	v [ebp+arg_0], eax
	11 sub_31486A
	short loc_31306D
	a [ebp+arg_0]
Linux Kernel Exploit	ation
	sh fahriang ()
	sh [ebp+arg_4] sh edi
	short loc 31306D
cmp	p [ebp+arg_0], esi
	short loc_31308F
Modern Binary Exploitatio	; CODE XREF: sub 312FI
Modern Dinary Exploitatio	; sub_312FD8+59
i pus	
$\frac{1}{1000} \frac{1}{1000} \frac{1}{1000$	11 sub_31411B
CSCI 4968 - Spring 2015	
Datrial Diarnat	
Patrick Biernat	
cal	
	p short loc_31308C
loc_31307D:	
cal	
and	
loc 31308C:	

First order of Busines	jb sub push	edi sub_314623 eax, eax short loc_31306D [ebp+arg_0], ebx short loc_313066 eax, [ebp+var_70 eax, [ebp+var_84 short loc_313066 eax, [ebp+var_84 esi	
	push push push mov	esi eax edi [ebp+arg_0], eax	
		sub_31486A eax, eax short loc_31306D	
		esi eax, [ebp+arg_0] eax	
		esi, 1D0h esi [ebp+arg_4]	
		edi sub_314623 eax, eax short loc_31306D	
		<pre>[ebp+arg_0], esi short loc_31308F</pre>	
loc_31306			
loc_31306	call		
		sub_3140F3 eax, eax	
		short loc_31307D sub_3140F3 short loc_31308C	
; loc_31307		sub_3140F3	
	and or	eax, 0FFFFh eax, 80070000h	
loc_31308	BC: mov	[ebp+var_4], eax	

# First order of Business

ALSINALDER CALLED

# RETERPORARY

TEMPORARY

memegenerator.net

# A constraints of the second se

# DELETES /TMP

## **Lecture Overview**

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

		sub_314623
		short loc_31306D
		[ebp+arg_0], ebx
		short loc_313066
		eax, [ebp+var_70]
		eax, [ebp+var_84]
		short loc_313066
		eax, [ebp+var_84]
	pusn	esi
		[ebp+arg_0], eax
		sub_31486A
		short loc_31306D
		eax, [ebp+arg_0]
		[ebp+arg_4]
		sub_314623
		short loc_31306D
		[ebp+arg_0], esi
		short loc_31308F
c_313066:		
		sub_31411B
c_31306D:		
		sub_3140F3
		short loc_31307D
		sub_3140F3
		short loc_31308C
c_31307D:		
	call	sub_3140F3
	and	eax, OFFFFh
c_31308C:		
		[ebp+var 4], eax

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

		sub_314623		
		short loc_31306		
_		[ebp+arg_0], eb		
	jnz	short loc_31306		
<b>ri</b>	100	eax, [ebp+var 7		
		eax, [ebp+var_8	4]	
		short loc 31306		
		eax, [ebp+var_8		
	pusn	esi		
		[ebp+arg_0], ea		
ning	Tin	sub_31486A		
nin	5 111	eax, eax		
	12	short loc 31306		
		eax, [ebp+arg_0		
		eax		
		[ebp+arg_4] edi		
		sub_314623		
		short loc_31306		
		[ebp+arg_0], es		
		short loc_31308		
066:				
		ODh		
		sub_31411B		
06D:				
		sub_3140F3		
		eax, eax		
		short loc_31307		
		sub_3140F3		
		short loc_31308	С	
	call	sub_3140F3		
		eax, OFFFFh		
08C:				
United				

loc\_31308C:

ebp+var 4], eax

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

#### Userspace is an *abstraction* that runs "on top" of the kerne

		sub_314623	
		short loc_31306D	
		[ebp+arg_0], ebx	
tri	ามซ	short loc_313066	
		eax, [ebp+var 70]	
		eax, [ebp+var_84]	
	The		
		short loc_313066	
		eax, [ebp+var_84]	
	pusn	esi	
	pusn	CSI	
•	MOV	[ebp+arg_0], eax	
nning		sub_31486A	
3111111	test		
		short loc_31306D	
		eax, [ebp+arg 0]	
		eax	
,,	pusn	esi	
on" c	of th	e kernel	
op c	Fusi		•
		sub_314623	
		short loc_31306D	
		<pre>[ebp+arg_0], esi</pre>	
		short loc_31308F	
313066:			
	call	sub_31411B	
31306D:			
		sub 3140F3	
		eax, eax	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
31308C:			
		[ebp+var_4], eax	

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

Userspace is an abstraction that runs "on top" of the kern

- 1. Filesystem I/O
- 2. Privilege Levels (Per User/Per Group)<sup>100\_313</sup>
- 3. Syscalls
- 4. Processes
- 5. And so much more

		sub_314623	
		short loc_31306D	
_		[ebp+arg_0], ebx	
ri	jnz	short loc_313066	
	477	eax, [ebp+var_70	
		eax, [ebp+var_84	
		short loc_313066	
		eax, [ebp+var_84	
	pusn	631	
	mov	[ebp+arg_0], eax	
ning	Jin	sub_31486A	
31111	Itest		
		short loc_31306D	
		<pre>eax, [ebp+arg_0]</pre>	
	push		_
ר" ר	h fast h	e kerne	
	Pusk		•
		sub_314623	
		short loc_31306D	
		<pre>[ebp+arg_0], esi</pre>	
		short loc_31308F	
66:			
		sub_31411B	
6D:			
		sub_3140F3	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
	call	sub_3140F3	
	and	eax, OFFFFh	
BC:			

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

Userspace is an abstraction that runs "on top" of the kern

- 1. Filesystem I/O
- 2. Privilege Levels (Per User/Per Group)<sup>10C\_313066</sup>
- 3. Syscalls
- 4. Processes
- 5. And so much more

<pre>call sub_314623 test ear, ear jz short loc_31306D cmp [ebp+arg_0], ebx short loc_313066 ear, [ebp+var_70] ear, [ebp+var_84] jb short loc_313066 sub ear, [ebp+var_84] push ear push eai push eai push ear, ear jz short loc_31306D push esi lea ear, [ebp+arg_0] push eas mov esi, 1D0h push eas iea ear, [ebp+arg_0] push eas mov esi, 1D0h push esi jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F ; sub_312FD8+55 push ODh call sub_3140F3 test ear, ear jg short loc_31307D call sub_3140F3 test ear, ear jg short loc_31308C ; CODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test ear, ear jg short loc_31307D call sub_3140F3 jmp short loc_31308C</pre>	push	edi	
<pre>jz short loc_31306D cmp [ebp+arg_0], ebx short loc_313066 ear, [ebp+var_0] jb short loc_313066 sub ear, [ebp+var_84] push esi push esi push edi mov [ebp+arg_0], eax sub_31486A ear, eax jz short loc_31306D push esi lea ear, [ebp+arg_0] push esi iea ear, [ebp+arg_0] push esi iea ear, [ebp+arg_0] push esi iz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F ; cODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test ear, eax jg short loc_31308C ; cODE XREF: sub_312FD8</pre>		sub_314623	
<pre>cmp [ebp+arg_0], ebx short loc_313066 eax, [ebp+var_70] eax, [ebp+var_84] jb short loc_313066 sub eax, [ebp+var_84] push esi push esi push esi push esi lea eax, [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push esi lea eax, [ebp+arg_0] push esi cofuther[Keernel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F ; sub_312FD8+55 push ODh call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 test eax, eax jg short loc_31308C</pre>			
<pre>short loc_313066 eax, [ebp+var_70] eax, [ebp+var_84] jb short loc_313066 sub eax, [ebp+var_84] push esi push esi push eax push edi nov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F : cODE XREF: sub_312FD8 feal sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 test eax, eax jg short loc_31308C ; cODE XREF: sub_312FD8</pre>			
<pre>eax, [ebp+var_70] eax, [ebp+var_84] jb short loc_313066 sub eax, [ebp+var_84] push esi push esi push edi nov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push esi lea eax, [ebp+arg_0] push esi of the by Kethel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F : cODE XREF: sub_312FD8 ; sub_312FD8+55 push ODh call sub_31411B : cODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31308C ; cODE XREF: sub_312FD8</pre>	cmb	<pre>[ebp+arg_0], ebx</pre>	
<pre>cax, [ebp+var_94] jb short loc_313066 sub eax, [ebp+var_84] push esi push esi push eai nev [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax nov esi, 1DDh push esi oficetheiceth</pre>	- 377		
<pre>jb short loc_313066 sub eax, [ebp+var_84] push esi push eax push edi mov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi <b>OfusChebsKethel.</b> call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>sub eax, [ebp+var_84] push esi push esi push eax push edi mov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push esi cfisther[it] call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>push esi push esi push eax push edi mcv [ebp+arg_0], eax sub_31486A eax, sub_31486A eax, sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push esi of the prover of the prove of the</pre>		short loc_313066	
<pre>push esi push eax push edi mov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi <b>of_stheinkeerinel.</b> call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>		<pre>eax, [ebp+var_84]</pre>	
<pre>push eax push edi mov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi fistheinfeetinet. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>push edi mov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Ofisthebikaerinel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	pusn	631	
<pre>mov [ebp+arg_0], eax sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Ofisthebitaetineet. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>Stert sub_31486A eax, eax jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Ofisthebikaernel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Ofuthebkeetheel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	mov		
<pre>jz short loc_31306D push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Ofuthebkeetheel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	ngin	sub_31486A	
<pre>push esi lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Ofusthebicernel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	Otest		
<pre>lea eax, [ebp+arg_0] push eax mov esi, 1D0h push esi Of the b keernel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>		short loc_31306D	
<pre>push eax mov esi, 1D0h push esi Ofuthebkeethel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>mov esi, 1D0h push esi Of_stheb/cernel. call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>		<pre>eax, [ebp+arg_0]</pre>	
<pre>call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	pusn	esi	
<pre>call sub_314623 test eax, eax jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	otth	e kernel	
<pre>jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>	Pusk		•
<pre>jz short loc_31306D cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>cmp [ebp+arg_0], esi jz short loc_31308F</pre>			
<pre>jz short loc_31308F</pre>			
<pre>; CODE XREF: sub 312FD8 ; sub_312FD8+55 push ODh call sub_31411B ; CODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8</pre>			
; sub_312FD8+59 push ODh call sub_31411B ; CODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8		short loc_31308F	
; sub_312FD8+59 push ODh call sub_31411B ; CODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8			
<pre>push ODh call sub_31411B             ; CODE XREF: sub_312FD8             ; sub_312FD8+49     call sub_3140F3     test eax, eax     jg short loc_31307D     call sub_3140F3     jmp short loc_31308C     ; CODE XREF: sub_312FD8</pre>			
<pre>call sub_31411B</pre>			
; CODE XREF: sub_312FD8 ; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8			
; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8			
; sub_312FD8+49 call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8			
<pre>call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8</pre>			
<pre>test eax, eax jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8</pre>	call		
jg short loc_31307D call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8			
call sub_3140F3 jmp short loc_31308C ; CODE XREF: sub_312FD8		short loc 31307D	
jmp short loc_31308C ; CODE XREF: sub_312FD8			
; CODE XREF: sub_312FD8			
call sub 3140F3			
	call	sub_3140F3	

