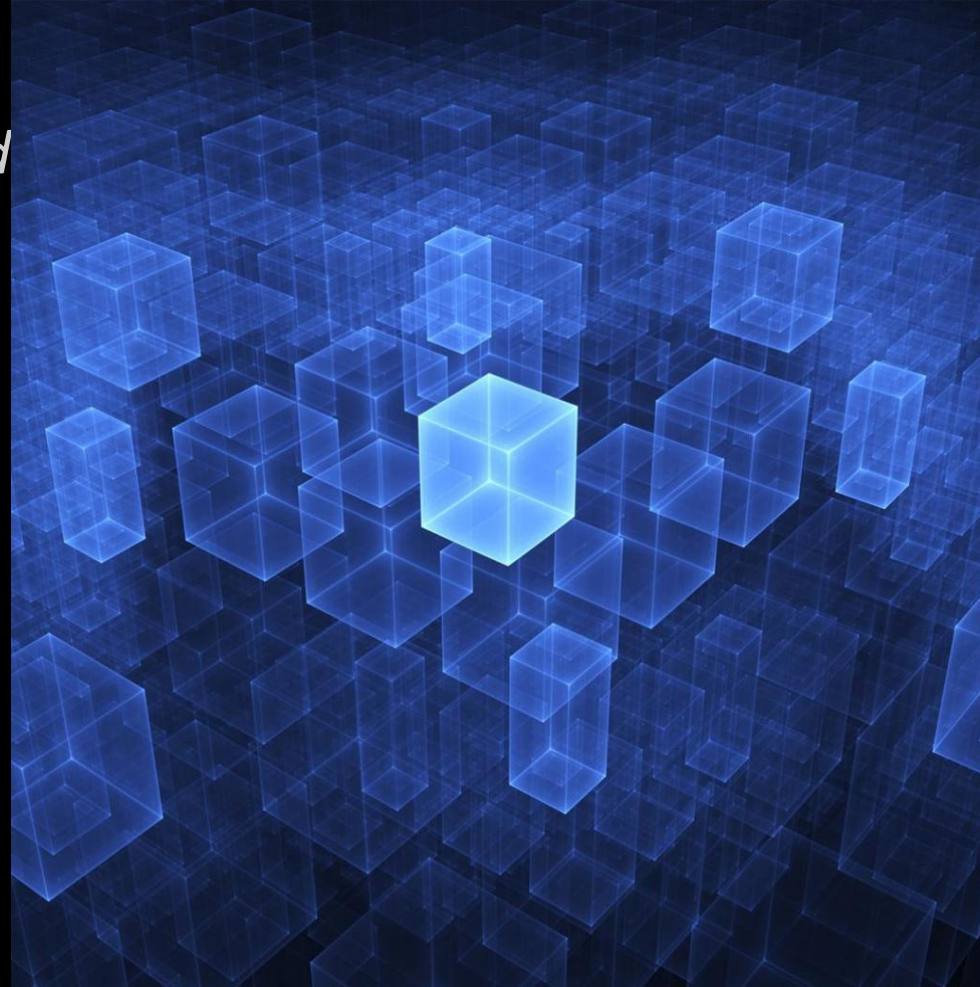
```
loc_31307D: call sub_3140F3 ; CODE XREF: sub_312FD8
and eax, 0FFFFFFh
or eax, 80070000h

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

Obligatory Matrix Analogy

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

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



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

loc_31308C:

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

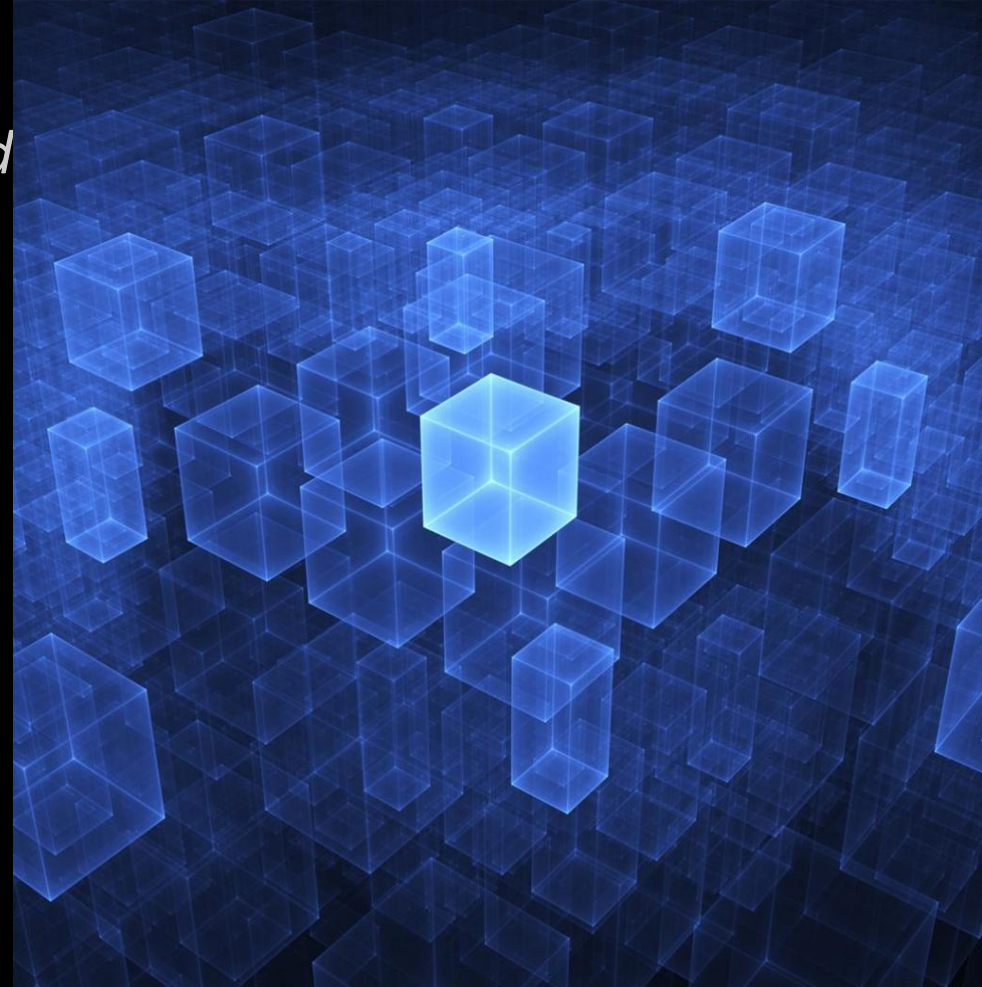
; CODE XREF: sub_312FD8

Obligatory Matrix Analogy

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

“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



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

loc_31308C:

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

; CODE XREF: sub_312FD8

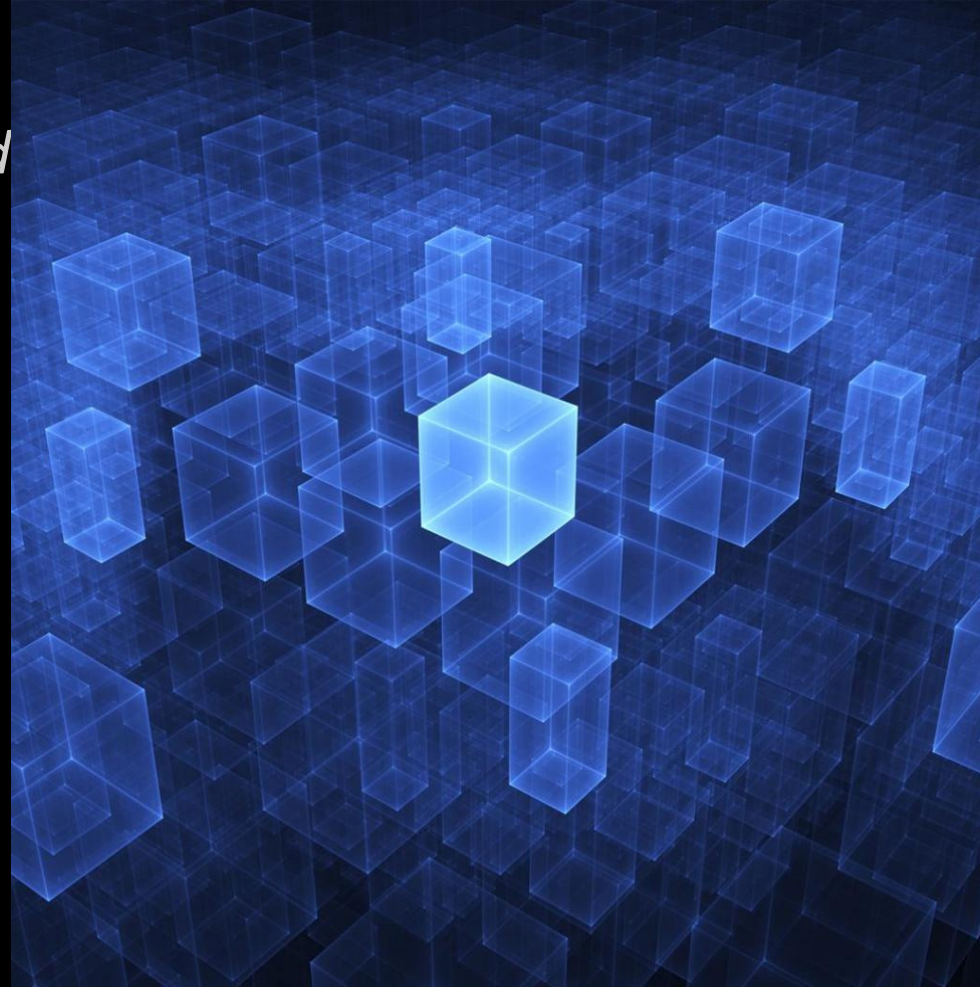
Obligatory Matrix Analogy

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

“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