			n		
5	G-			0	

; CODE XREF: sub\_ bp+var 41. eax

## What's a Kernel?

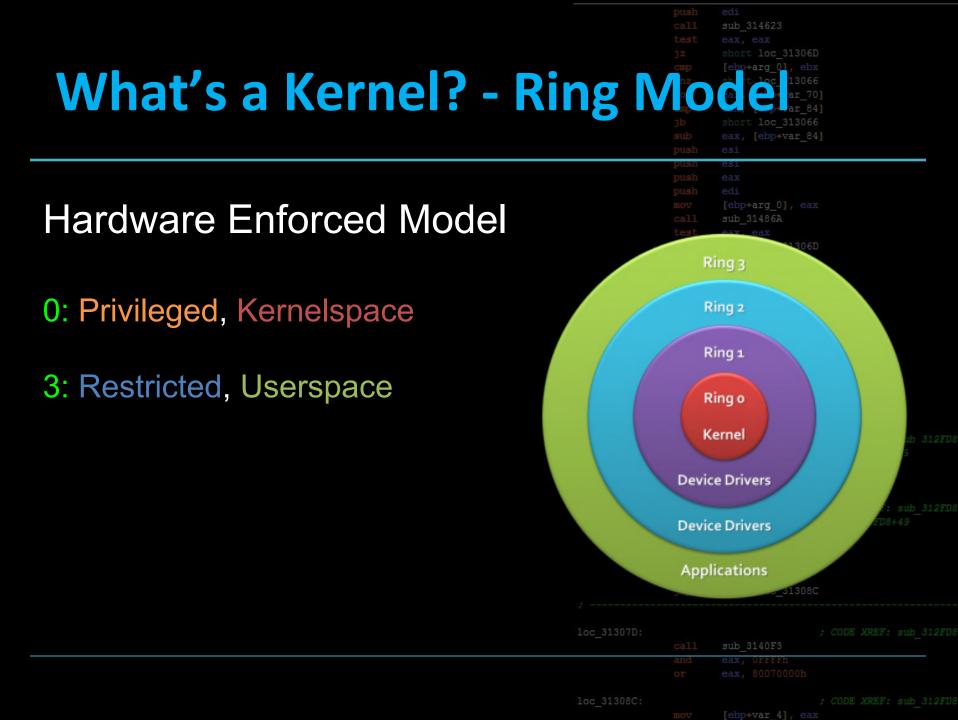
What's a Kernel?		edi sub_314623 eax, eax short loc_31306D [ebp+arg_0], ebx short loc_313066 eax, [ebp+var_70] eax, [ebp+var_84] short loc_313066 eax, [ebp+var_84] esi	
Low Level code with two major res	push push ponsibit jz push	esi eax edi [ep+arg_0], eax eax, eax short loc_31306D esi	
<ol> <li>Interact with and control hardware com</li> <li>Provide an Environment in which Applic</li> </ol>	push	<pre>eax, [ebp+arg_0] eax esi, 1D0h esi [ebp+arg_4] full_site(23) eax, eax short loc_31306D [ebp+arg_0], esi short loc_31308F</pre>	
	Loc_313066: push call Loc_31306D: call test	0Dh sub_31411B ;; sub_3140F3 eax, eax	
	jg call jmp ; Loc_31307D: call	<pre>short loc_31307D sub_3140F3 short loc_31308C sub_3140F3 </pre>	

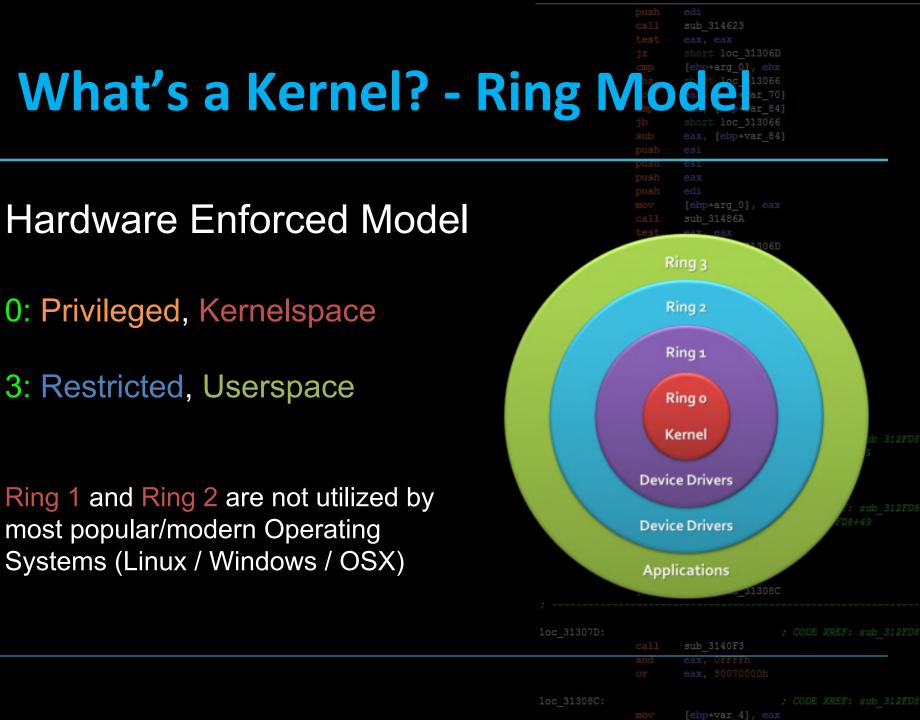
- Interact with and control hardware compo 1.
- Provide an Environment in which Applicat 2.

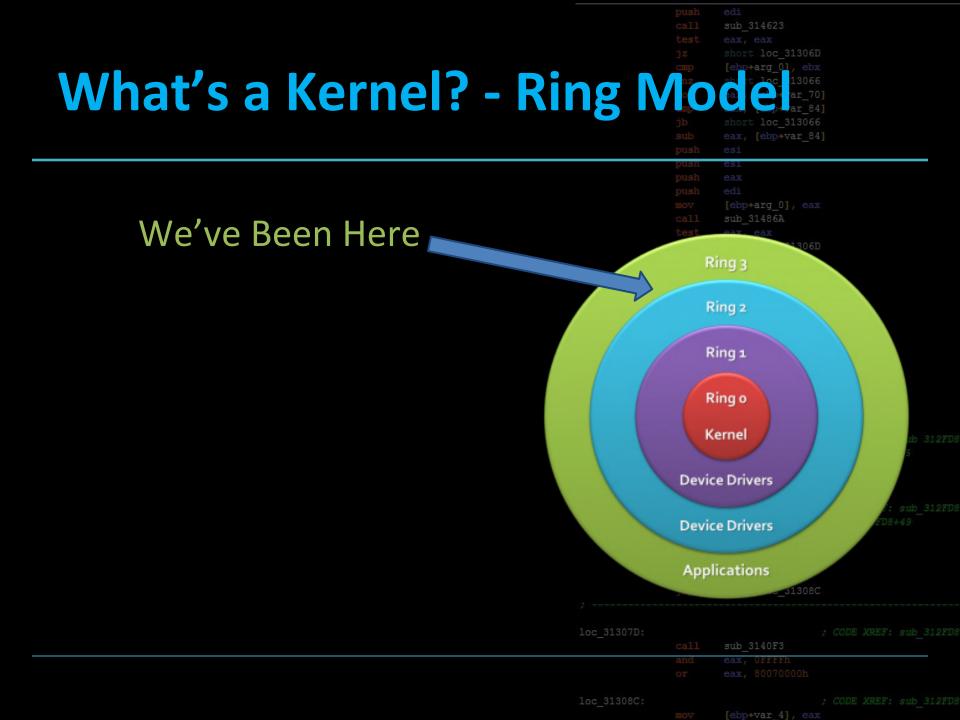
	[ebp+arg_0], esi
	short loc_31308F
	sub_31411B
	sub_3140F3
	short loc_31307D
	sub_3140F3
	short loc_31308C
	sub_3140F3
	eax, OFFFFh

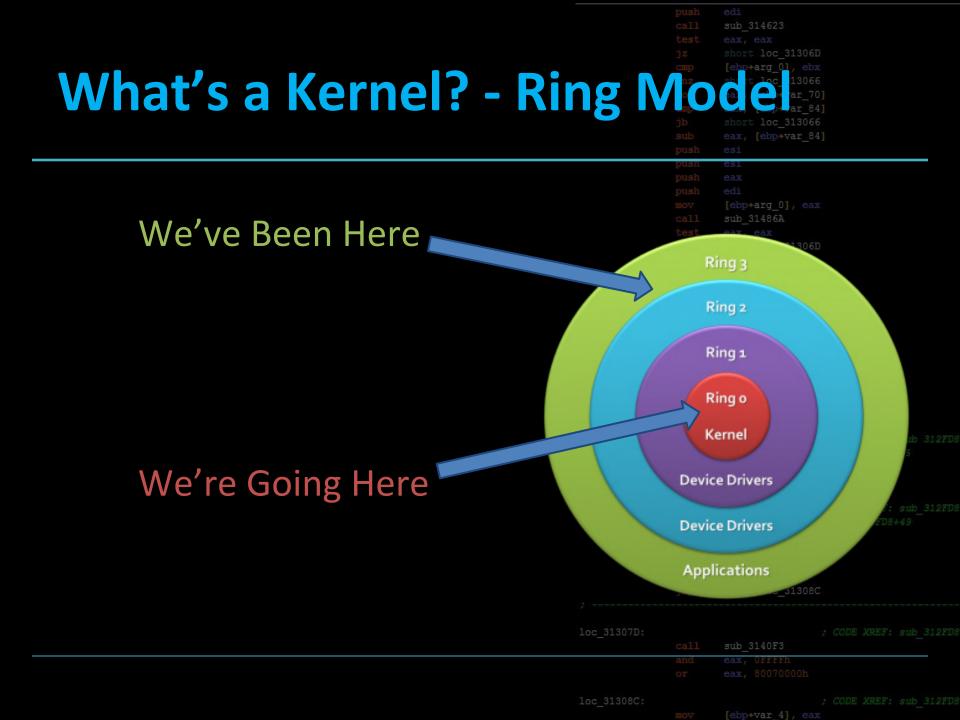
bc_31308C:			
	[ebp+var	4],	

# What's a Kernel? Low Level code with two major responsibilities Interact with and control hardware components 1. Provide an Environment in which Applications can run 2. The Kernel is the core of the operating system sub 3140F3 sub 3140F3





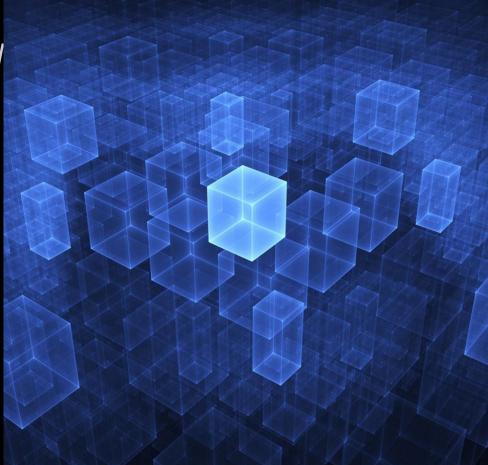




## **Obligatory Matrix Analogy**

edi sub\_314623 eax, eax short loc\_31306D [ebp+arg\_0], ebx short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] esi

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



d eax, 0FFFFh eax, 80070000

loc\_31308C:

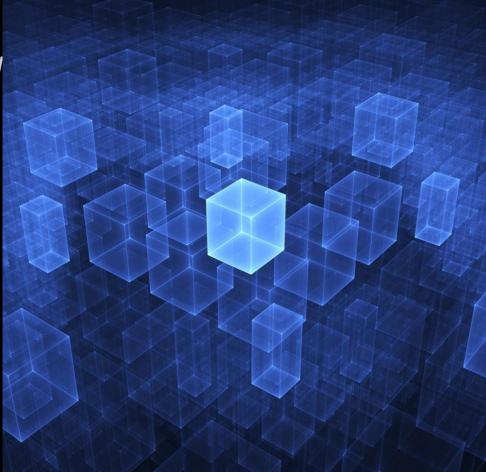
; CODE XREF: sub\_3: hpp+var 4], eax

## **Obligatory Matrix Analogy**

edi sub\_314623 eax, eax short loc\_31306D [ebp+arg\_0], ebx short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] 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



nd eax, OFFFFh r eax, 8007000

loc\_31308C:

; CODE XREF: sub\_31 pp+var 4], eax

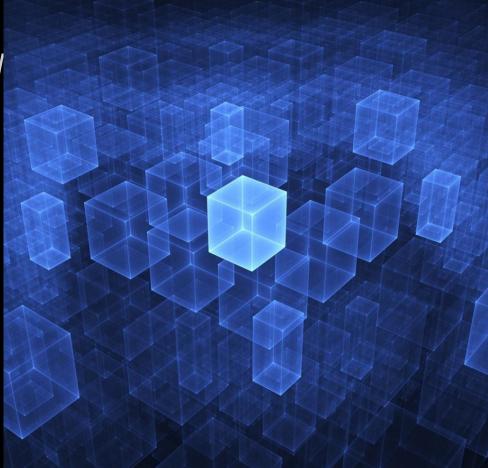
## **Obligatory Matrix Analogy**

edi sub\_314623 eax, eax short loc\_31306D [ebp+arg\_0], ebx short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] 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



d eax, 0FFFFh eax, 8007000





# Kernel Pwning in Popular

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

Remember JailbreakMe?