```
and eax, 0FFFFFFh
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



```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnc short loc_313066
mov eax, [ebp+arg_7]
cmp [ebp+var_84], eax
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, [ebp+arg_0]
push esi
push [ebp+arg_0]
push edi
sub_31486A
eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jz short loc_31306D
loc_313066:
push 0Dh
sub_31411B
loc_31307D:
; CODE XREF: sub_312FD8
; sub_312FD8+49
sub_3140F3
loc_31307D:
; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

Kernel Pwning in Popular Culture

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

Remember JailbreakMe?



```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnz short loc_313066
mov eax, [ebp+arg_7]
cmp [ebp+var_84], eax
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+var_84]
push eax
mov esi, [ebp+var_84]
push esi
push [ebp+var_84]
push edi
sub_31486A
eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jz short loc_31306D
loc_313066:
push 0Dh
sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
; CODE XREF: sub_312FD8
; sub_312FD8+49
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31307D:
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
```

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



```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
jnc short loc_313066
mov eax, [ebp+arg_7]
cmp [ebp+var_84], eax
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+var_84]
push eax
mov esi, [ebp+var_84]
push esi
push [ebp+var_84]
push edi
sub_31486A
eax, eax
jz short loc_31306D
cmp [ebp+arg_0], [ebp+arg_0]
jz short loc_31306D
loc_313066:
push 0Dh
sub_31411B
; CODE XREF: sub_312FD8
; sub_312FD8+49
; CODE XREF: sub_312FD8
; sub_312FD8+49
; CODE XREF: sub_312FD8
; sub_312FD8+49
loc_31307D:
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
```

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+55
push    0Dh
call    sub_31411B

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

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

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

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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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

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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

```
call sub_3140F3
and eax, 0FFFFFFh
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
cmp eax, [ebp+var_70]
cmpr 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_314623
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

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

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]
cmp eax, [ebp+var_84]
jbe short loc_313066
sub eax, [ebp+var_84]
push esi
push esi
push eax
push edi
mov [ebp+arg_0], eax
call sub_314623
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jnz short loc_313066
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0Dh
call sub_31411B
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C
; -----
loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

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

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

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

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

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

Basic Exploitation Strategy

The Vulnerabilities

You already know how to find these!

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

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

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

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

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

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
jnz    short loc_31306D
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jnz    short loc_313066
push   esi
push   esi
push   eax
push   edi
call   sub_31486A, eax
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+55
push   0Dh
call   sub_31411B

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

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

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

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

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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

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

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

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

; -----
loc_31307D:                                     ; CODE XREF: sub_312FD8
call   sub_3140F3
and   eax, 0FFFFFFh
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)**.

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
jnz short loc_31306D
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
push esi
push esi
push eax
push edi
call sub_31486A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push [ebp+var_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

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


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
jnz short loc_31306D
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
mov eax, [ebp+var_84]
push esi
push esi
push eax
push edi
mov [ebp+arg_0], eax
call sub_31426A
test eax, eax
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

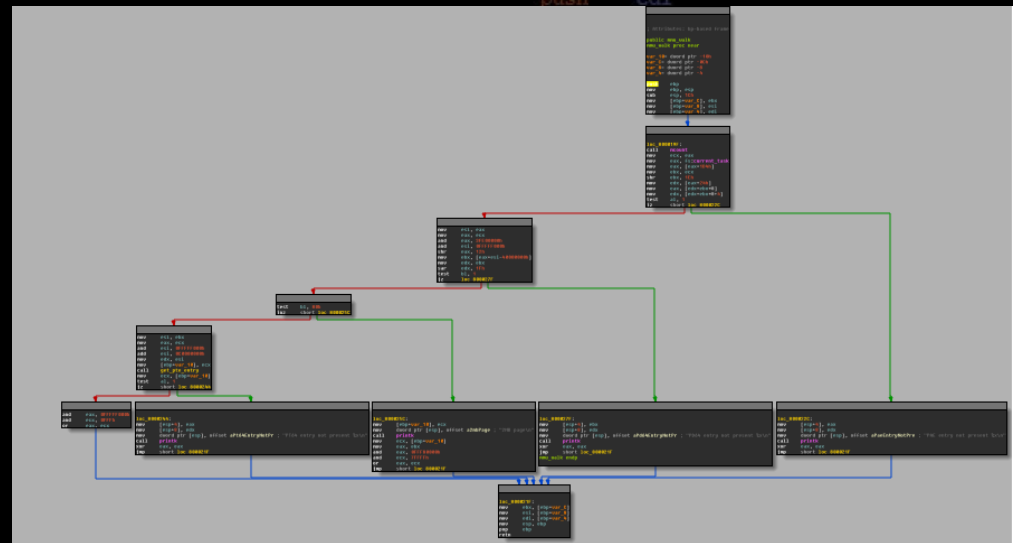
loc_31308C: ; CODE XREF: sub_312FD8
mov [ebp+var_4], eax
```

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)

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



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

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

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

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

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

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

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

loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
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
jnz short loc_31306D
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
mov eax, [ebp+var_84]
push esi
push esi
push eax
push edi
call sub_31486A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

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

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
jz      short loc_31306D
push    esi
mov     eax, [ebp+var_70]
cmp     eax, [ebp+var_84]
jnz     short loc_313066
mov     esi, [ebp+var_84]
push   esi
push   esi
push   eax
push   edi
call   sub_31486A
test   eax, eax
jz     short loc_31306D
push   esi
lea   eax, [ebp+arg_0]
push  eax
mov   esi, 1D0h
push  [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+55
push  0Dh
call  sub_31411B

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

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

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

Basic Exploitation Strategy

Gaining Code Execution

You already know how to do this too!

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

loc_313066:
; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0Dh
call sub_31486A

loc_31306D:
; CODE XREF: sub_312FD8+9
; sub_312FD8+9
push 0
call sub_31486A

loc_313071:
; CODE XREF: sub_312FD8+13
; sub_312FD8+13
push 0
call sub_31486A

loc_313074:
; CODE XREF: sub_312FD8+17
; sub_312FD8+17
push 0
call sub_31486A

loc_313077:
; CODE XREF: sub_312FD8+1B
; sub_312FD8+1B
push 0
call sub_31486A

loc_31307A:
; CODE XREF: sub_312FD8+1F
; sub_312FD8+1F
push 0
call sub_31486A

loc_31308C:
; CODE XREF: sub_312FD8+23
; sub_312FD8+23
mov [ebp+var_4], eax
and eax, 0FFFFFFh
or eax, 80070000h
```



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
jnz short loc_31306D
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jz short loc_313066
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_313066
cmp [ebp+arg_0], [ebp+arg_0]
jz short loc_313066

loc_313066:
; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0Dh
call sub_314623

loc_313069:
; CODE XREF: sub_312FD8
; sub_312FD8+59
push 0Dh
call sub_314623

loc_313071:
; CODE XREF: sub_312FD8
; sub_312FD8+5F
call sub_3140F3
and eax, 0FFFFFFh
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!

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
jnz short loc_31306D
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jz short loc_313066
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_313066
cmp [ebp+arg_0], [ebp+arg_4]
jz short loc_313066
loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0Dh
call sub_314623
loc_313069: ; CODE XREF: sub_312FD8
; sub_312FD8+9
push 0
test eax, eax
jz short loc_313066
loc_313071: ; CODE XREF: sub_312FD8
; sub_312FD8+1F
call sub_3140F3
and eax, 0FFFFFFh
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!

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
jnz short loc_31306D
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jz short loc_313066
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_313066
cmp [ebp+arg_0], [ebp+arg_4]
jz short loc_313066
loc_313066:
; CODE XREF: sub_312FD8
; sub_312FD8+55
push 0Dh
call sub_314623
loc_313067:
; CODE XREF: sub_312FD8
; sub_312FD8+9
push 9
call sub_314623
loc_313071:
; CODE XREF: sub_312FD8
; sub_312FD8+13
push 13
call sub_314623
loc_31308C:
; CODE XREF: sub_312FD8
; sub_312FD8+17
mov [ebp+var_4], eax
and eax, 0FFFFFFh
or eax, 80070000h
```

Basic Exploitation Strategy

Gaining Code Execution

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

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

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

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

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

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

Basic Exploitation Strategy

Gaining Code Execution

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

printf()
memcpy()
malloc()

--->
--->
--->

printfk()
copy_from_user()/copy_to_user()
kmallocc()

```
push    edi
call   sub_314623
test   eax, eax
jz     short loc_31306D
mov    ebx, [ebp+arg_0]
jnz   short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jbe   short loc_313066
sub    esi, [ebp+var_84]
push   esi
push   esi
push   eax
push   edi
call   sub_31486A
test   eax, eax
jz     short loc_31306D
push   esi
lea   eax, [ebp+arg_0]
push   eax
mov    esi, 1D0h
push   esi
push   [ebp+arg_4]
push   edi
call   sub_314623
test   eax, eax
jz     short loc_31306D
push   esi
jz     short loc_31306F
loc_313066:                                     ; CODE XREF: sub_312FD8
; sub_312FD8+55
push   0Dh
call   sub_31411B
loc_31306D:                                     ; CODE XREF: sub_312FD8
; sub_312FD8+49
call   sub_3140F3
test   eax, eax
jg     short loc_31307D
call   sub_3140F3
jmp    short loc_31308C
; -----
loc_31307D:                                     ; CODE XREF: sub_312FD8
call   sub_3140F3
and    eax, 0FFFFFFh
or     eax, 80070000h
loc_31308C:                                     ; CODE XREF: sub_312FD8
mov    [ebp+var_4], eax
```

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
jz short loc_31306D
mov eax, [ebp+var_70]
mov eax, [ebp+var_84]
push esi
push esi
push eax
push edi
call sub_31486A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
jz short loc_31306F
```

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

loc_313068: ; 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, 0FFFFFFh
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

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

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

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

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

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

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

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

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

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

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

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
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_31486A
test  eax, eax
jz   short loc_31306D
jz   short loc_31308F
```

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

```
push    0Dh
call   sub_31411B
```

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

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

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

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

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

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

Basic Exploitation Strategy

Gaining Code Execution

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

```
push    esi
call   sub_314623
test   eax, eax
jnz    short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jnz    short loc_313066
push   ebp
push   esi
push   esi
PID: 0      TASK: ffff81121fff987b0 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: 0000000af482de4 R12: ffff81091fe27950
R13: 0000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG RAX: ffffffff80000000 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: ffff81011ffee000 R9: 000000000000003f
R10: ffff81091fdc4008 R11: 0000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 0000000000000000 R15: ffff81118d6700c0
ORIG RAX: ffffffff80000000 CS: 0010 SS: 0018
#12 [ffff81011ffefea0] acpi_processor_idle_simple at ffffffff801a6b29
```