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



## **Kernel Basics**

## Your Kernel is:

## Managing your Processes Managing your Memory Coordinating your Hardware

		sub_314623
		short loc_31306D
		[ebp+arg_0], ebx
		short loc_313066
		eax, [ebp+var_70]
		eax, [ebp+var_84]
		short loc_313066
		eax, [ebp+var_84]
	pusn	1631
		[ebp+arg_0], eax
		sub_31486A
		short loc_31306D
		eax, [ebp+arg_0]
		[ebp+arg_4]
		edi
		sub_314623
		eax, eax
		short loc_31306D
		[ebp+arg_0], esi
		short loc_31308F
100 2120551		
loc_313066:		
		sub_31411B
		200_214110
loc_31306D:		
		sub 3140F3
		short loc_31307D
		sub_3140F3
		short loc 31308C
loc_31307D:		
		sub_3140F3
	and	eax, OFFFFh
loc_31308C:		

## **Kernel Basics**

## Your Kernel is:

## Managing your Processes Managing your Memory Coordinating your Hardware

## A crash oftentimes means a reboot!

	sub_314623
	short loc_31306D
	[ebp+arg_0], ebx
	short loc_313066
	eax, [ebp+var_70]
	eax, [ebp+var_84]
	short loc_313066
	eax, [ebp+var_84]
pusn	831
	[ebp+arg_0], eax
	sub_31486A
	short loc_31306D
	<pre>eax, [ebp+arg_0]</pre>
	[ebp+arg_4]
	sub_314623
	short loc_31306D
	[ebp+arg_0], esi
	short loc_31308F
	sub_31411B
	sub_3140F3
	short loc_31307D
	sub_3140F3
	short loc_31308C
	sub_3140F3
and	eax, OFFFFh

loc\_31308C:

; CODE XREF: sub\_31 abp+var 41, eax

## **Kernel Basics**

## Your Kernel is:

Managing your Processes Managing your Memory Coordinating your Hardware

A crash oftentimes means a reboot!

	pusn	esi
		[ebp+arg_0], eax
		sub_31486A
		short loc_31306D
		eax, [ebp+arg_0]
		[ebp+arg_4]
		sub_314623
		short loc_31306D
		[ebp+arg_0], esi
		short loc_31308F
66:		
/		
		sub_31411B
6D:		

; sub\_312FD8+49

call sub

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

loc_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc_31308C:			
		[ebp+var 4], ea	

<b>Basic Exploitation Strate</b>	push call test jz cmp test jz cmp test jz sub sub push	edi sub_314623 eax, eax short loc_313 [ebp+arg_0], short loc_313 eax, [ebp+var eax, [ebp+var short loc_313 eax, [ebp+var esi	ebx 066 _70] _84] 066
The Kernel is typically the most powe	push push push <b>rful</b> jz	esi edi [ehp+arg_0], place short loc_313	we
can find bugs		esi eax, [ebp+arg eax esi, 1D0h esi [ebp+arg_4] edi sub_314623 eax, eax short loc_313 [ebp+arg_0], short loc_313	06D esi
loc_313066:		- ODh sub_31411B	
loc_31306D:			

; 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 140F3 OFFFFh

eax, 80070000h

loc\_31308C:

and or

bp+var 4], eax

<b>Basic Exploitation St</b>	rate	push call test jz cmp Sinz jb sub push	edi sub_314623 eax, eax short loc_31306D [ebp+arg_0], ebx short loc_313066 eax, [ebp+var_70] eax, [ebp+var_84] short loc_313066 eax, [ebp+var_84] esi
The Kernel is typically the most	powe	<u>rful</u>	place we
can find bugs			<pre>short loc_31306D esi eax, [ebp+arg_0] eax esi, 1D0h esi [ebp+arg_4]</pre>
But, how do we go from "vulne execution" without bringing do system?		e re	on ax, e porivileged
<i>System:</i>			sub_31411B
	loc_31306D: ;		; CODE XREF: sub_312FD8 ; sub_312FD8+49 sub_3140F3 eax, eax short loc_31307D sub_3140F3 short loc_31308C
	loc_31307D:		; CODE XREF: sub_312FD8 sub_3140F3
		and or	eax, 0FFFFh eax, 80070000h
	loc_31308C:		; CODE XREF: sub_312FD8 [ebp+var_4], eax

#### Basic Exploitation Strates The Big Picture bus eax, eax acall sub\_314623 test test test acax, eax acax, eax

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

		sub_314623	
	test	eax, eax	
	<b>A</b> T		
		ebre 🛶 🚺 ebr	
	jnz	shor 223066	
		eax, [ebp+var_70	
		eax, [ebp+var_84	
		short loc_313066	
		eax, [ebp+var_84	
	pusn	031	
		[ebp+arg_0], eax	
		sub_31486A	
		short loc 31306D	
	n n		
		eax, [ebp+arg_0]	
		eax	
	NOV		
/e	nush		
	push	[ebp+arg 4]	
		edi	
$\mathbf{d}$		sub_314623	
u			
		eax, eax	
		short loc_31306D	
		[ebp+arg_0], esi	
		short loc_31308F	
		ODh	
		sub_31411B	
		sub_3140F3	
		eax, eax	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
	call	sub_3140F3	
		eax, OFFFFh	

#### Basic Exploitation Strategy (all sub\_314623 eax, eax bor (all sub\_314623 eax, eax nov The Vulnerabilities push est

## You already know how to find these

short loc_31306D
eax, [ebp+arg_0]
[ebp+arg_4]
sub_314623
short loc_31306D
<pre>[ebp+arg_0], esi</pre>
short loc_31308F

		eax, eax short loc 3130	
		sub_3140F3 short loc_3130	
loc_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	

308C: Tebrevar 41

ebp+var 4], eax

#### Basic Exploitation Strates av. eax. The Vulnerabilities push avb\_314623 eax. eax. avb\_3166 eax. [ebp+var\_70] eax. [ebp+var\_84] short loc\_313066 eax. [ebp+var\_84] est

## You already know how to find these!

# Kernel vulnerabilities are almost *exactly* the same as userland vulnerabilities.

j	push	esi	
		[ebp+arg_4]	
		sub_314623	
		short loc_31306D	
		[ebp+arg 0], esi	
		short loc_31308F	
loc 313066:			
		sub_31411B	
loc 31306D:			
		sub_3140F3	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
loc 31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc 31308C:			
		[ebp+var 4], eax	

oc 31306D

#### Basic Exploitation Stratest eax, eax The Vulnerabilitiess push push Eall Sub\_314623 eax, eax Sub\_314623 eax, [ebp+var\_70] eax, [ebp+var\_84] est Sub\_314623 eax, [ebp+var\_84] est

## 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

By now, finding these should be a fan

4. Use-After-Free

<i>tiy</i> the	e sa	ame as		
		[ebp+arg_4]		
		sub_314623		
		short loc_313	6D	
		[ebp+arg_0],		
		short loc_313	8F	
oc 313066:				
		sub_31411B		
oc 31306D:				
		sub 3140F3		
		short loc 313		
		sub 3140F3		
		short loc 313		
niliar	proc	Cess		
oc_31307D:				
		sub_3140F3		
	and	eax, OFFFFh		

Loc 31306D

	3		

; CODE XREF: SUD\_ ebp+var 41. eax

#### Basic Exploitation Stratest eax, eax mov eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84]

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

	short loc_31306D	
	<pre>eax, [ebp+arg_0]</pre>	
	[ebp+arg_4]	
	sub_314623	
	short loc_31306D	
	<pre>[ebp+arg_0], esi</pre>	
	short loc_31308F	
	sub_31411B	
	sub_3140F3	
	short loc_31307D	
	sub_3140F3	
	short loc_31308C	
call	sub_3140F3	
and	eax, OFFFFh	

#### Basic Exploitation Stratest eax, eax The Vulnerabilities push eax, eax abor 200 acall sub\_314623 eax, eax abor 200 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] eax

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)

		<pre>eax, [ebp+arg_0</pre>	
	mov		
rn	el <sup>s</sup> S	pace.	
	Lush		
		edi sub 314623	
		eax, eax	
		short loc 31306	
		[ebp+arg_0], es	
		short loc 31308	
66:			
		sub_31411B	
6D:			
		sub 3140F3	
		eax, eax	
		short loc_31307 sub 3140F3	
		short loc 31308	
		SHOLE TOC_31300	
		sub_3140F3	
		eax, OFFFFh	

			n	$\sim$	
U,	5		U	5	

#### Basic Exploitation Strategy (all sub\_314623) The Vulnerabilities push bior loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] est

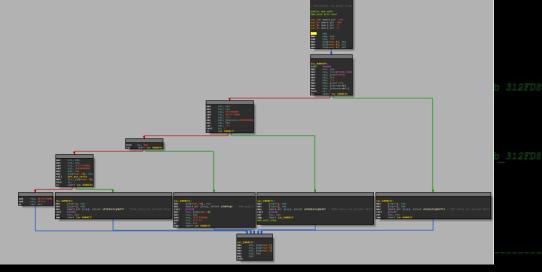
LKMs are just binary blobs like your familiar ELF's, EXE's and MACH-O's. (On Linux, they even use the ELF format)

		e51	
		eax, [ebp+arg_0]	
		[ebp+arg_4]	
		sub_314623	
		short loc_31306D	
		[ebp+arg 0], esi	
		short loc_31308F	
Loc 313066:			
		sub 31411B	
		bus_oraris	
Loc 31306D:			
		sub 3140F3	
		short loc 31307D	
		sub_3140F3	
Loc 31307D:			
	call	sub 3140F3	
		eax, OFFFFh	
Loc 31308C:			
		[ebp+var 4], eax	



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:			
		sub_3140F3	
	and	eax, OFFFFh	
loc_31308C:			

#### Basic Exploitation Stratest inv The Vulnerabilities usbort loc 313066 eax, [ebp+var\_70] eax, [ebp+var\_70] eax, [ebp+var\_84] short loc 313066 eax, [ebp+var\_84] eax

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

Linux.

loc\_31308C:

; CODE XREF: sub\_31. [ebp+var 4], eax

#### Basic Exploitation Strategy (all sub\_314623 eax, eax The Vulnerabilities push ctl sub\_314623 eax, eax (ebp+var\_70) eax, [ebp+var\_84] est call sub\_314623 eax, eax (ebp+var\_70) eax, [ebp+var\_84] est call sub\_314623 eax, eax (ebp+var\_84] est call sub\_314623 eax, eax (ebp+var\_84] est call sub\_314623 eax, eax (ebp+var\_84] est call sub\_314623 eax, eax (ebp+var\_84]

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

insmod rmmod Ismod

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

test eax, eax jz short loc\_31306D cmp [ebp+arg\_0], esi jz short loc\_31308F

loc_313066:			
		sub_31411B	
loc_31306D:			
		sub_3140F3	
		short loc_31307D	
		sub_3140F3	
		short loc_313080	
loc_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc_31308C:			
		[ebp+var_4], eax	

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

insmod--->Insert a module into the running kernelrmmod--->Remove a module from the running kernelIsmod--->List currently loaded modules

loc\_313066:

### A general familiarity with these is helpful

s helpful		0Dh sub_31411B	
loc_31306D:			
		sub_3140F3	
		short loc_31307	D
		sub_3140F3	
		short loc_31308	C
loc 31307D:			
		sub_3140F3	
	and	eax, OFFFFh	

loc\_31308C:

; CODE XREF: SUD\_3. ebp+var 4], eax



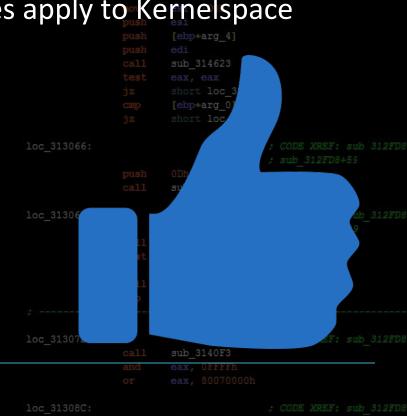
loc\_31308C:

; CODE XREF: sub\_312F [ebp+var 4], eax

### Basic Exploitation Strates Gaining Code Execution Basic Exploitation Strates (air, ear (abp+var\_84) (air) (air) (abp+var\_84) (abp+var\_84)

### You already know how to do this too!

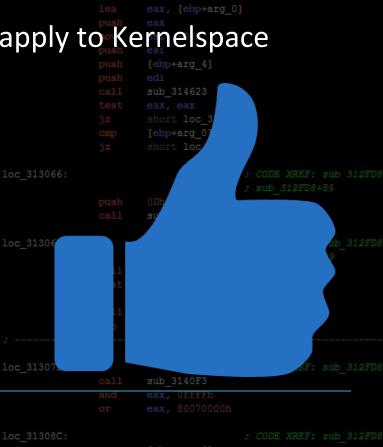
### The same basic exploitation techniques apply to Kernelspace (After all, it's just x86 code!)



### 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.



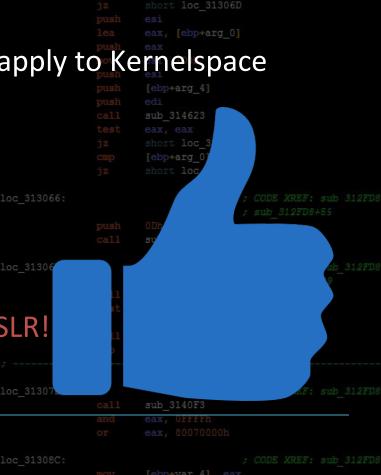
Loc 31306D

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!



### Basic Exploitation Strates (at at at a sub\_314623 (ax, eat at a sub\_314623 (at a sub\_3146

Common Library calls are sometimes differe	ent, so	there is	
		eax, eax	
a slight learning curve involved.		short loc 31306	D
		eax, [ebp+arg_0	
		[ebp+arg_4]	
		sub_314623	
		short loc_31306	D
		[ebp+arg_0], es	
		short loc_31308	
loc_3130	)66:		
		sub_31411B	
loc_3130	)6D:		
		sub_3140F3	
		short loc_31307	D
		sub_3140F3	
		short loc_31308	C.
loc_3130			
	call	sub_3140F3	
	and	eax, OFFFFh	
loc_3130	)8C:		

# Basic Exploitation Strategy and an active and active activ

Common Libra a slight learnin	•	ometimes <i>different</i> ved.	push SCall test jz push lea push mov push	eax, eax short loc_31300 esi eax, [ebp+arg_( eax esi, 1D0h esi [ebp+arg_4]	6D
printf()	>	printk()		edi sub_314623 eax, eax	
memcpy()	>	copy_from_u	user	()/copy	to_user()
malloc()	>	kmalloc()13066:			
				0Dh sub_31411B	
		loc_31306D:		<pre>sub_3140F3 eax, eax short loc_3130' sub_3140F3 short loc_3130'</pre>	
		, loc_31307D:	call and	sub_3140F3 eax, 0ffffh	; CODE XREF: sub_312FD8
		loc_31308C:			



Common Libra a slight learnin	•		So there is sub_old back is 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]
printf()	>	printk()	push edi call sub_314623 test eax, eax
memcpy()	>	copy_from_us	er()/copy_to_user()
malloc()	>	kmalloc()	- ; CODE XREF: sub 312FD8 ; sub 312FD8+55
			push ODh call sub_31411B
Typically, wha man page awa	-		google-search or sizes call sub_3140F3 test eax, eax jg short loc_31307D call sub_3140F3 jup short loc_31308C
		loc_31307D:	; CODE XREF: sub_312FD8 call sub_3140F3 and eax, OFFFFh

loc 31308C:		
	Lobration 11	

# Basic Exploitation Strategy (and the strategy) (and

### **Debugging** kernel code can be difficult

	[ebp+arg_0], eax	
	sub_31486A	
	short loc_31306D	
	eax, [ebp+arg_0]	
	[ebp+arg_4]	
	sub_314623	
	short loc_31306D	
	[ebp+arg_0], esi	
	short loc_31308F	
	sub_31411B	
	sub_3140F3	
	short loc_31307D	
	sub_3140F3	
	short loc_31308C	
call	sub_3140F3	
and	eax, OFFFFh	

	21	13		$\sim$	
5	2				

[ebp+var 4], eax

### **Debugging** kernel code can be difficult

We can't just run the kernel in gdb

	pusn	es1
		[ebp+arg_0], eax
		sub_31486A
		short loc_31306D
		eax, [ebp+arg_0]
		[ebp+arg_4]
		sub_314623
		short loc_31306D
		[ebp+arg_0], esi
		short loc_31308F
)66:		
		sub_31411B
)6D:		
		sub_3140F3
		short loc_31307D
		sub_3140F3
		short loc_31308C
)7D:		
		sub_3140F3
		eax, 0FFFFh eax, 80070000h
		dax, doorddon
)8C:		
		[ebp+var 4], eax
		CDD VAL 4 . CAA

## Basic Exploitation Strategy eax, eax Gaining Code Executions of the provided to the provided t

**Debugging** kernel code can be difficult

We can't just run the kernel in gdb

	[ebp+arg_0], eax
	sub_31486A
	short loc_31306D
	<pre>eax, [ebp+arg_0]</pre>
	[ebp+arg_4]
nr m	αστασ

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.

-1.	100_313066;			
			sub_31411B	
	loc_31306D:			
			sub_3140F3	
			eax, eax	
			short loc_313071	
			sub 3140F3	
			short loc_313080	
	loc_31307D:			
			sub_3140F3	
		and	eax, OFFFFh	
	loc_31308C:			
			[ebp+var_4], eau	

### Basic Exploitation Strategy (all sub\_314623 eax, eax Gaining Code Execution (ext (all sub\_314623 eax, eax (abp+var\_70] (ax, eax (abp+var\_70] (abp+var\_84]

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

TASK: ffff81121ff987b0 CPU: 2 COMMAND: "swapper" PID: 0 #0 [ffff81011fff3b80] crash kexec at ffffffff800b1287 #1 [ffff81011fff3c40] die at ffffffff80065137 #2 [ffff81011fff3c80] do page fault at fffffff8006741e #3 [ffff81011fff3d70] error exit at fffffff8005ddf9 [exception RIP: uhci scan schedule+162] RIP: ffffffff880218ee RSP: ffff81011fff3e20 RFLAGS: 00010007 RAX: 0000002019105000 RBX: 0000002019105000 RCX: ffff81121ff8cb68 RDI: ffff81091fe27950 RDX: 00000000000000 RSI: 0000000000000000 RBP: ffff81011fff3ed0 R8: 000000000000000000 R9: ffff81012b4f7df8 R10: 000000000000000 R11: 0000000af482de4 R12: ffff81091fe27950 R13: 000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367 #4 [ffff81011fff3e98] uhci hub status data at ffffffff880232da [uhci hcd] #5 [ffff81011fff3ec8] usb hcd poll rh status at fffffff80200275 #6 [ffff81011fff3f08] run timer softirg at ffffffff8009a819 #7 [ffff81011fff3f58] do softirg at ffffffff800125a9 #8 [ffff81011fff3f88] call softirg at ffffffff8005e30c #9 [ffff81011fff3fa0] do softirg at ffffffff8006d630 #10 [ffff81011fff3fb0] apic timer interrupt at fffffff8005dc9e --- <IRO stack> ---#11 [ffff81011ffefdf8] apic timer interrupt at ffffffff8005dc9e [exception RIP: acpi safe halt+37] RIP: ffffffff801a62ab RSP: ffff81011ffefea0 RFLAGS: 00000246 RAX: 000000000000000 RBX: ffff81121ff1f8a8 RCX: 00000000000000000 RDX: 00000000000000 RSI: 000000000000000 RDI: 0000000000000000 RBP: ffff81011ffefee8 R8: ffff81011ffee000 R9: 000000000000003f R10: ffff81091fdc4008 R11: 0000000af482de4 R12: ffff81118d6700c0 R13: 000000000402040 R14: 000000000000000 R15: ffff81118d6700c0 ORIG RAX: fffffffffffffff CS: 0010 SS: 0018 #12 [ffff81011ffefea0] acpi processor idle simple at fffffff801a6b29

eax, 80070000

loc\_31308C:

ar Al eav

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

Call Trace Register Dump Stack Dump

PID: 0 TASK: ffff81121ff987b0 CPU: 2 COMMAND: "swapper"
<pre>#0 [ffff81011fff3b80] crash_kexec at ffffffff800b1287</pre>
<pre>#1 [ffff81011fff3c40]die at ffffffff80065137</pre>
<pre>#2 [ffff81011fff3c80] do_page_fault at ffffffff8006741e</pre>
<pre>#3 [ffff81011fff3d70] error exit at ffffffff8005ddf9</pre>
[exception RIP: uhci_scan_schedule+162]
RIP: ffffffff880218ee RSP: ffff81011fff3e20 RFLAGS: 00010007
RAX: 0000002019105000 RBX: 0000002019105000 RCX: ffff81121ff8cb68
RDX: 00000000000000 RSI: 0000000000000 RDI: ffff81091fe27950
RBP: ffff81011fff3ed0 R8: 00000000000000 R9: ffff81012b4f7df8
R10: 000000000000001 R11: 0000000af482de4 R12: ffff81091fe27950
R13: 000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG_RAX: ffffffffffffffff CS: 0010 SS: 0018
<pre>#4 [ffff81011fff3e98] uhci_hub_status_data at ffffffff880232da [uhci_hc</pre>
<pre>#5 [ffff81011fff3ec8] usb_hcd_poll_rh_status at fffffff80200275</pre>
<pre>#6 [ffff81011fff3f08] run_timer_softirg at ffffffff8009a819</pre>
<pre>#7 [ffff81011fff3f58]do_softirg at ffffffff800125a9</pre>
<pre>#8 [ffff81011fff3f88] call_softirg at ffffffff8005e30c</pre>
<pre>#9 [ffff81011fff3fa0] do_softirg at ffffffff8006d630</pre>
<pre>#10 [ffff81011fff3fb0] apic_timer_interrupt at fffffff6005dc9e</pre>
<irq stack=""></irq>
<pre>#11 [ffff81011ffefdf8] apic_timer_interrupt at fffffff8005dc9e</pre>
[exception RIP: acpi_safe_halt+37]
RIP: ffffffff801a62ab RSP: ffff81011ffefea0 RFLAGS: 00000246
RAX: 000000000000000 RBX: ffff81121ff1f8a8 RCX: 0000000000000000
RDX: 00000000000000 RSI: 0000000000000 RDI: 000000000000000000000000000000000000
RBP: ffff81011ffefee8 R8: ffff81011ffee000 R9: 00000000000003f
R10: ffff81091fdc4008 R11: 0000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 00000000000000 R15: ffff81118d6700c0
ORIG_RAX: ffffffffffffffff CS: 0010 SS: 0018
<pre>#12 [ffff81011ffefea0] acpi processor idle simple at fffffff801a6b29</pre>
and eax, OFFFh

oc\_31308C:

, CODE AREF: SUD

### Basic Exploitation Strategy (all sub\_314623 eax, eax Gaining Code Execution (eax) (ebp+var\_84)

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.