```
and    eax, 0FFFFFFFh
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
jnz short loc_313066
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jnz short loc_313066
push esi
push esi
PID: 0 TASK: ffff81121fff987b0 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: 0000000af482de4 R12: ffff81091fe27950
R13: 0000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG RAX: ffffffff80200367 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: ffff81011ffefee0 R9: 000000000000003f
R10: ffff81091fdc4008 R11: 0000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 0000000000000000 R15: ffff81118d6700c0
ORIG RAX: ffffffff801a62ab CS: 0010 SS: 0018
#12 [ffff81011ffefea0] acpi_processor_idle_simple at ffffffff801a6b29
```

```
and eax, 0FFFFFFFh
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

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

```
push    edi
call   sub_314623
test   eax, eax
jnz    short loc_31306D
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jnz    short loc_313066
push   esi
push   esi
PID: 0      TASK: ffff81121fff987b0 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: 0000000af482de4 R12: ffff81091fe27950
R13: 0000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG RAX: ffffffff80200367 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: ffff81011ffefee0 R9: 000000000000003f
R10: ffff81091fdc4008 R11: 0000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 0000000000000000 R15: ffff81118d6700c0
ORIG RAX: ffffffff801a62ab CS: 0010 SS: 0018
#12 [ffff81011ffefea0] acpi_processor_idle_simple at ffffffff801a6b29
```

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

loc_31308C:

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

; CODE XREF: sub_312FD8

Basic Exploitation Strategy

Elevate Privileges

Remember: The Kernel manages running processes

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

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

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

loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
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
jnz short loc_313066
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
push esi
push esi
push eax
push edi
call sub_31486A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

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

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  
jnz short loc_313066  
mov eax, [ebp+var_70]  
cmp eax, [ebp+var_84]  
jnz short loc_313066  
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+55
```

```
push 0Dh  
call sub_31411B
```

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

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

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

```
push    edi
call   sub_314623
test   eax, eax
jnz   short loc_313066
mov    eax, [ebp+var_70]
cmp    eax, [ebp+var_84]
jnz   short loc_313066
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+55
push   0Dh
call   sub_31411B

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

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

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

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  
jnz     short loc_313066  
mov     eax, [ebp+var_70]  
cmp     eax, [ebp+var_84]  
jnz     short loc_313066  
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+55  
push    0Dh  
call    sub_31411B  
  
loc_31306D:                                     ; CODE XREF: sub_312FD8  
                                               ; sub_312FD8+49  
call    sub_3140F3  
test    eax, eax  
jg      short loc_31307D  
call    sub_3140F3  
jmp     short loc_31308C  
-----  
loc_31307D:                                     ; CODE XREF: sub_312FD8  
call    sub_3140F3  
and     eax, 0FFFFFFh  
or      eax, 80070000h  
  
loc_31308C:                                     ; CODE XREF: sub_312FD8  
mov     [ebp+var_4], eax
```

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  
jnz     short loc_313066  
mov     eax, [ebp+var_70]  
cmp     eax, [ebp+var_84]  
jnz     short loc_313066  
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+55  
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, 0FFFFFFh  
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  
jnz   short loc_313066  
mov    eax, [ebp+var_70]  
cmp    eax, [ebp+var_84]  
jnz   short loc_313066  
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, eax  
mov    esi, 1D0h  
push   esi  
push   [ebp+arg_4]  
push   edi  
call   sub_314623  
test   eax, eax  
jz    short loc_31306D  
cmp    [ebp+arg_0], esi  
jz    short loc_31308F  
  
loc_313066:                                     ; CODE XREF: sub_312FD8  
                                             ; sub_312FD8+55  
push   0Dh  
call   sub_31411B  
  
loc_31306D:                                     ; CODE XREF: sub_312FD8  
                                             ; sub_312FD8+49  
call   sub_3140F3  
test   eax, eax  
jg    short loc_31307D  
call   sub_3140F3  
jmp   short loc_31308C  
-----  
loc_31307D:                                     ; CODE XREF: sub_312FD8  
call   sub_3140F3  
and    eax, 0FFFFFFh  
or     eax, 80070000h  
  
loc_31308C:                                     ; CODE XREF: sub_312FD8  
mov    [ebp+var_4], eax
```

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  
jnz   short loc_313066  
mov    eax, [ebp+var_70]  
cmp    eax, [ebp+var_84]  
jnz   short loc_313066  
push   esi  
push   esi  
push   eax  
push   edi  
mov    [ebp+arg_0], eax  
call   sub_31486A  
test   eax, eax  
jz    short loc_31306D  
push   esi  
mov    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+55
```

```
call   sub_31411B
```

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

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

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

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

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

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

Basic Exploitation Strategy

Elevate Privileges

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

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

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

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

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

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

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

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

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

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

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
jnz short loc_313066
mov eax, [ebp+var_70]
cmp eax, [ebp+var_84]
jnz short loc_313066
push eax, [ebp+var_84]
push esi
push esi
push eax
push edi
call [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
call [ebp+arg_0], esi
jz short loc_31308F

loc_313066: ; CODE XREF: sub_312FD8
; sub_312FD8+55
push esi
mov esi, 1D0h
call sub_31411B

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

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

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

Basic Exploitation Strategy

Returning To UserSpace

Why bother returning to **Userspace**?

```
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

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

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

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

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   esi, [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+55
push  0Dh
call  sub_31411B

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

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

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

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

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

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

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

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


Basic Exploitation Strategy

Returning To UserSpace

How does the kernel do it?

```
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

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

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

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

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

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
jnz    short loc_313066
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+55
push  0Dh
call  sub_31411B

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

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

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

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
```

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

```
push    0Dh
call    sub_31411B
```

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

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

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

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

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

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

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
jnz short loc_313066
mov eax, [ebp+var_70]
mov [ebp+var_84], eax
push esi
push esi
push eax
push edi
call [ebp+arg_0], eax
call sub_31306A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

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

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_313066
mov eax, [ebp+var_70]
mov [ebp+var_84], eax
push esi
push esi
push eax
push edi
call [ebp+arg_0], eax
call sub_31306A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

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

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

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

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_31306D
mov eax, [ebp+var_70]
mov eax, [ebp+var_84]
push esi
push esi
push eax
push edi
call [ebp+arg_0], eax
call sub_31306A
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F
```

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

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

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

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

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

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

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

Basic Exploitation Strategy

Enjoying our Root Privs

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

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

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

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

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

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

Basic Exploitation Strategy

Enjoying our Root Privs

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
jnz     short loc_313066
mov     eax, [ebp+var_70]
call    sub_314866
push    esi
push    esi
push    eax
push    edi
call    sub_31486A
test    eax, eax
jz      short loc_31306D
push    esi
lea    eax, [ebp+arg_0]
push    eax
mov    esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp    [ebp+arg_0], esi
jz      short loc_31308F

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

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

; -----
loc_31307D:                                ; CODE XREF: sub_312FD8
call    sub_3140F3
and    eax, 0FFFFFFh
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
jg      short loc_313066
mov     eax, [ebp+var_84]
jnb     short loc_313066
sub     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    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], esi
jz      short loc_31308F

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

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

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

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

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

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

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

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

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

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