PID: 0 TASK: ffff81121ff987b0 CPU: 2 COMMAND: "swapper"
#0 [ffff81011fff3b80] crash kexec at ffffffff800b1287
#1 [ffff81011fff3c40] die at fffffff80065137
<pre>#2 [ffff81011fff3c80] do page fault at fffffff8006741e</pre>
#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: 000000000000000 RSI: 0000000000000 RDI: ffff81091fe27950
RBP: ffff81011fff3ed0 R8: 00000000000000 R9: ffff81012b4f7df8
R10: 000000000000000 R11: 0000000af482de4 R12: ffff81091fe27950
R13: 000000000000286 R14: ffff81091fe27800 R15: ffffffff80200367
ORIG RAX: ffffffffffffffffffffffffffffffffffff
#4 [ffff81011fff3e98] uhci_hub_status_data at ffffffff880232da [uhci_hc
#5 [ffff81011fff3ec8] usb hcd poll rh status at fffffff80200275
#6 [ffff81011fff3f08] run timer softirg at ffffffff8009a819
<pre>#7 [ffff81011fff3f58] do softirg at ffffffff800125a9</pre>
#8 [ffff81011fff3f88] call softirg at ffffffff8005e30c
<pre>#9 [ffff81011fff3fa0] do softirg at ffffffff8006d630</pre>
<pre>#10 [ffff81011fff3fb0] apic timer interrupt at fffffff8005dc9e</pre>
<irq stack=""></irq>
<pre>#11 [ffff81011ffefdf8] apic timer interrupt at fffffff8005dc9e</pre>
[exception RIP: acpi safe halt+37]
RIP: ffffffff801a62ab RSP: ffff81011ffefea0 RFLAGS: 00000246
RAX: 000000000000000 RBX: ffff81121ff1f8a8 RCX: 0000000000000000
RDX: 00000000000000 RSI: 0000000000000 RDI: 000000000000000000000000000000000000
RBP: ffff81011ffefee8 R8: ffff81011ffee000 R9: 00000000000003f
R10: ffff81091fdc4008 R11: 0000000af482de4 R12: ffff81118d6700c0
R13: 000000000402040 R14: 00000000000000 R15: ffff81118d6700c0
ORIG RAX: ffffffffffffffffffffffffffffffffffff
<pre>#12 [ffff81011ffefea0] acpi processor idle simple at ffffffff801a6b29</pre>
and eav OFFFTh

eax, 80070000

loc\_31308C:

ar Al eav

### Basic Exploitation Strategy and the st eat eat and the st ea

Remember: The Kernel manages running processes

		loc_31307D:
sub_3140F3		
eax, OFFFFh	and	
		loc_31308C:
[ebp+var 4], eap		

### Basic Exploitation Strategy inv short loc\_31 Elevate Privileges stort loc\_31 eax, [ebp+var\_84] eax

Remember: The Kernel manages running processes

Therefore: The Kernel keeps track of permissions

		eax		
na n		esses		
	call	sub_31486A		
		short loc_31306		
ermi	ssic	INS <sup>[ebp+arg_0]</sup>		
		[ebp+arg_4]		
		sub_314623		
		short loc_31306		
		[ebp+arg_0], esi		
		short loc_31308B		
13066:				
		sub 31411B		
		000_011110		
L306D:				
		sub 3140F3		
		short loc_313071	)	
		sub 3140F3		
		short loc_313080		
L307D:				
	call	sub_3140F3		
		eax, OFFFFh		
L308C:				
		[ebp+var_4], eax		

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

		short loc_31306	
perm	issi	ONS <sup>[ebp+arg_0</sup>	1]
•			
		[ebp+arg_4]	
		sub_314623	
		short loc_31306	
		[ebp+arg_0], es	
		short loc_31308	
Loc_313066:			
		sub_31411B	
Loc 31306D:			
		sub_3140F3	
		short loc_31307	
		sub 3140F3	
		short loc_31308	
Loc_31307D:			
	call	sub_3140F3	
	and	eax, OFFFFh	

			n	~	
5	5			5	

ebp+var 4], eax

### Basic Exploitation Strategy inv short loc\_313066 Elevate Privilegestar, [ebp+var\_84] art [ebp+var\_84] brot loc\_313066 art, [ebp+var\_84] brot loc\_313066 art, [ebp+var\_84] brot loc\_313066 brot loc\_31066 b

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

push esi lea eax, [ebp+arg\_0] push eax mov esi, 1D0h push esi push [ebp+arg\_4] push edi call sub\_314623 test eax, eax jz short loc\_31306D cmp [ebp+arg\_0], esi jz short loc\_31308F ; CODE XREF: sub\_312F ; sub\_312FD8+55 push ODh call sub\_31411B ; CODE XREF: sub\_312F ; sub\_312FD8+49 call sub\_3140F3 test eax, eax jg short loc\_31307D call sub\_3140F3

 call
 sub\_3140F3

 jmp
 short

 icc\_31307D:
 ; CODE XREF: sub\_312FD8

 call
 sub\_3140F3

 and
 eax, 0FFFFh

 or
 eax, 80070000h

 loc\_31308C:
 ; CODE XREF: sub\_312FD8

### Basic Exploitation Strategy and the st eat eat and the st ea

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

int commit\_creds(struct cred \*new) {

. . .

			short loc 31306D	
			<pre>eax, [ebp+arg_0]</pre>	
ſ				
			[ebp+arg_4]	
			sub_314623	
			short loc_31306D	
			[ebp+arg_0], esi	
			short loc_31308F	
	loc_313066:			
			sub_31411B	
	loc_31306D:			
			sub 3140F3	
			short loc_31307D	
			short loc_31308C	
	loc_31307D:			
			sub_3140F3	
		and	eax, OFFFFh	

loc	31	30	8C	

; CODE XREF: SUD\_31 ebp+var 4], eax

### Basic Exploitation Strategy inv and the str

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!

sub 3140F3

# **Basic Exploitation Strategy**

Elevate Privi	ileg	push		4] 6 4]
The kernel is helpful again!			esi eax edi [ebp+arg_0], ea sub_31486A eax, eax	
<pre>struct cred *prepare_kernel_cred(struct</pre>	t task_	jz push struc nov	esi ctax csi, 100h	
}			esi [ebp+arg_4] edi sub_314623 eax, eax short loc_31306 [ebp+arg_0], es short loc_31308	
	loc_313066;		0Dh sub_31411B	
	loc_31306D:		<pre>sub_3140F3 eax, eax short loc_31307 sub_3140F3 short loc_31308</pre>	
	loc_31307D;	call and	sub_3140F3 eax, 0FFFFh	; CODE XREF: sub_312FD8
	loc_31308C:			

### Basic Exploitation Strategy Here ar (all sub\_314623 ear (ar (all sub\_314623 ear (all sub\_3

The kernel is helpful again!

struct cred \*prepare\_kernel\_cred(struct task\_struct \*daemon) {

	•••	[ebp+arg_4]
ļ		sub_314623
J		
		short loc_31306D
		<pre>[ebp+arg_0], esi</pre>
		short loc_31308F
"		

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			<pre>sub_3140F3 eax, eax short loc_31307D sub_3140F3 short loc_31308C</pre>	
	loc 31307D:			
	100_313070.		sub_3140F3	
		and	eax, OFFFFh	
	loc_31308C:			
			[ebp+var 4], eax	

### Basic Exploitation Strategy inv eax, [ebp+var\_84] Elevate Privilegestar, [ebp+var\_84] eax, [ebp+var\_84]

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

1 Sub\_31486A st eax, eax short loc\_31306D sh esi a eax, [ebp+arg\_0] sh eax 7 esi, 1D0h

push edi call sub\_314623 test eax, eax jz short loc\_31306D cmp [ebp+arg\_0], esi jz short loc 31308F

313066: ; CODE XREF: sub 312FI ; sub\_312FD8+55 push ODh call sub\_31411B

; sub\_31 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, OFFFTh or eax, 80070000h loc\_31308C: ; CODE XREF: sub\_312FD8

## Basic Exploitation S Elevate Privileg

Great! Now we can map out what we need to

- Create a "root" "struct creds" by calling prepare\_kernel\_cred( NULL );
- Call commit\_creds(root cred \*);

		sub_314623	
	test	eaxeax	
		short loc_31	
		eax, [ebp+var_7	
	emp	eax, [ebp+var_8	
		short loc_31306 bax, [ebp+var_8	
		esi	*
	pusn	:031	
_	push		
haad	to c	() () () () () () () () () () () () () (	
		sub 31486A	
		short loc_31306	
lina		<pre>eax, [ebp+arg_0</pre>	
ling			
		[ebp+arg_4]	
		edi	
		sub_314623 eax, eax	
		short loc_31306	
		[ebp+arg_0], es	
		short loc 31308	
313066:			
		sub_31411B	
_31306D:			
		sub_3140F3	
		eax, eax	
	jg call	short loc_31307 sub 3140F3	
		short loc_31308	
31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
_31308C:			

### Basic Exploitation Strategy inv and the str

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

- Create a "root" "struct creds" by calling prepare\_kernel\_cred( NULL );
- 2. Call commit\_creds(root cred \*);
- 3. Enjoy our new root privileges!



### Basic Exploitation Strategy Nov Set (all Sub\_314623 Basic Exploitation Strategy Nov Set (all Sub\_314623 Set (all Sub\_314633 Set (all Sub\_314633 Set (all Sub\_314633 Se

Why bother returning to Uppergraph			edi	
Why bother returning to Userspace?			<pre>[ebp+arg_0], ea sub_31486A</pre>	
			short loc_31300	
			eax, [ebp+arg_(	
			[ebp+arg_4]	
			sub_314623	
			short loc_31300	
			[ebp+arg_0], es	
			short loc_31308	5 E
	loc_313066:			
			sub_31411B	
	loc_31306D:			
			sub 3140F3	
			short loc_3130	0
			sub_3140F3	
			short loc_31308	
	1 010070-			
	loc_31307D:		sub 3140F3	
		and	eax, OFFFFh	
	100 2120901			

# Basic Exploitation Strategy Returning To UserSpace

Why bother returning to Userspace?

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

		short loc_31306	
nuch	eas	sier fron	n
		[ebp+arg_4]	
		sub 314623	
		short loc_31306D	
		[ebp+arg_0], esi	
		short loc_31308E	
313066:			
		sub_31411B	
_31306D:			
		sub_3140F3	
		short loc_313071	
		sub_3140F3	
		short loc_313080	
_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
_31308C:			
		[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

_	mov	eax, [epp+var_/u	
<b>Jser</b>		eax, [ebp+var 84	
JAAR		kiete	
	pus	Hes1	
	pusn	031	
$\mathbf{O}$			
<b>e</b> ?		[ebp+arg_0], eax	
		sub_31486A	
		eax, eax	
		short loc_31306D	
		esi	
are <i>much</i>	eas	sier fron	1
	MOV	esi, 1D0h	
		[ebp+arg_4]	
		sub_314623	
invito:		short loc_31306D	
/ay to:		[ebp+arg_0], esi	
		short loc_31308F	
loc 313066:			
		sub_31411B	
loc_31306D:			
		sub_3140F3	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
loc 31307D:			
		sub 3140F3	
	and	eax, OFFFFh	
loc_31308C:			

# Basic Exploitation Strategy Returning To UserSpace

How does the kernel do it?			[ebp+arg_0], eax	
			sub_31486A	
			short loc_31306D	
			<pre>eax, [ebp+arg_0]</pre>	
			[ebp+arg_4]	
			sub_314623	
			short loc_31306D	
			[ebp+arg_0], esi	
			short loc_31308F	
	loc_313066:			
			sub_31411B	
	loc_31306D:			
			sub_3140F3	
			short loc_31307D	
			sub_3140F3	
			short loc_31308C	
	loc_31307D:			
			sub 3140F3	
		and	eax, OFFFFh	
	loc_31308C:			

### Basic Exploitation Strategy Nov Basic Exploitation Strategy Returning To UserSpace

How does the kernel do it?			eax edi [ebp+arg_0], eax sub_31486A eax, eax short loc 313061	
push\$SS_USER_VALUEpush\$USERLAND_STACKpush\$USERLAND_EFLAGSpush\$CS_USER_VALUEpush\$USERLAND_FUNCTION_ADDRESS			esi eax, [ebp+arg_0] eax esi, 1D0h esi [ebp+arg_4] edi sub_314623 eax, eax short loc_31306] [ebp+arg_0], esi short loc_31308]	
swapgs iretq	loc_313066:		0Dh sub 31411B	
	loc_31306D:			
	;	call	sub_3140F3	; CODE XREF: sub_312FD8
	loc_31308C:		eax, OFFFFh eax, 80070000h [ebp+var 4], eau	

### Basic Exploitation Strategy Nov Basic Exploitation Strategy Returning To UserSpace

HOW (	loes the kernel do it?			[ebp+arg_0], ea	
				sub_31486A	
				eax, eax short loc 31306	D
					<i></i>
push	\$SS USER VALUE			eax, [ebp+arg_0	
push					
push	\$USERLAND STACK				
push				esi [ebp+arg_4]	
push	\$USERLAND EFLAGS			edi	
				sub_314623	
push	\$CS USER VALUE			eax, eax	
•				<pre>short loc_31306 [ebp+arg_0], es</pre>	
push	\$USERLAND_FUNCTION_ADDRESS			short loc 31308	
	· <u> </u>				
swapgs		loc_313066:			
irota					
iretq				sub_31411B	
		loc_31306D:			
				aul 3141F3	
I NIS V	<i>vill usually</i> get you out of "Kerr	nei ivioc	lest S	sarely.	
	<i>y b y</i>			short loc_31307	D
				sub_3140F3	
				short loc_31308	
		loc_31307D:			
			call	sub_3140F3	
				Cax, UFFFFI	

loc 31308C: ; CODE XREF: sub 312FD8

### Basic Exploitation Strategy Basic Exploitation Strategy bort loc\_31 Bort loc\_3

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

V	itseli	short loc_31306D	
J	push	esi	
		<pre>eax, [ebp+arg_0]</pre>	
		[ebp+arg_4]	
		sub_314623	
		short loc_31306D	
		[ebp+arg_0], esi	
		short loc_31308F	
66:			
		sub_31411B	
6D:			
		sub 3140F3	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
		sub_3140F3	
	and	eax, OFFFFh	
8C:			

# Basic Exploitation Strategy and and an article and article ar

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		
IS	high	Jaci	sub_3		
		test	eax, eax short loc_313061 esi		
	DV IL	seii	short loc_31306	D	
	J	push			
			eax, [ebp+arg_0]		
			[ebp+arg_4]		
			sub_314623		
			short loc_31306	D	
			[ebp+arg_0], est		
			short loc_31308		
	313066:				
			sub_31411B		
	31306D:				
			sub 3140F3		
			short loc 313071	D	
			sub 3140F3		
			short loc_313080	С	
loc 3	31307D:				
			sub 3140F3		
		and	eax, OFFFFh		
	31308C:				
			[ebp+var_4], eas		

### Basic Exploitation Strategy Basic Exploitation Strategy Basic Local Stra

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

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

AS

いして	SHOLF TOC 21300D
push	esi
	<pre>eax, [ebp+arg_0]</pre>
	[ebp+arg_4]
	sub_314623
	short loc_31306D
	<pre>[ebp+arg_0], esi</pre>
	short loc_31308F

loc\_313066:

; CODE XREF: sub 312FD

You need to be very careful about destroying Kernel state.

	loc_31306D:			
egfault probably means a reboo	ot!		<pre>sub_3140F3 eax, eax short loc_31307I sub_3140F3 short loc_313080</pre>	
	loc_31307D:		sub 3140F3	
		and or	eax, OFFFFh eax, 80070000h	
	loc_31308C:			
			[ebp+var 4], eax	

### Basic Exploitation Strategy Enjoying our Root Prise (100 - 31 - 201) Busic Exploitation Strategy (100 - 201) Busic Explored Bu

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

short loc 31306D	
eax, [ebp+arg_0]	
[ebp+arg_4]	
sub_314623	
short loc 31306D	
[ebp+arg_0], esi	
short loc_31308F	
sub_31411B	
sub_3140F3	
short loc_31307D	
sub_3140F3	
short loc_31308C	
sub_3140F3	
eax, OFFFFh	

loc\_31308C:

[ebp+var 4], eax

### Basic Exploitation Strategy box ex, [ebp+var\_90] Enjoying our Root Prive et [ebp+var\_94] box et [ebp+var\_94] est

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

We can do whatever we want!

		short loc_31306	D
		<pre>eax, [ebp+arg_0</pre>	
		[ebp+arg_4]	
		sub 314623	
		short loc_31306	D
		[ebp+arg 0], es	
		short loc 31308	
13066:			
1306D:			
		sub 3140F3	
		short loc 31307	D
		sub_3140F3	
			C
1307D:			
		sub 3140F3	
	and	eax, OFFFFh	
1308C:			

Kernel Space Pro	push call test jz cmp tecyt jb sub push	eax, [ebp+var_84 esi	
By now, you're familiar with the alphabe	et soup	of explo	oit
mitigations	test jz push lea push mov push push call test jz cmp jz 13066: push call		
	1306D: call test jg call jmp 1307D: call	<pre>sub_3140F3 eax, eax short loc_31307D sub_3140F3 short loc_31308C</pre>	
loc_3	and or 1308C:	eax, OFFFFh eax, 80070000h	; CODE XREF: sub_312FD8

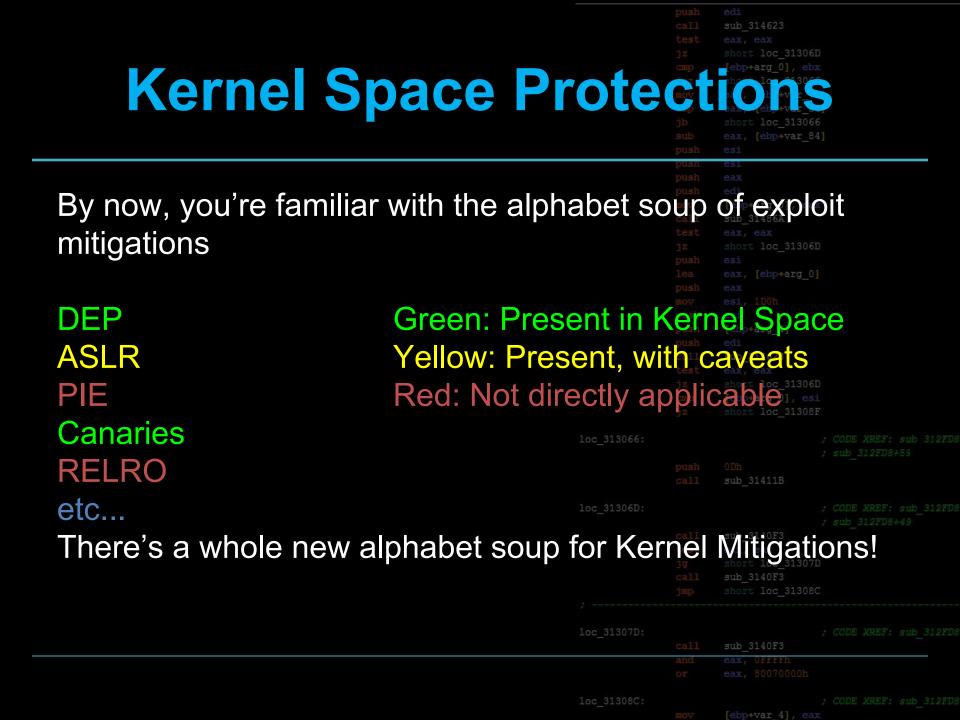
Kernel Space P	pus cal jz jz <b>rotec</b> jb sut pus	<pre>sub_314623 st eax, eax short loc_313 ebp+arg_0], holofter short loc_313 st eax, [ebp+varg_0], short loc_313 b eax, [ebp+varg_0]</pre>	ebx 30 50 50 50 50
By now, you're familiar with the alpl mitigations		an edi p of exp st eax, eax short loc_31: a eax, [ebp+arg	
DEP ASLR PIE		v esi, 1D0h sh esi sh [ebp+arg_4] sh edi 11 sub_314623 st eax, eax short loc_31: p [ebp+arg_0],	
Canaries RELRO	loc_313066: pus cal		
etc	loc_31306D: cal tes jg cal jm ;	st eax, eax short loc_31:	
	loc_31307D: cal and or loc 31308C:	d eax, OFFFFh	; CODE XREF: sub_312FD8

# Kernel Space Protection

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

DEP ASLR PIE Canaries RELRO etc... Green: Present in Kernel Space Yellow: Present, with caveats Red: Not directly applicable

loc_313066:			
loc_31306D:			
loc_31307D:		sub_3140F3	
	and or	eax, OFFFFh eax, 80070000h	
loc_31308C:		[ebp+var_4], eax	



# Kernel Space Protections

Some new words in our soup

MMAP\_MIN\_ADDR KALLSYMS RANDSTACK STACKLEAK SMEP / SMAP

	jb	short loc_313066
		eax, [ebp+var_84]
	pusn	651
		[ebp+arg_0], eax
		sub_31486A
		short loc_31306D
		eax, [ebp+arg_0]
		[ebp+arg_4]
		sub_314623
		short loc_31306D
		[ebp+arg_0], esi
		short loc_31308F
loc_313066:		
		sub_31411B
loc_31306D:		
		sub 3140F3
		short loc_31307D
		sub_3140F3
		short loc_31308C
loc_31307D:		
		sub_3140F3
	and	eax, OFFFFh
loc 31308C:		
		[ebp+var_4], eax

sub 314623

Kernel Space Pr	ote	push call test jz cmp jb sub push	edi sub_314623 eax, eax short loc_31306 [ebp+arg_0], eb bhc_lot_at306 eax, [ebp+var_8 esi esi	x 6
Some new words in our soup (There's pla	enty more)		eax edi [ebp+arg_0], ea sub_31486A eax, eax	
MMAP_MIN_ADDR KALLSYMS RANDSTACK STACKLEAK SMEP / SMAP	loc_313066:		<pre>short loc_31306 esi eax, [ebp+arg_0 eax esi, 1D0h esi [ebp+arg_4] edi sub_314623 eax, eax short loc_31306 [ebp+arg_0], es short loc_31308</pre>	
Most of these will be off for the labs!	loc_31306D:		ODh sub_31411B sub_3140F3 eax, eax short loc_31307 sub_3140F3 short loc_31308	
	loc_31307D: loc_31308C:	call and or	sub_3140F3 eax, 0FFFFh eax, 80070000h	; CODE XREF: sub_312FD8 ; CODE XREF: sub_312FD8

### Kernel Space Protect ioc\_313066 ax, eax abort loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] est

push

This makes exploiting NULL pointer dereferences harder.

	short loc_31306D
	eax, [ebp+arg_0]
	[ebp+arg_4]
	sub_314623
	short loc_31306D
	[ebp+arg_0], esi
	short loc 31308F
	sub_31411B
	sub_3140F3
	short loc_31307D
	sub_3140F3
	short loc_31308C
	sub_3140F3
and	eax, OFFFFh

### Kernel Space Protections inz short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] esi

pusn pusn

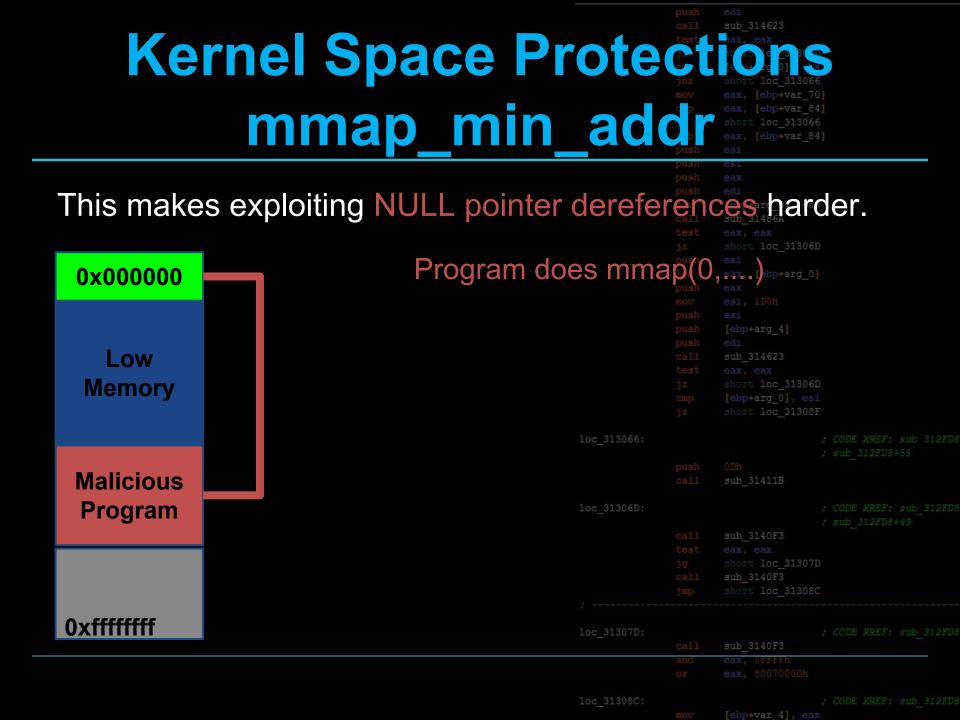
### This makes exploiting NULL pointer dereferences harder.