```
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], ebx
mov [ebp+var_84], ebx
mov eax, [ebp+var_84]
jb short loc_313066
sub 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 [ebp+arg_0]
push edi
call sub_31466A
test eax, eax
jz short loc_31306D
cmp [ebp+var_84], esi
jz short loc_31308F

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

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

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

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

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
jg short loc_313066
mov eax, [ebp+var_84]
jnb short loc_313066
sub 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 [ebp+arg_0]
push edi
call sub_31466A
test eax, eax
jz short loc_31306D
cmp [ebp+var_84], esi
jz short loc_31308F
```

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

```
push 0Dh
call sub_31411B
```

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

```
call sub_3140F3
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C
```

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

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

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

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

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

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

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

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

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

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

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

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

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]
cbp   eax, [ebp+var_84]
jz     short loc_313066
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   esi
push   [ebp+arg_4]
push   edi
call   sub_314623
test   eax, eax
jz     short loc_31306D
cmp   [ebp+arg_0], esi
jz     short loc_31308F

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

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

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

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

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]
cbp 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 esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
jz short loc_31308F

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

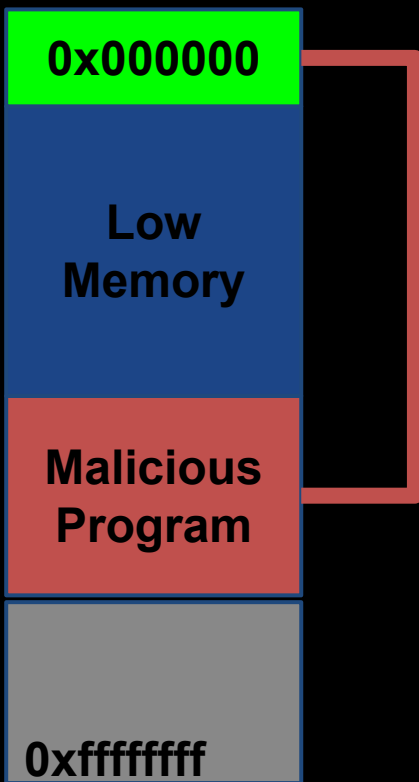
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C

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

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


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

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

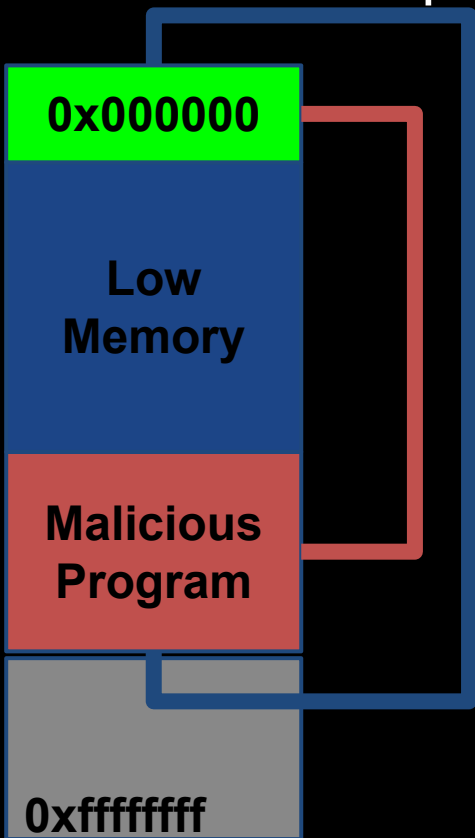
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C

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

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

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]
cbp 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
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+55
push 0Dh
call sub_31411B

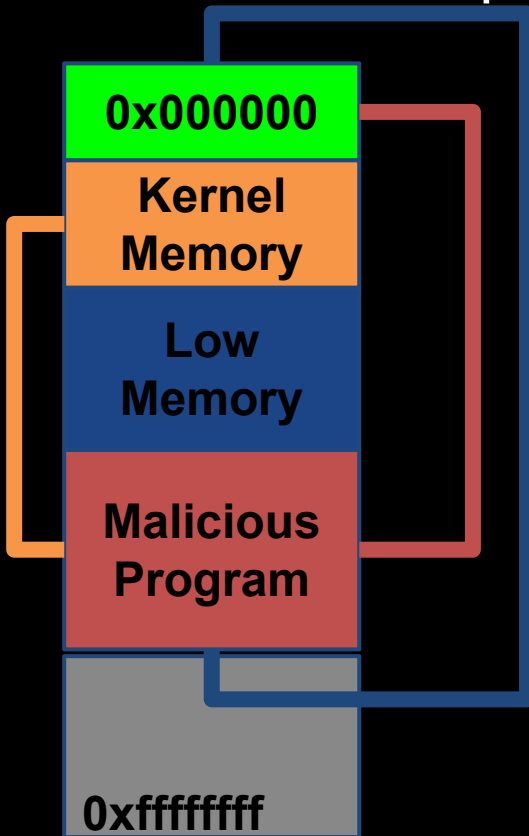
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C

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

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

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
jnz short loc_313066
mov eax, [ebp+var_70]
cbp 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
mov esi, 1D0h
push esi
push [ebp+arg_4]
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
shl eax, 31308F

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

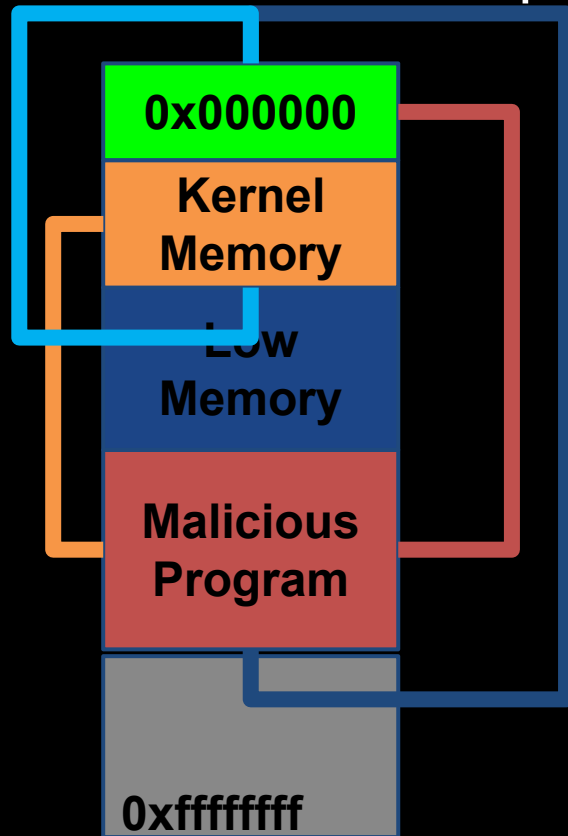
loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C

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

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

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]
cbp 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
mov esi, 1D0h
push esi
push [ebp+arg_4]
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
shl eax, 31308F
```

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

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

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

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

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

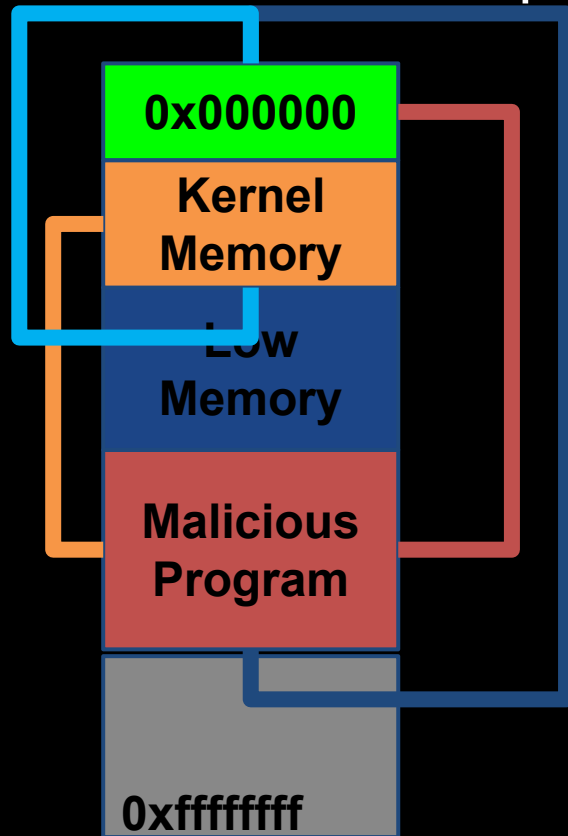
```
loc_31308C: ; CODE XREF: sub_312FD8
```

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

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
jnz short loc_313066
mov eax, [ebp+var_70]
cbp 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
push esi
push [ebp+arg_4]
push edi
call sub_314623
test eax, eax
jz short loc_31306D
cmp [ebp+arg_0], esi
push 0Fh
call sub_31411B
loc_31306D:
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C
; -----
loc_31307D:
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
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!

```
push edi
call sub_314623
test eax, eax
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_313066
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
mov esi, 1D0h
push esi
push esi
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+55
push 0Dh
call sub_31411B

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

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

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

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
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
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+55
; sub_312FD8+55
push 0Dh
call sub_31411B

loc_31306D: ; CODE XREF: sub_312FD8+49
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
jmp short loc_31308C
; -----
loc_31307D: ; CODE XREF: sub_312FD8
call sub_3140F3
and eax, 0FFFFFFh
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
```

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

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

loc_31306D: ; CODE XREF: sub_312FD8
; sub_312FD8+49
call sub_3140F3
test eax, eax
jg short loc_31307D
call sub_3140F3
cmp [ebp+arg_0], esi

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

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

Can still be a useful source of information on older systems

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

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

```
push   0Dh
call   sub_31411B
```

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

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

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

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

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

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

Kernel Space Protections

SMEP / SMAP

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

```
push edi
call sub_314623
test eax, eax
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
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+var_70]
push eax
mov esi, [ebp+var_84]
push esi
push [ebp+var_70]
push esi
call sub_312FD8
test eax, eax
jz short loc_313066
cmp [ebp+var_70], [ebp+var_84]
jz short loc_313066
loc_313066:
push 0
call sub_312FD8
loc_31306D:
call sub_312FD8
test eax, eax
jg short loc_31307D
call sub_312FD8
jmp short loc_31307D
loc_31307D:
call sub_3140F3
and eax, 0FFFFFFh
or eax, 80070000h
loc_31308C:
mov [ebp+var_4], eax
```



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");  
}
```

0x000000

Kernel
Memory

Low
Memory

Malicious
Program

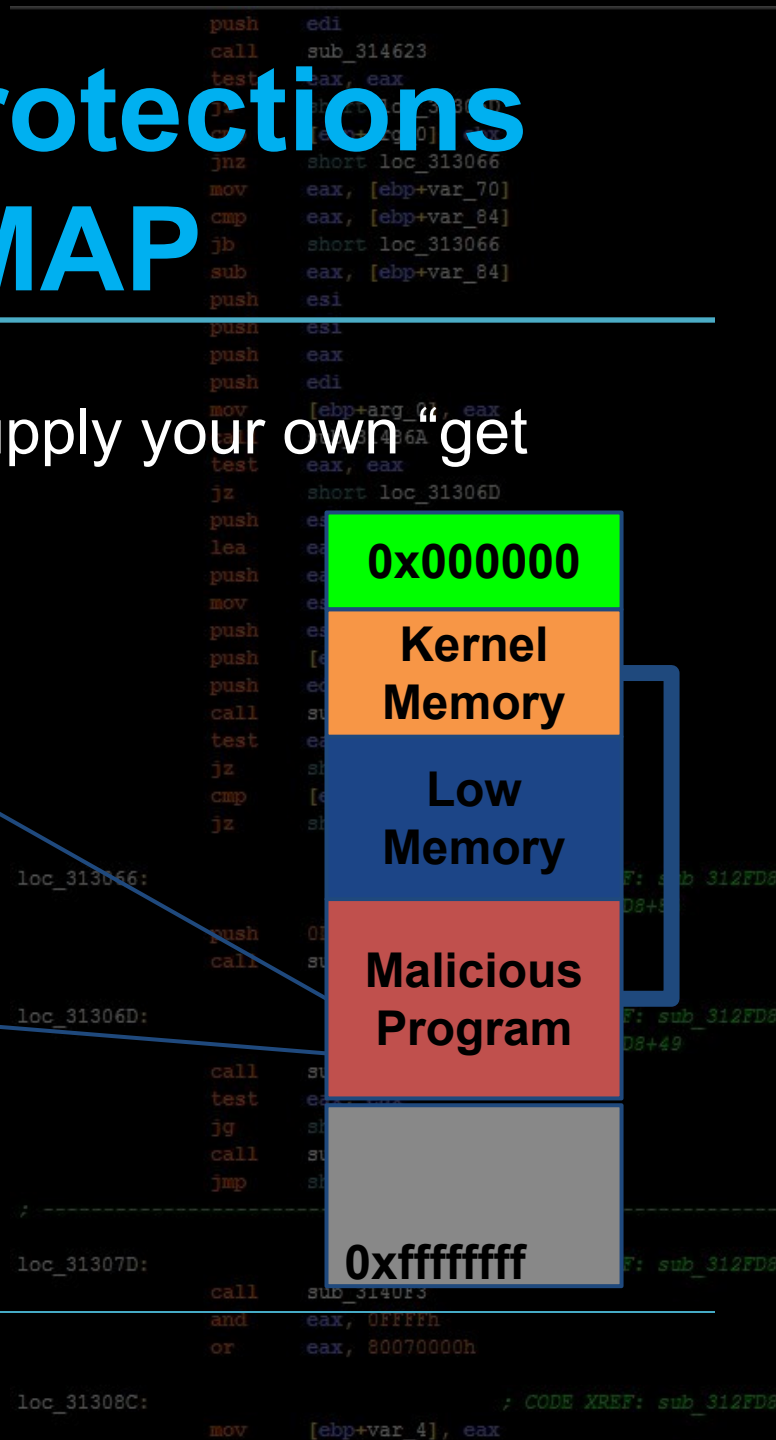
0xffffffff

```
push edi  
call sub_314623  
test eax, eax  
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  
test eax, eax  
jz short loc_31306D  
push edi  
lea eax, [ebp+var_70]  
push eax  
mov esi, [ebp+var_84]  
push esi  
push [ebp+var_70]  
push [ebp+var_84]  
call sub_313066  
test eax, eax  
jz short loc_31306D  
cmp [ebp+var_70], [ebp+var_84]  
jz short loc_31306D  
loc_313066: push 0  
call sub_313066  
loc_31306D: call sub_313066  
test eax, eax  
jg short loc_31307D  
call sub_313066  
jmp short loc_31307D  
loc_31307D: call sub_3140F3  
and eax, 0FFFFFFh  
or eax, 80070000h  
loc_31308C: mov [ebp+var_4], eax ; CODE XREF: sub_312FD8
```

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");
}
```

0x000000

Kernel
Memory

Low
Memory

Malicious
Program

0xffffffff

loc_31308C:

mov [ebp+var_4], eax

or eax, 80070000h

and eax, 0FFFFFFh

call sub_3140F3

loc_31307D:

jmp sh

call su

ig sh

test ea

call su

loc_31306D:

call su

loc_313066:

push 01

call su

push ea

loc_313065:

push ea

mov ea

push ea

lea ea

push ea

push ea

test ea

call su

push ea

push ea

lea ea

push esi

push esi

push edi

mov [ebp+arg_0], eax

test eax, eax

loc_31306A:

loc_313069:

loc_313068:

loc_313067:

loc_313066:

loc_313065:

loc_313064:

loc_313063:

loc_313062:

loc_313061:

loc_313060:

loc_31305F:

loc_31305E:

loc_31305D:

loc_31305C:

loc_31305B:

loc_31305A:

loc_313059:

loc_313058:

loc_313057:

loc_313056:

loc_313055:

loc_313054:

loc_313053:

loc_313052:

loc_313051:

loc_313050:

loc_31304F:

loc_31304E:

loc_31304D:

loc_31304C:

loc_31304B:

loc_31304A:

loc_313049:

loc_313048:

loc_313047:

loc_313046:

loc_313045:

loc_313044:

loc_313043:

loc_313042:

loc_313041:

loc_313040:

loc_31303F:

loc_31303E:

loc_31303D:

loc_31303C:

loc_31303B:

loc_31303A:

loc_313039:

loc_313038:

loc_313037:

loc_313036:

loc_313035:

loc_313034:

loc_313033:

loc_313032:

loc_313031:

loc_313030:

loc_31302F:

loc_31302E:

loc_31302D:

loc_31302C:

loc_31302B:

loc_31302A:

loc_313029:

loc_313028:

loc_313027:

loc_313026:

loc_313025:

loc_313024:

loc_313023:

loc_313022:

loc_313021:

loc_313020:

loc_31301F:

loc_31301E:

loc_31301D:

loc_31301C:

loc_31301B:

loc_31301A:

loc_313019:

loc_313018:

loc_313017:

loc_313016:

loc_313015:

loc_313014:

loc_313013:

loc_313012:

loc_313011:

loc_313010:

loc_31300F:

loc_31300E:

loc_31300D:

loc_31300C:

loc_31300B:

loc_31300A:

loc_313009:

loc_313008:

loc_313007:

loc_313006:

loc_313005:

loc_313004:

loc_313003:

loc_313002:

loc_313001:

loc_313000:

loc_312FFD:

loc_312FFC:

loc_312FFB:

loc_312FFA:

loc_312FF9:

loc_312FF8:

loc_312FF7:

loc_312FF6:

loc_312FF5:

loc_312FF4:

loc_312FF3:

loc_312FF2:

loc_312FF1:

loc_312FF0:

loc_312FEF:

loc_312FE8:

loc_312FE7:

loc_312FE6:

loc_312FE5:

loc_312FE4:

loc_312FE3:

loc_312FE2:

loc_312FE1:

loc_312FE0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

loc_312FD7:

loc_312FD6:

loc_312FD5:

loc_312FD4:

loc_312FD3:

loc_312FD2:

loc_312FD1:

loc_312FD0:

loc_312FDF:

loc_312FDE:

loc_312FDD:

loc_312FDC:

loc_312FDB:

loc_312FDA:

loc_312FD9:

loc_312FD8:

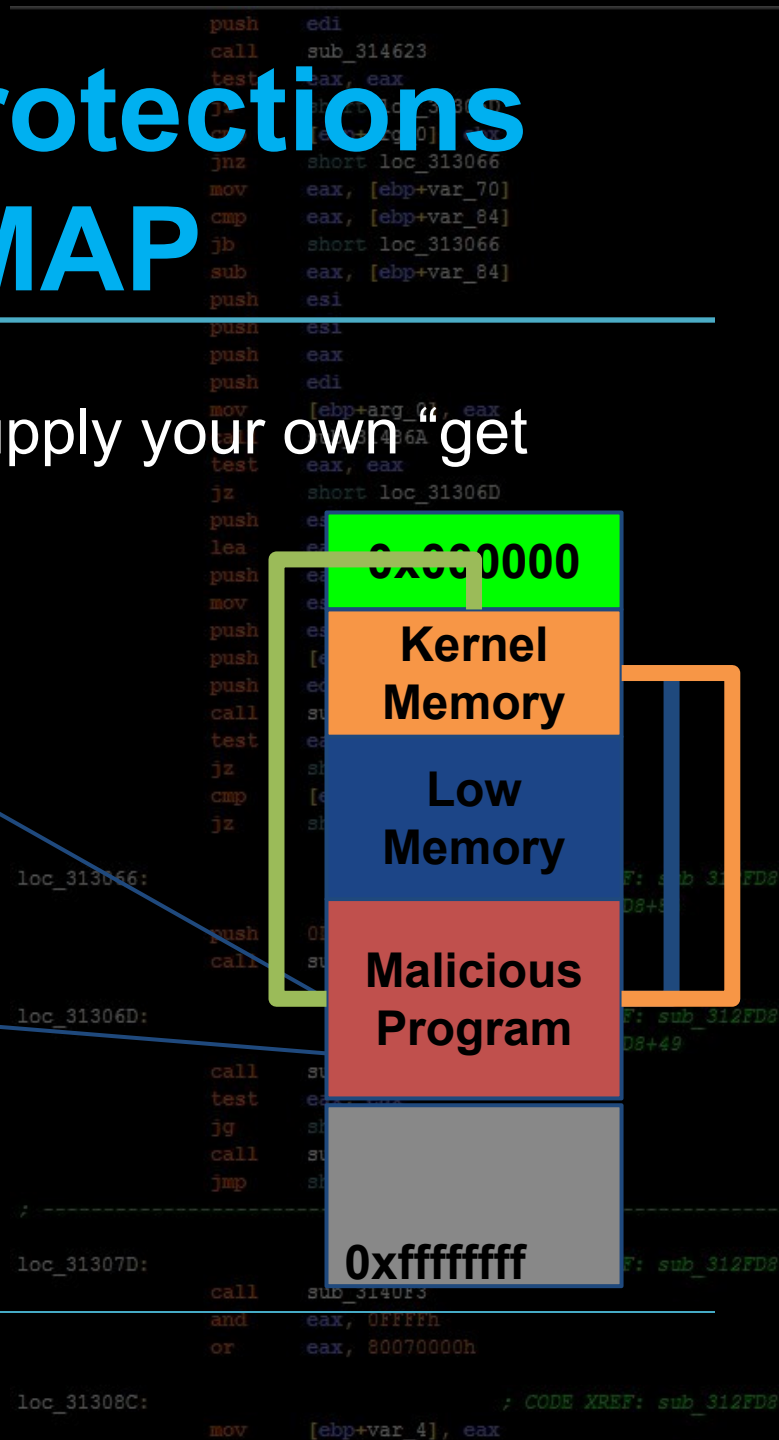
loc_312FD7:

loc_312FD6:

Kernel Space Protections SMEP / SMAP

Common Exploitation Technique: Supply your own "get root" code.

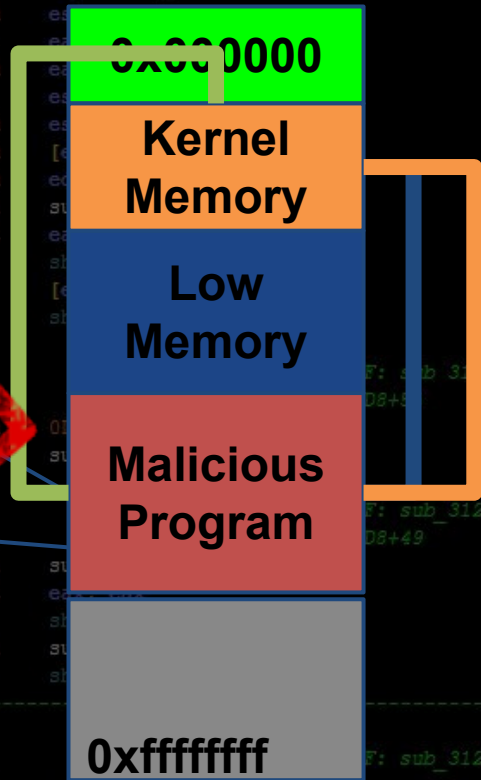
```
void get_r00t() {  
    commit_creds(prepare_kernel_cred(0));  
}  
  
int main(int argc, char * argv) {  
    ...  
    trigger_fp_overwrite(&get_r00t);  
    ...  
    //trigger fp use  
    trigger_vuln_fp();  
    // Kernel Executes get_r00t()  
    ...  
    // Now we have root  
    system("/bin/sh");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



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");  
}
```



```
push edi  
call sub_314623  
test eax, eax  
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  
test eax, eax  
jz short loc_31306D  
push esi  
lea eax, [ebp+var_70]  
push eax  
mov esi, [ebp+var_70]  
push esi  
push [ebp+var_70]  
call sub_314623  
test eax, eax  
jz short loc_31306D  
cmp [ebp+var_70], [ebp+var_84]  
jz short loc_31306D  
push esi  
call sub_314623  
test eax, eax  
jg short loc_31306D  
call sub_314623  
jmp short loc_31306D  
  
loc_31307D: call sub_3140F3  
and eax, 0FFFFFFh  
or eax, 80070000h  
  
loc_31308C: mov [ebp+var_4], eax ; CODE XREF: sub_312FD8
```

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

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

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

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

loc_313066:
push 0Dh
call sub_31411B

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

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

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

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
push    esi
push    eax
push    edi
mov     [ebp+arg_0], eax
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+55
push    0Dh
call    sub_31411B

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

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

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
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+55
push 0Dh
call sub_31411B

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

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

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

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
push esi
push eax
push edi
mov [ebp+arg_0], eax
call sub_31466F
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
push 0Dh
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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

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

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

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

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
```

```
push esi
push eax
push edi
mov [ebp+arg_0], eax
call sub_31406F
test eax, eax
jz short loc_31306D
push esi
lea eax, [ebp+arg_0]
push eax
```

```
push 00h
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+55
```

```
push 0Dh
call sub_31411B
```

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

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

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

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

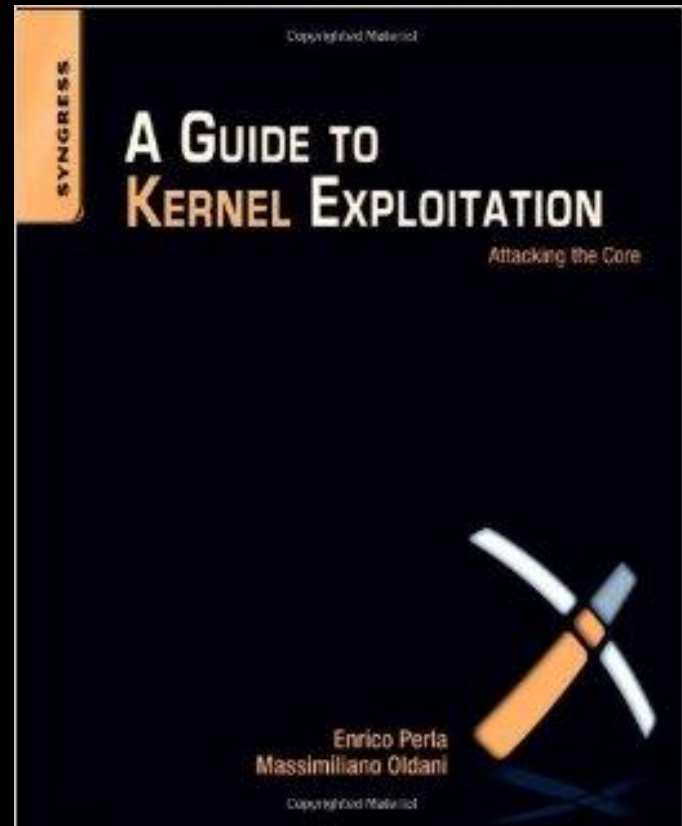
```
loc_31308C: ; CODE XREF: sub_312FD8
```

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

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
```



sub_312FD8
8

sub_312FD8
9

sub_312FD8

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

loc_31308C: ; CODE XREF: sub_312FD8

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