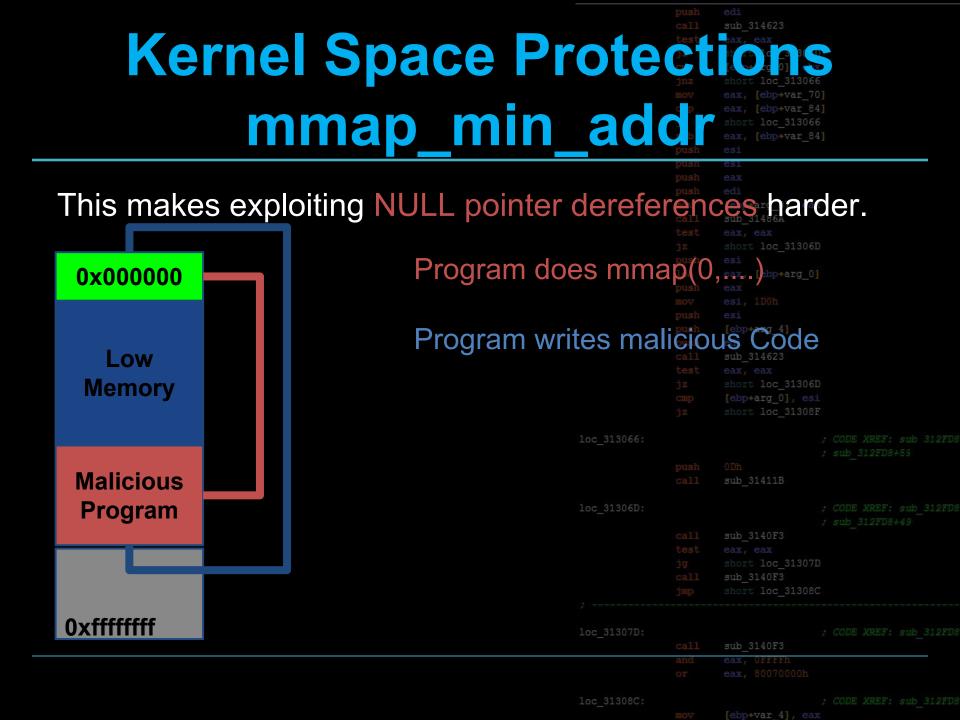
Low Memory **Malicious** Program Oxfffffff

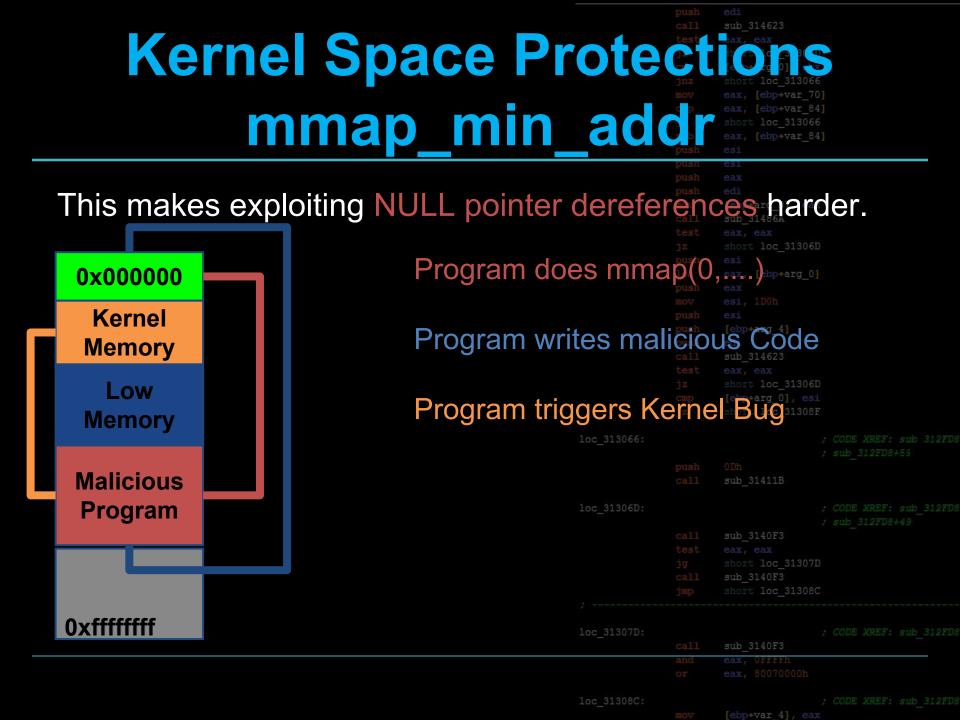
> and eax, OFFFFh or eax, 80070000h

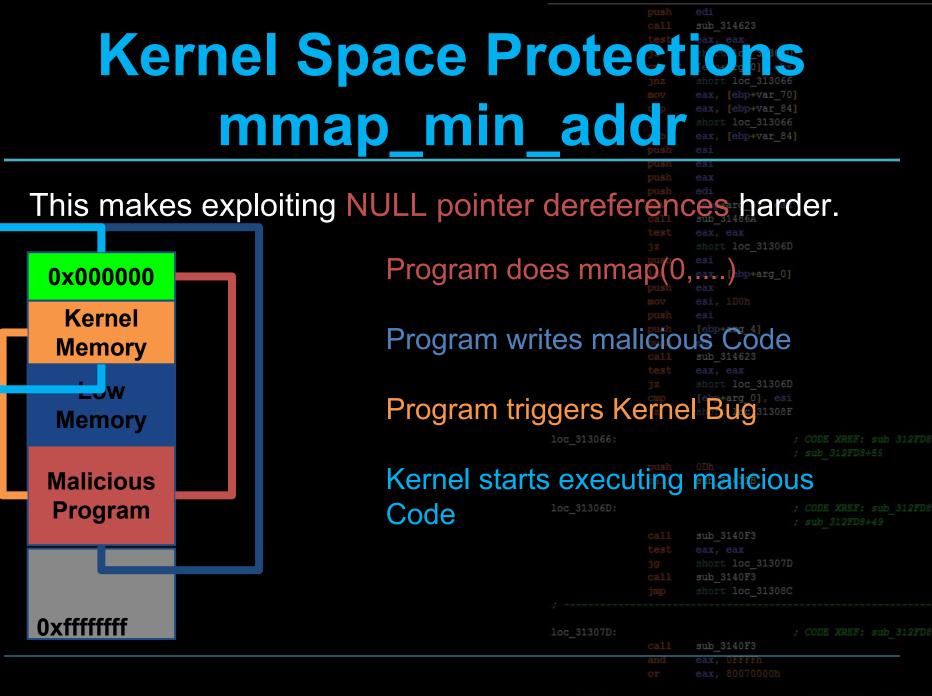
loc\_31308C:

; CODE XREF: sub\_3: ebp+var 4], eax







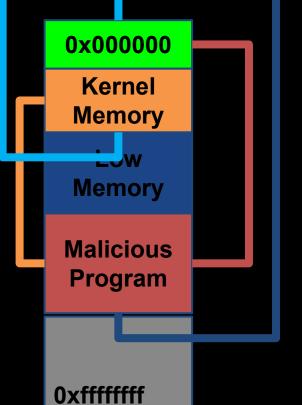


	21		R	
TOUR	ο.	L ()	u,	

; CODE XREF: sub\_ [ebp+var 4]. eax

### Kernel Space Protect ion 313066 wv sub 314623 sub 31

### 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.

; CODE XREF: sub 312FD8 ; sub\_312FD8+59

loc_31306D:			
;			
	and or	eax, OFFFFh eax, 80070000h	
100 313080			

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

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

		short loc_31308H	
loc_313066:			
		sub_31411B	
loc_31306D:			
		sub_3140F3	
		short loc_31307I	
		sub_3140F3	
		short loc_313080	
loc_31307D:			
	call	sub_3140F3	
		eax, OFFFFh	
loc_31308C:			

/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

fffffff810908c0 T commit\_creds fffffff81b01390 R \_\_ksymtab\_commit\_creds ffffffff81b1cf38 r \_\_kcrctab\_commit\_creds ffffffff81b2c33b r kstrtab commit\_creds

	xpic		outan
		sub 314623	
		short loc 31306	
		[ebp+arg 0], es	
		short loc_31308	
loc 313066:			
		sub_31411B	
loc 31306D:			
	call	sub 3140F3	
		short loc_31307	P
		sub 3140F3	
			C
Loc 31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc_31308C:			

kallsyms used to be world-readable.

Now, it returns 0's for unprivileged users

\$: cat /proc/kallsyms | grep commit\_creds

	sub_314623
test	ax, eax
jnz	short loc_313066
	eax, [ebp+var 70]
	eax, [ebp+var_84]
	short loc_313066
	eax, [ebp+var_84]
pusn	esi
	[ebp+arg_0], eax
	sub_31486A
	short loc_31306D
	<pre>eax, [ebp+arg_0]</pre>
	[ebp+arg_4]
	sub_314623
	short loc_31306D
	<pre>[ebp+arg_0], esi</pre>
	short loc_31308F
	sub_31411B
	sub 3140F3

### Can still be a useful source of information on older systems

loc_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc_31308C:		[ebp+var 4]. e	

# Kernel Space Protectic SMEP / SMAP

**SMEP: Supervisor Mode Execution Protection** 

Introduced in Intel IvyBridge

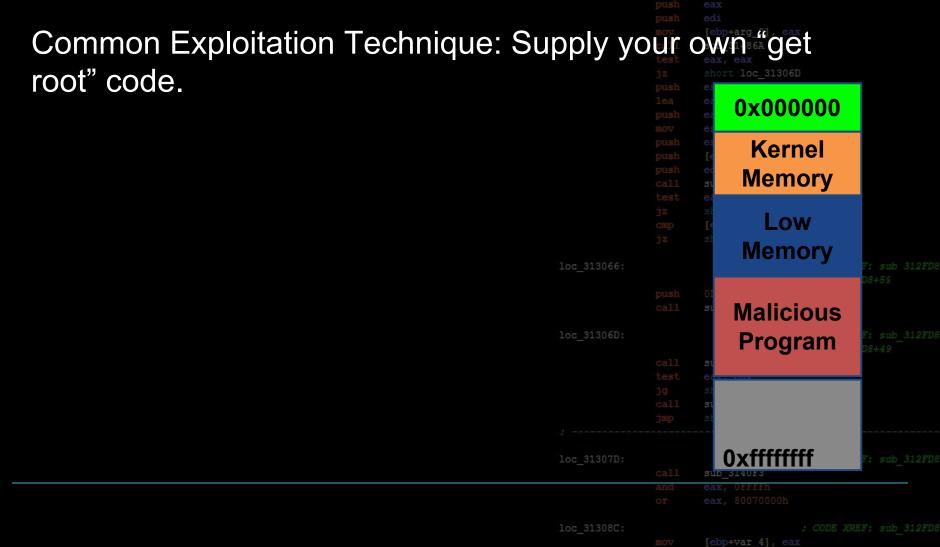
### SMAP: Supervisor Mode Access Protection

Introduced in Intel Haswell

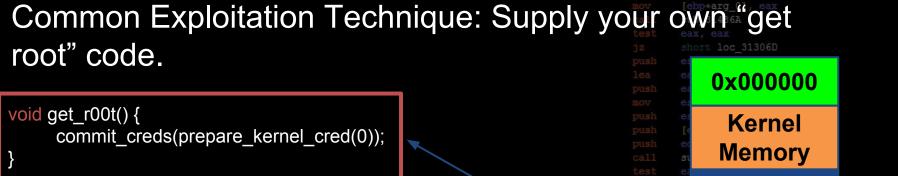
		sub_314623	
	test	eax, eax	
<b>- Y</b>			
	jnz	short loc_313066	
		eax, [ebp+var_70]	
		eax, [ebp+var 84]	
		short loc_313066	
		eax, [ebp+var 84]	
	pusn	esi	
		[ebp+arg_0], eax	
CTI	on	sub_31486A	
	test		
		short loc 31306D	
		esi	
		eax, [ebp+arg_0]	
		eax	
		[ebp+arg_4]	
		edi	
		sub_314623	
		short loc_31306D	
n		[ebp+arg_0], esi	
		short loc_31308F	
		; CODE XREF: sub 312FL ; sub 312FD8+59	
		0Dh	
		sub_31411B	
		: CODE XREF: sub 312FI	
		sub_3140F3	
		eax, eax	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
		; CODE XREF: sub_312FL	
	call	sub_3140F3	
	and	eax, OFFFFh	

TOC 212000	

### Kernel Space Protections inz test toc\_313066 ax, eax ax, eax av, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] ex, [ebp+var\_84



### Kernel Space Protections inz sub\_314623 test test to\_313066 inz short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066

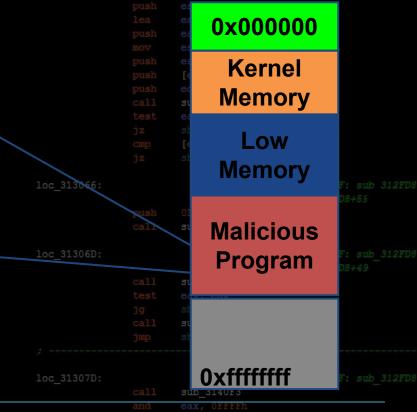


```
int main(int argc, char * argv) {
```

```
trigger_fp_overwrite(&get_r00t);
```

```
//trigger fp use
trigger_vuln_fp();
// Kernel Executes get_r00t
```

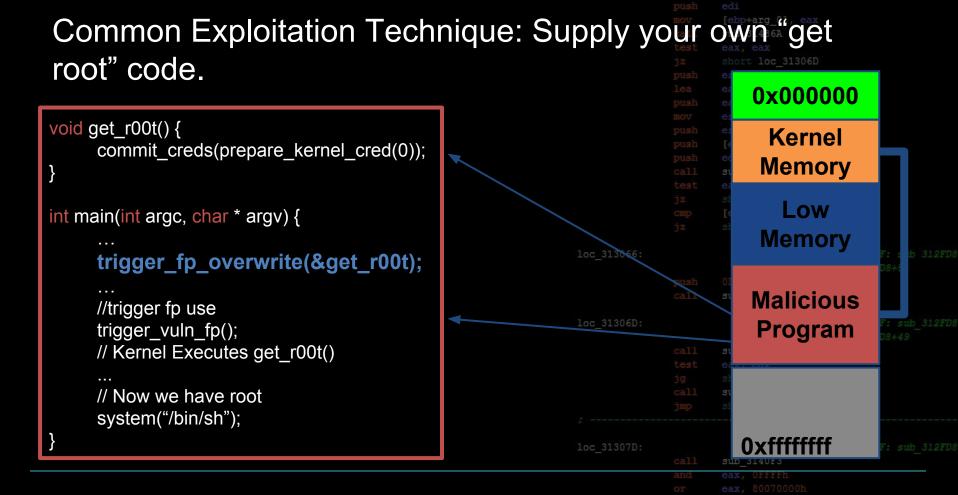
```
// Now we have root
system("/bin/sh");
```



loc\_31308C:

; CODE XREF: sub\_312F p+var\_4], eax

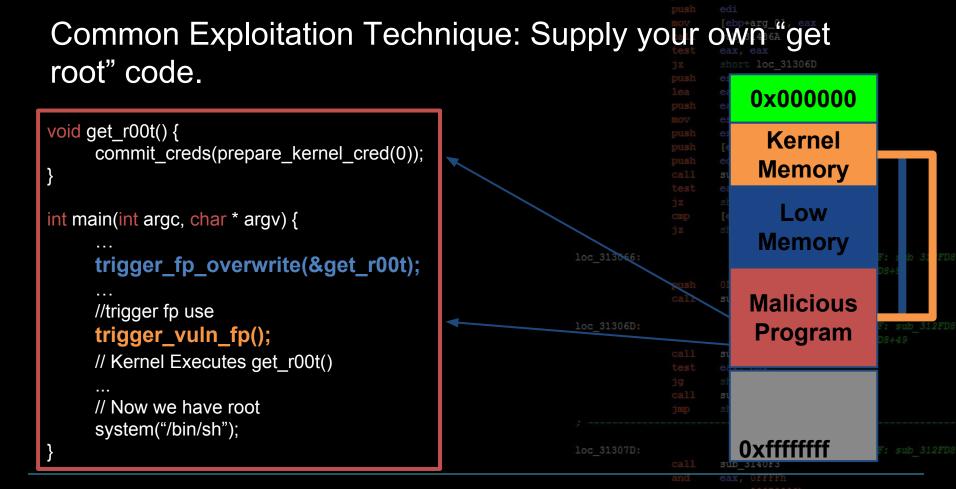
### Kernel Space Protections inz sub\_314623 test test to\_313066 inz short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] sub short loc\_313066



Loc\_31308C:

; CODE XREF: sub\_312F1 pp+var\_4], eax

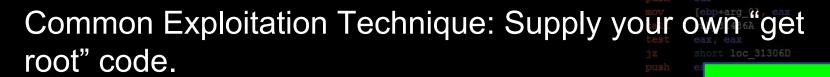
### Kernel Space Protections inz sub\_314623 test test to\_313066 inz short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] eax, [ebp+var\_84] eax, [ebp+var\_84] eax, [ebp+var\_84]

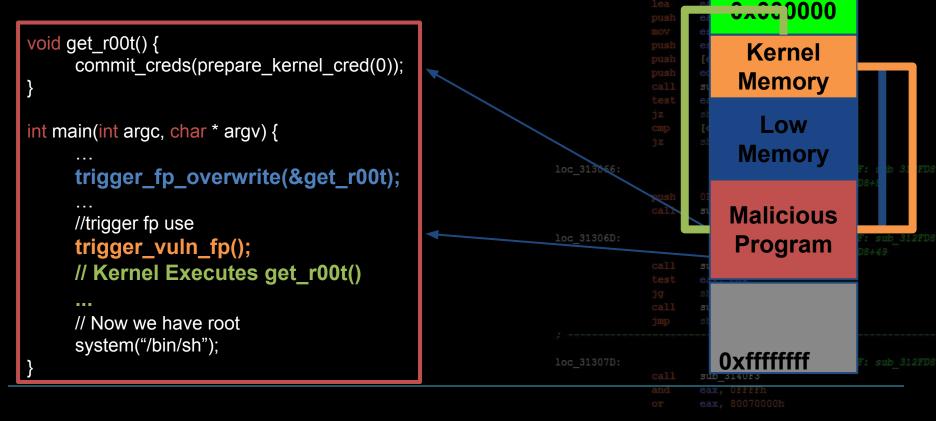


c\_31308C:

; CODE XREF: sub\_312F. p+var 4], eax

### Kernel Space Protections int sub\_314623 test test toc\_313066 int sub\_314623 test toc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] sub stort loc\_313066 eax, [ebp+var\_84] sub stort loc\_313066 eax, [ebp+var\_84] ex, [ebp+var\_84]

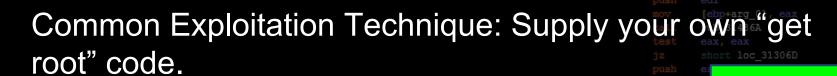




loc\_31308C:

; CODE XREF: sub\_312B p+var 4], eax

### Kernel Space Protections inz sub\_314623 test test to\_313066 inz short loc\_313066 eax, [ebp+var\_70] eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066 eax, [ebp+var\_84] short loc\_313066





; CODE XREF: sub\_3121

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".

	lea	eax, [ebp+arg_0]	
		[ebp+arg_4]	
		sub_314623	
		short loc_31306D	
		[ebp+arg_0], esi	
		short loc_31308F	
loc 313066:			
loc 31306D:			
		sub 3140F3	
		short loc_31307D	
		short loc 31308C	
loc 31307D:			
	call	, sub 3140F3	
	and	eax, OFFFFh	
loc_31308C:			
		[ebp+var 4], eax	

**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

		short loc_31308F	
loc_313066:			
		sub_31411B	
loc_31306D:			
		sub_3140F3	
		short loc_31307D	
		sub_3140F3	
		short loc_31308C	
loc_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc_31308C:			
		[ebp+var_4], eax	

**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

loc_31306D:			
		<pre>sub_3140F3 eax, eax short loc_31307 sub_3140F3 short loc_31308</pre>	
loc_31307D:		sub_3140F3	
	and or	eax, OFFFFh eax, 80070000h	
loc_31308C:			



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

+arg 4

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

loc_31307D:				
		sub_3140F3		
	and	eax, OFFFFh		
loc 31308C:				
		[ebp+var 4], ea		

## Example

	sub_314623
	short loc_31306D
	[ebp+arg_0], ebx
	short loc_313066
	eax, [ebp+var_70]
	eax, [ebp+var_84]
	short loc_313066
	eax, [ebp+var_84]
pusn	

## We'll walk through a short example of a backdoored LKM to

get a feel for dealing with the kernel.

		short loc_31306D
		eax, [ebp+arg_0]
		[ebp+arg_4]
		sub_314623
		short loc_31306D
		[ebp+arg_0], esi
		short loc_31308F
66:		
		sub_31411B
6D:		
	call	sub 3140F3
		short loc_31307D
		sub 3140F3
		short loc 31308C
		; CODE XREF: sub_312FD8
	call	sub_3140F3
	and	eax, OFFFFh
8C:		
		[ebn_war 4] eav

## Conclusion

	sub_314623
	short loc_31306D
	[ebp+arg_0], ebx
	short loc_313066
	<pre>eax, [ebp+var_70]</pre>
	<pre>eax, [ebp+var_84]</pre>
	short loc_313066
	<pre>eax, [ebp+var_84]</pre>
usn	851

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

		short loc 31306	
		eax, [ebp+arg_0	
		[ebp+arg_4]	
		sub_314623	
		short loc_31306	
		[ebp+arg_0], es	
		short loc_31308	
_313066:			
		sub_31411B	
04.00.00			
_31306D:			
		sub_3140F3	
		eax, eax	
		short loc_31307	
		sub_3140F3	
		short loc_31308	
: 31307D:			
		sub 3140F3	
	and	eax, OFFFFh	
_31308C:			

## Conclusion

	sub_314623
	short loc_31306D
	[ebp+arg_0], ebx
	short loc_313066
	<pre>eax, [ebp+var_70]</pre>
	<pre>eax, [ebp+var_84]</pre>
	short loc_313066
	<pre>eax, [ebp+var_84]</pre>
pusn	051

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

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

		eax, eax short loc_31306D [ebp+arg_0], esi short loc_31308F
loc_313066:		; CODE XREF: sub 312FD ; sub_312FD8+55 ODh sub_31411B
loc_31306D:		; CODE XREF: sub_312FD; ; sub_312FD8+49 sub_3140F3 eax, eax short loc_31307D sub_3140F3 short loc_31308C
;	call and or	; CODE XREF: sub_312FD; sub_3140F3 eax, OFFFFh eax, 80070000h
loc_31308C:		; CODE XREF: sub_312FD: [ebp+var_4], eax

## Conclusio

Conclusion	push call test jz cmp jnz mov cmp jb sub sub	edi sub_314623 eax, eax short loc_31306 [ebp+arg_0], eb short loc_31306 eax, [ebp+var_7] eax, [ebp+var_8 short loc_31306 eax, [ebp+var_8 esi	« 5 2] 4] 5
	push push	esi eax	
Kernel Exploitation is weird, but extreme	push Me test jz push	ecn (ebp+arg 0], ea (bp-acc) (bp-acc) (construction) (con	20
As userland exploit-dev becomes more challeng expensive, kernelspace is becoming a more attr	push	[ebp+arg 4]	re
		eax, eax short loc_31306	
A single bug can be used to bypass sandboxes,	and	gain roc	i 5 ; CODE XREF: sub 312F
privileges, which may otherwise be impossible		0Dh sub_31411B	
loc_31306D:			
		sub_3140F3 eax, eax short loc 31307	

loc_31306D:			
		sub_3140F3	
		short loc_31307	D
		sub_3140F3	
		short loc_31308	C
loc_31307D:			
		sub_3140F3	
	and	eax, OFFFFh	
loc 31308C:			

## Conclusion

	sub_314623
	short loc_31306D
	[ebp+arg_0], ebx
	short loc_313066
	eax, [ebp+var_70]
	eax, [ebp+var_84]
	short loc_313066
	eax, [ebp+var_84]
pusn	esi
	[ebp+arg_0], eax
	sub_31486A

## Copyrighted Material NGRE A GUIDE TO KERNEL EXPLOITATION Attacking the Core Ennico Perla Massimiliano Oldani Copurishing Malyrich

### The book on Kernel Exploitation: