CS5231: Systems Security

Lecture 6: Rootkit

What is a Rootkit?

- A rootkit is a set of programs and code that allows a permanent or consistent, undetectable presence on a computer.
- Goals:
  - **Hide malicious resources** (*e.g., processes, files, registry keys, open ports, etc.*)
  - **Provide hidden backdoor access**

What a Rootkit isn't?

- A rootkit does **NOT** compromise a host by itself
  - An exploit must be used to gain access to the host before a rootkit can be deployed
- The purpose of a rootkit is **NOT** to gain access to a system, but to preserve existing access
  - Rootkits hide processes, ports, files, and other resources from the OS and security programs

Brief History

- Early rootkits target
  - First "rootkit" appeared on SunOS in 1994
  - Replaced *login*, *ls*, *ps*, *netstat*, etc. to give an attacker hidden access
  - "Kits" to attain and maintain "root" access to machines
  - Eventually moved towards other platforms and kernel
- Windows popularity brought Windows rootkits

Why are Rootkits So Popular?

- Worms, trojans, malware are utilizing rootkits
  - Presence becomes hidden
  - Machines stay infected longer → can send spam and steal info longer → more money for attacker
- Some commercial software adopts rootkit technology
  - Sony DRM software

How Rootkits are Used?

A Staged View of an Attack

1. Vulnerability in a system is discovered
2. Vulnerability is exploited to gain access to the system
3. Attacker gains foothold on the system by escalating privileges, installing backdoor etc.
4. Attacker utilizes system access to steal info., launch other attacks, etc
5. Compromise is detected, and incident response is executed

Rootkits help an attacker to gain a stronger foothold on the system

Rootkits prevent or delay the detection by hiding an attacker's resource on the system

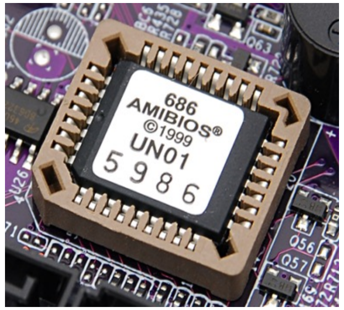
Rootkits - How They Work

- To hide in a system what you need to do?
  - Control the system
  - Act as a gatekeeper between what a user sees and what the system sees
  - Require administrator privileges to install

Rootkits – How They Work

- To hide what is taking place, an attacker wants to:
  - ***Survive*** system restart
  - ***Hide*** processes
  - ***Hide*** services
  - ***Hide*** listening TCP/UDP ports
  - ***Hide*** kernel modules (or drivers)

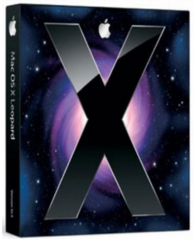
System Booting Sequence



BIOS

Drive

Operating System

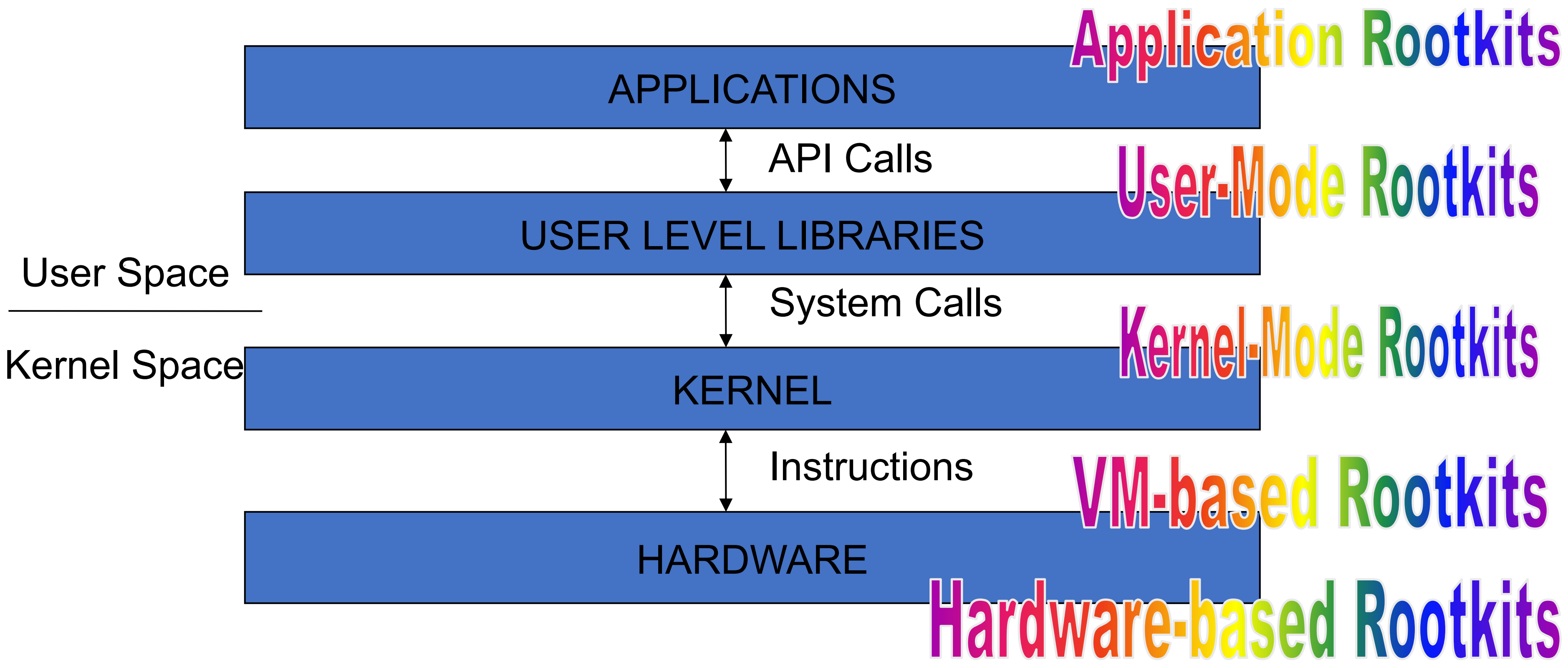


Data/Scripts

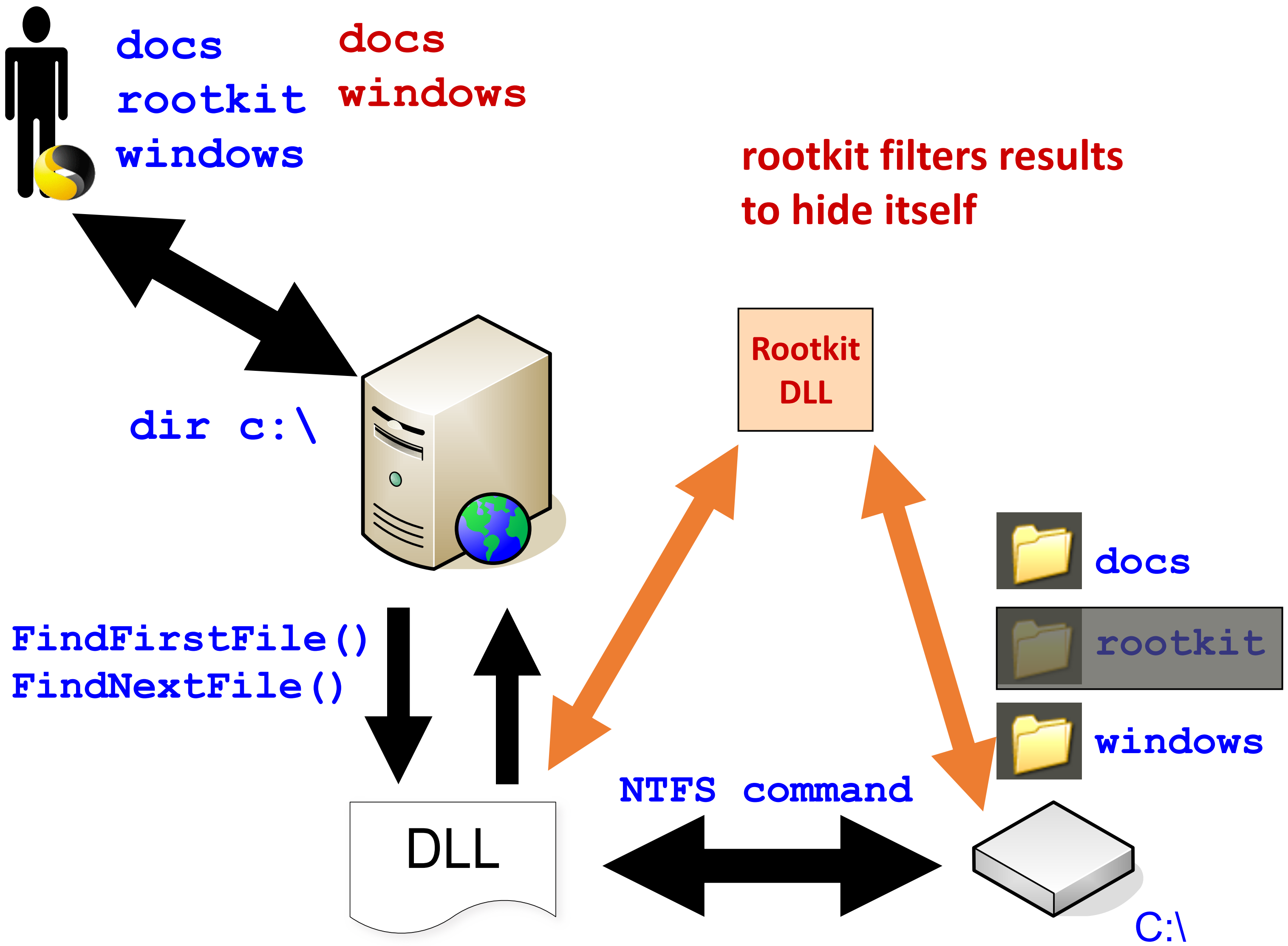
Microsoft® Office™

Application

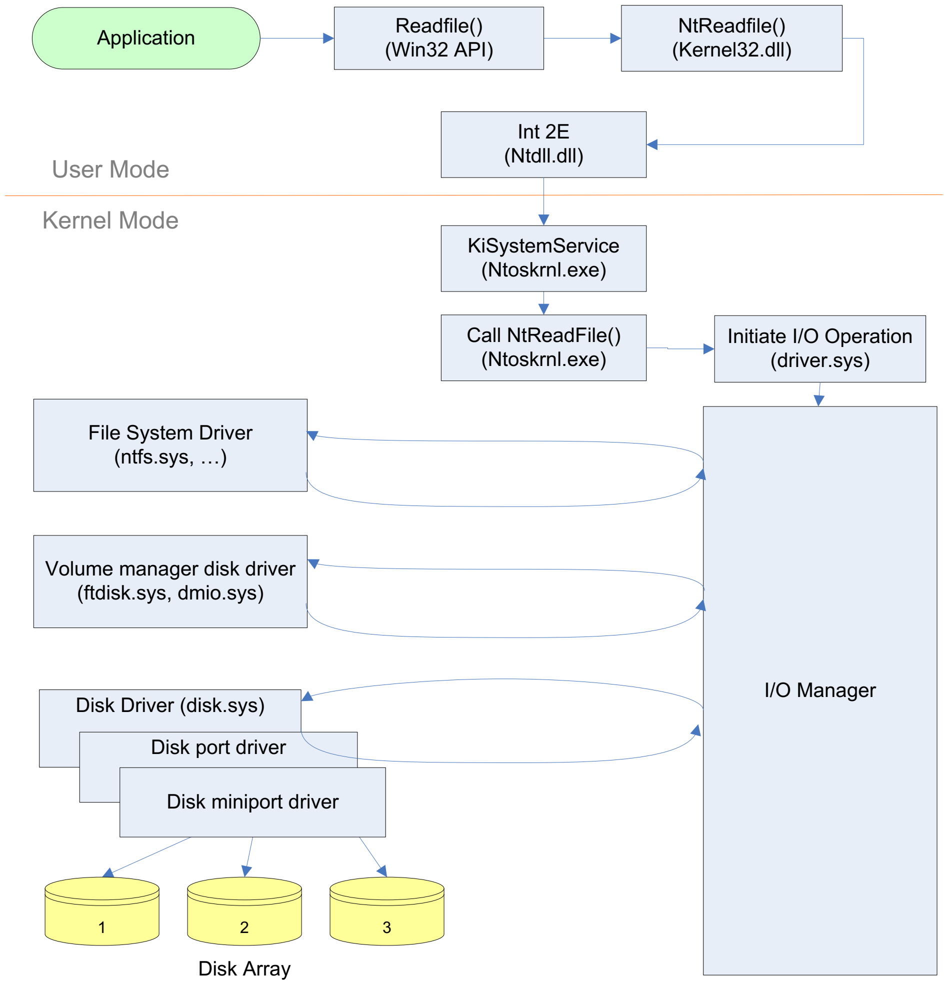
Different Types of Rootkits



How Rootkits Work

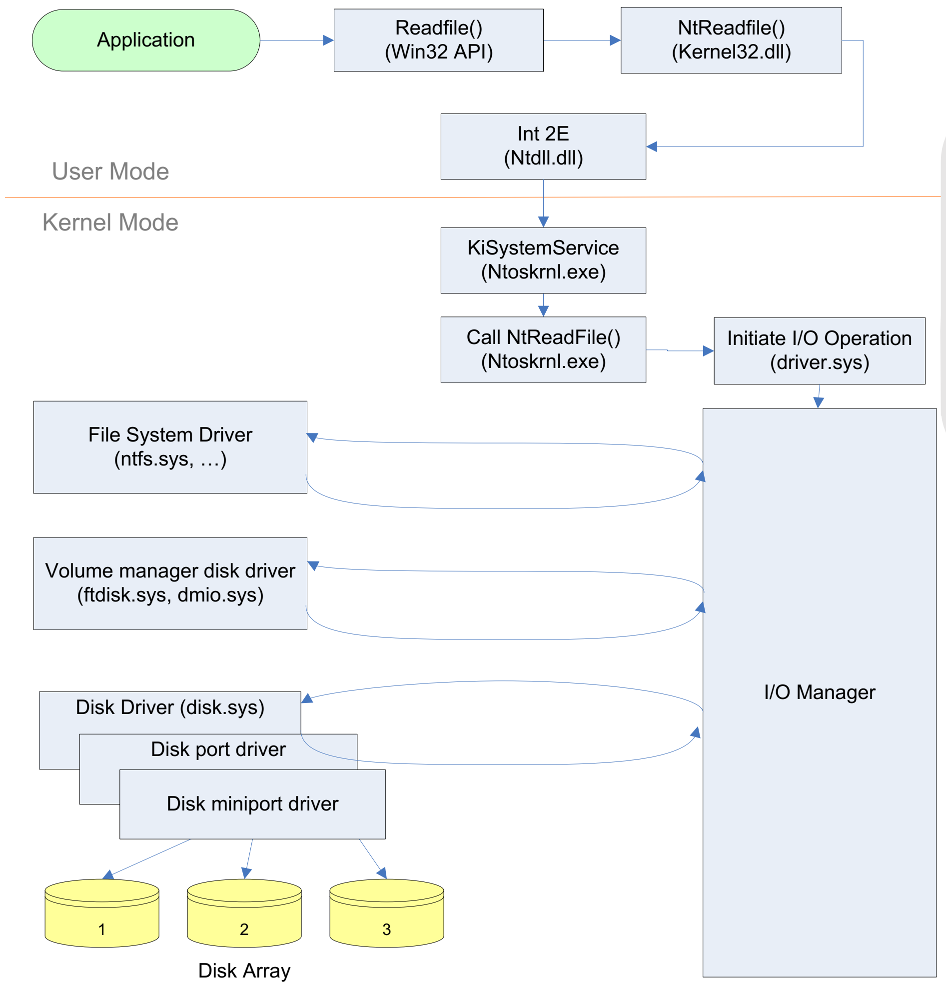


What Happens When You Read a File?



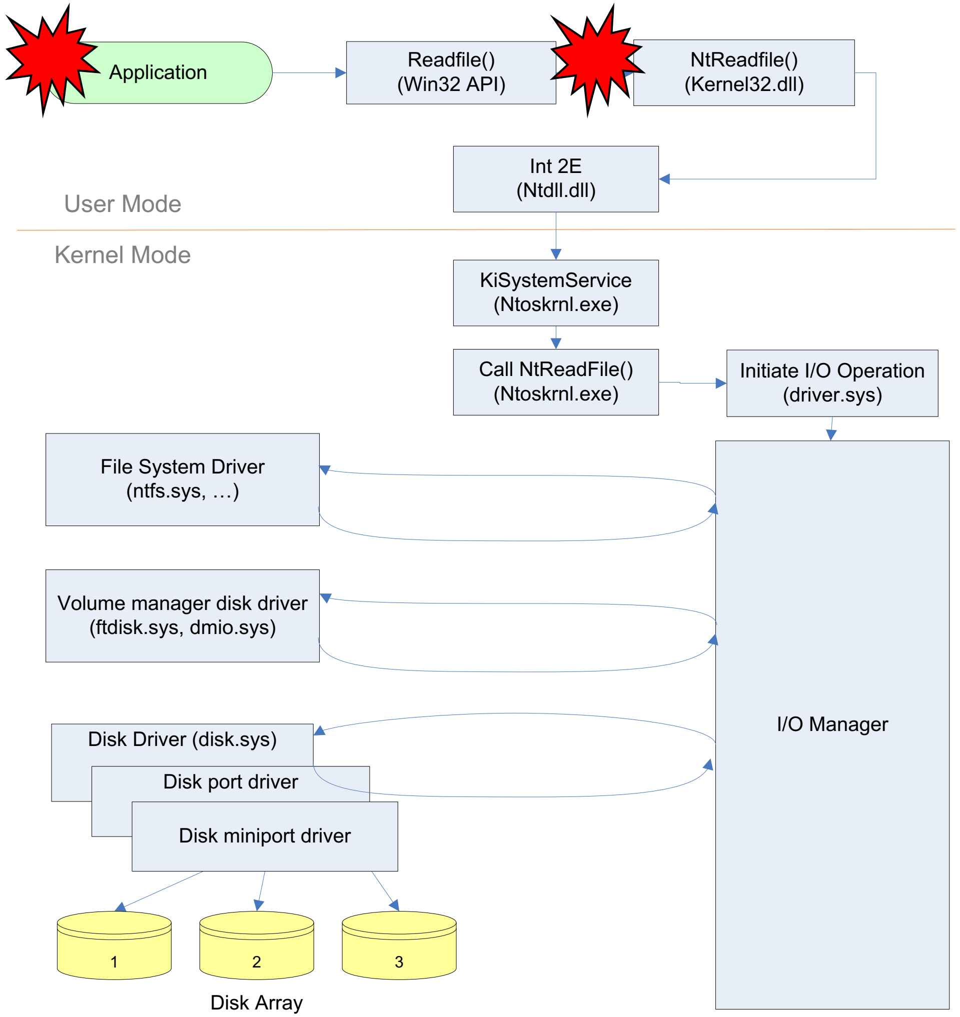
- ☐ Readfile() called on File1.txt
- ☐ Transition to Ring 0
- ☐ NtReadFile() processed
- ☐ I/O Subsystem called
- ☐ IRP generated
- ☐ Data at File1.txt requested from ntfs.sys
- ☐ Data on D: requested from dmio.sys
- ☐ Data on disk 2 requested from disk.sys

What Happens When You Read a File?



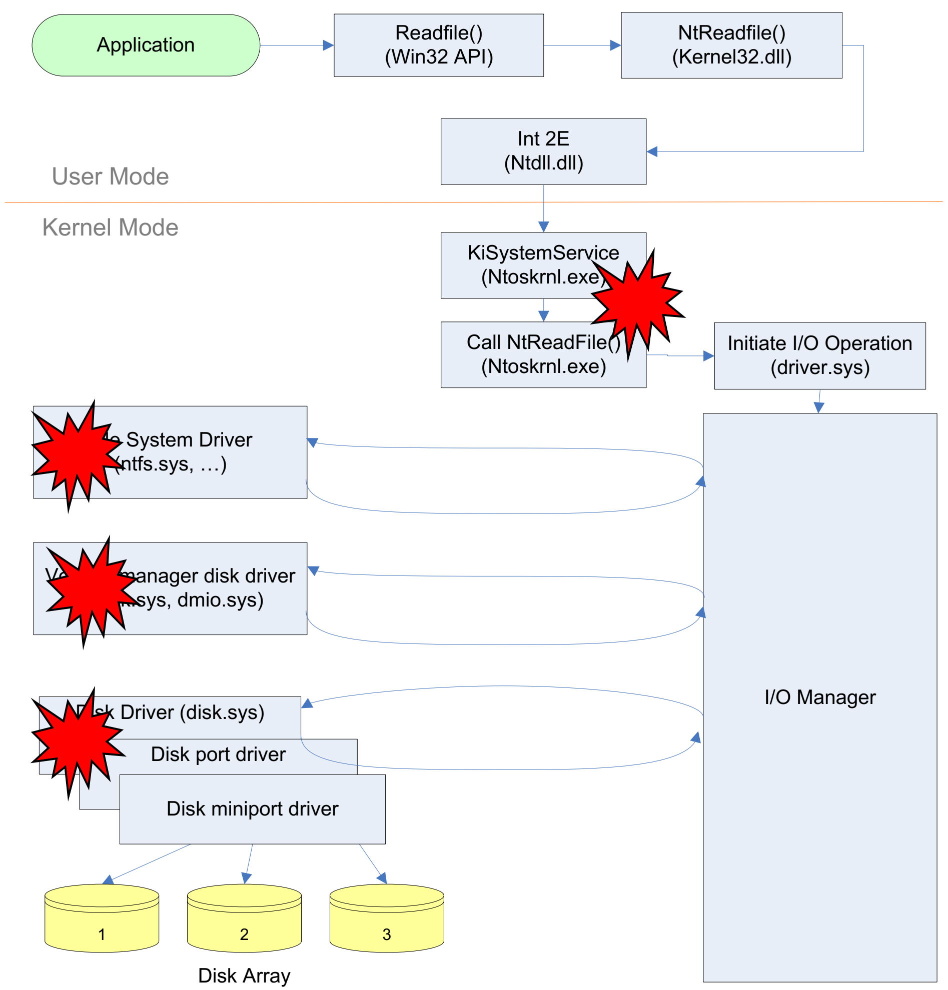
Question:
Where to hijack for a rootkit?

Usermode Rootkits



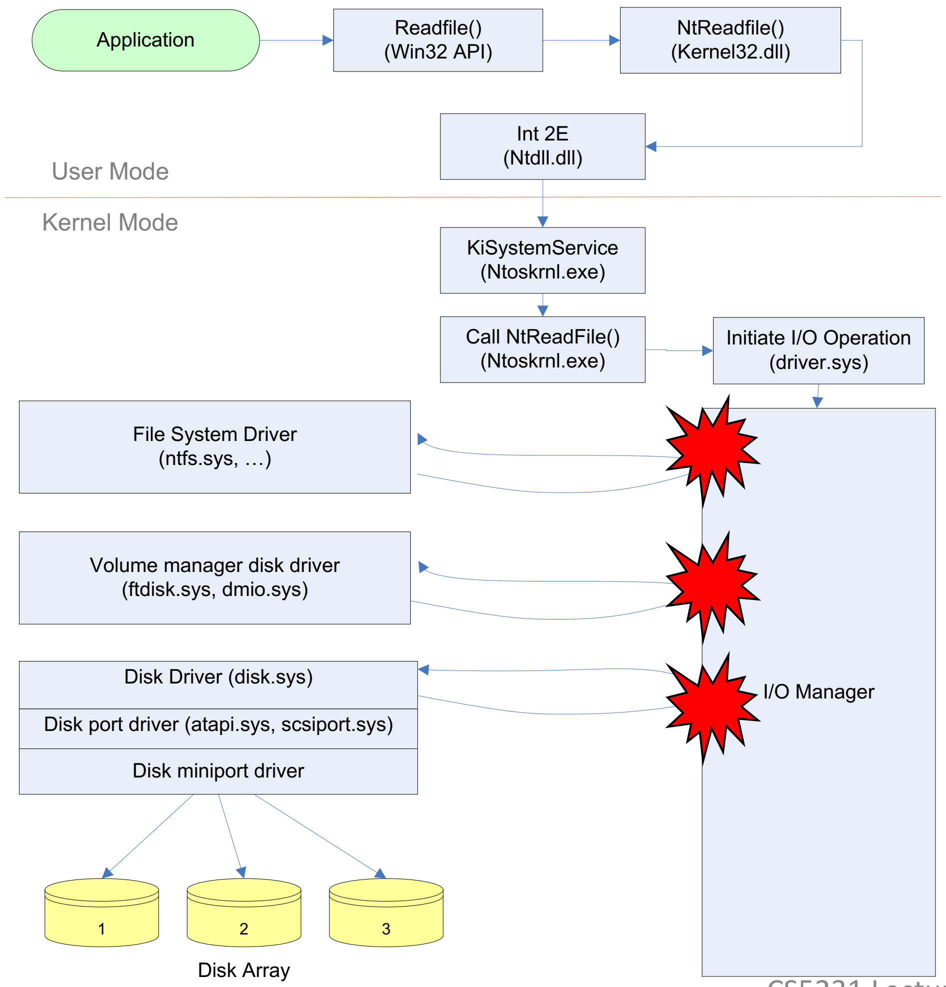
- Binary replacement
  - e.g., *modified EXE/DLL*
- Binary modification in memory
- User land hooking
  - e.g., *Hacker Defender, NTIllusion*

Kernel Rootkits



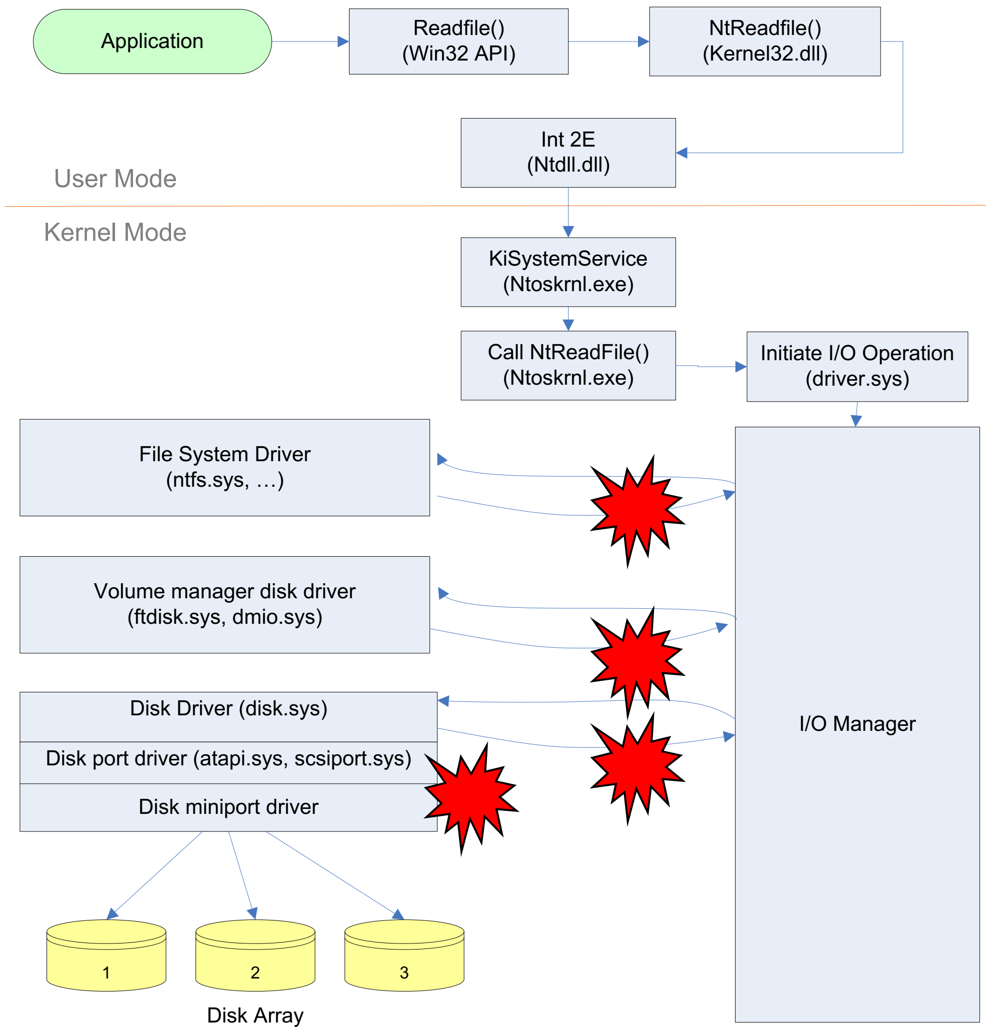
- Kernel hooking
  - e.g. *NtRootkit*
- Driver replacement
  - e.g., replace ntfs.sys with ntfss.sys
- Direct Kernel Object Manipulation – DKOM
  - e.g., *Fu, FuTo*

Kernel Rootkits



- IO Request Packet (IRP) Hooking
  - IRP Dispatch Table
  - e.g., *He4Hook* (some versions)

Kernel Rootkits



- Filter Drivers
  - The official Microsoft method
- Types
  - File system filter
  - Volume filter
  - Disk Filter
  - Bus Filter
  - e.g., *Clandestine File System Driver* (CFSD)

Current Rootkit Capabilities

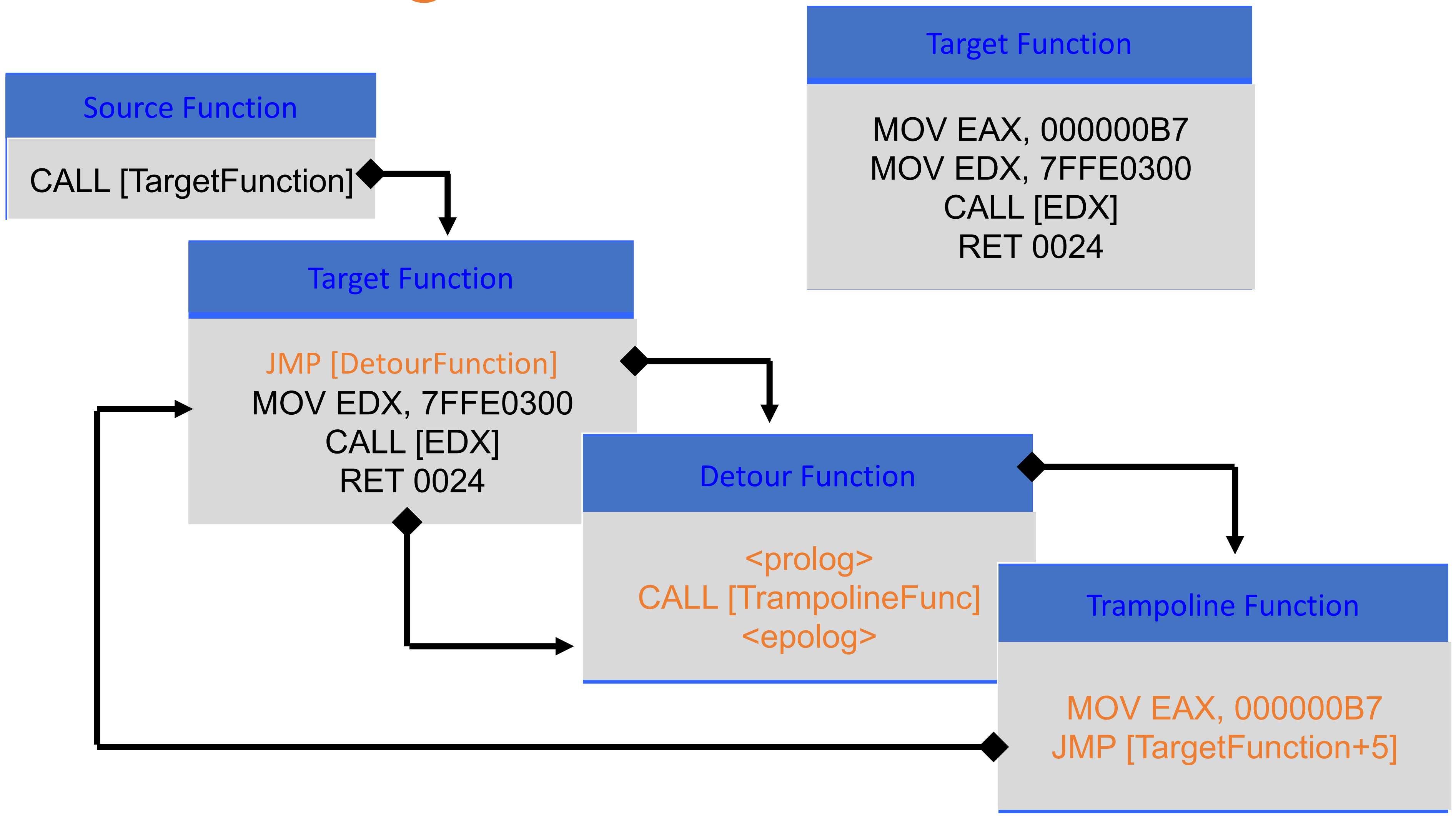
- Hide processes
- Hide files
- Hide registry entries
- Hide services
- Completely bypass personal firewalls
- Undetectable by anti virus
- Remotely undetectable
- Covert channels - undetectable on the network
- Install silently
- All capabilities ever used by viruses or worms

Basic Rootkit Techniques

- Inline hooking
- Import Address Table (IAT) hooking
- Export Address Table (EAT) hooking

- System Service Table (SSDT) hooking
- Interrupt Table hooking
- I/O Request Packet hooking
- Filter drivers
- Kernel object manipulation

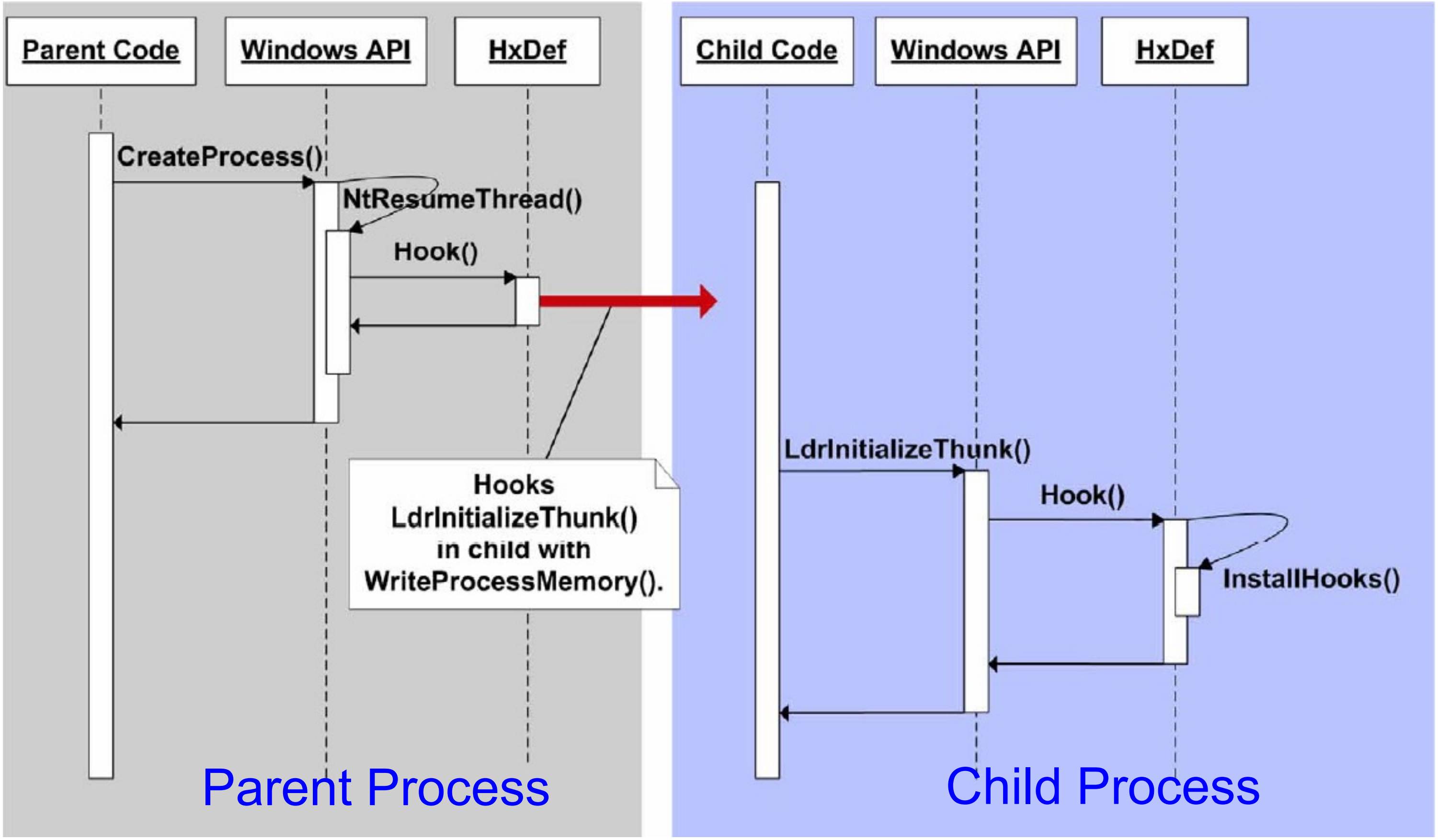
Inline Hooking



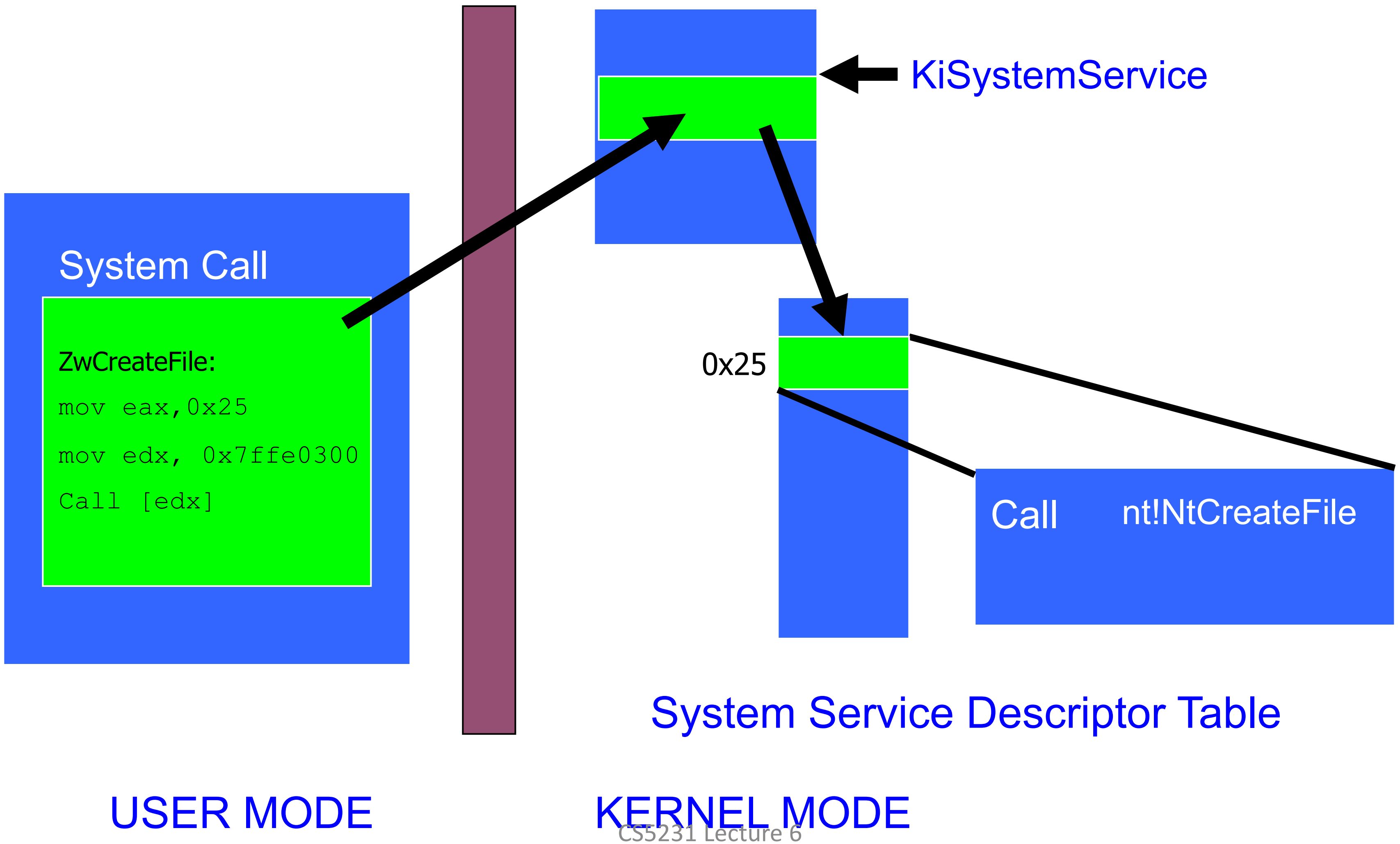
Rootkit Example: Hacker Defender

- One of the most popular rootkits in the wild
  - User-mode rootkit
  - Feature rich (hiding processes, TCP ports, etc.)
  - Very stable and portable
- Modifies the execution path of several native Windows API functions
  - Inline hooking through direct memory patching

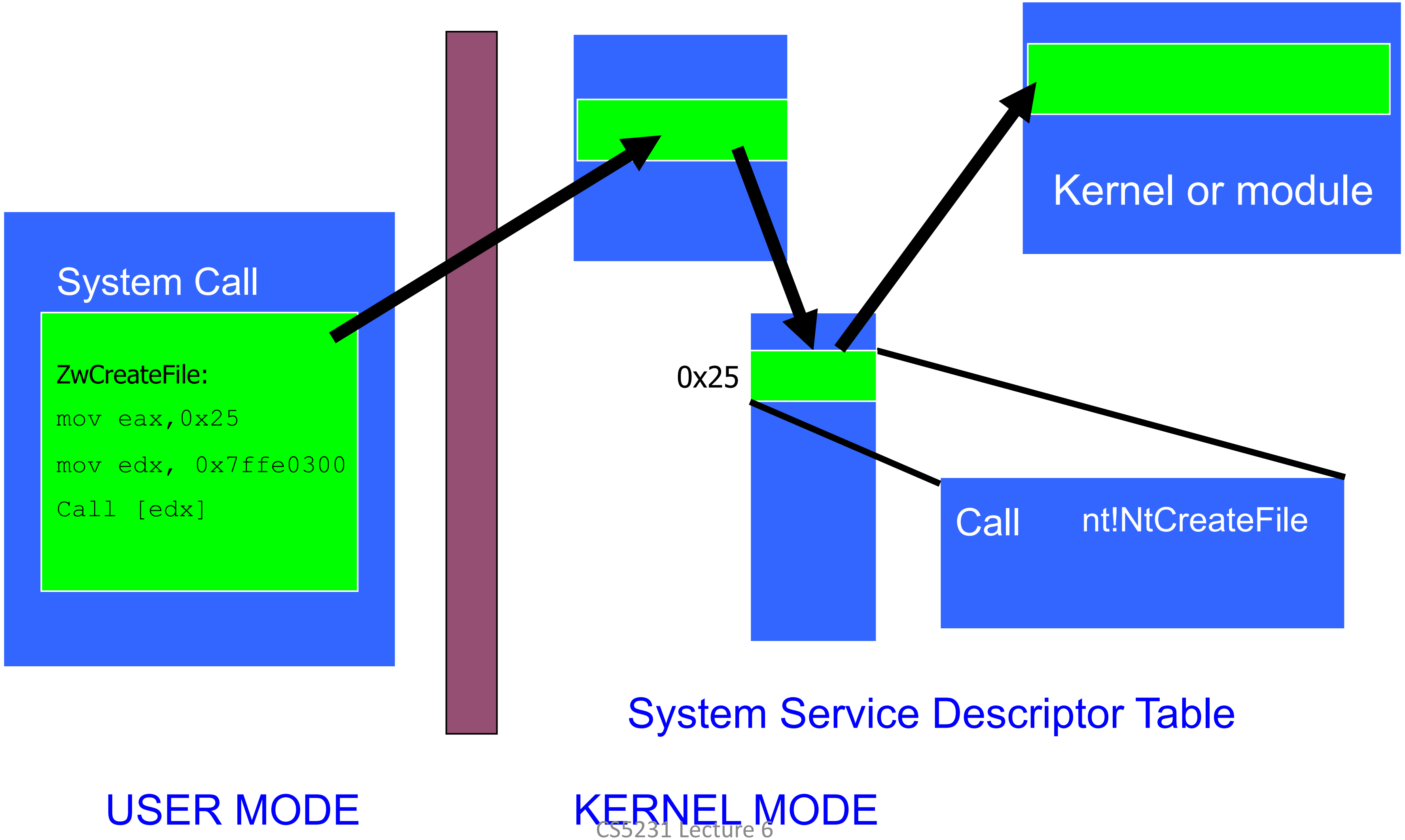
Hacker Defender – Hook Installation



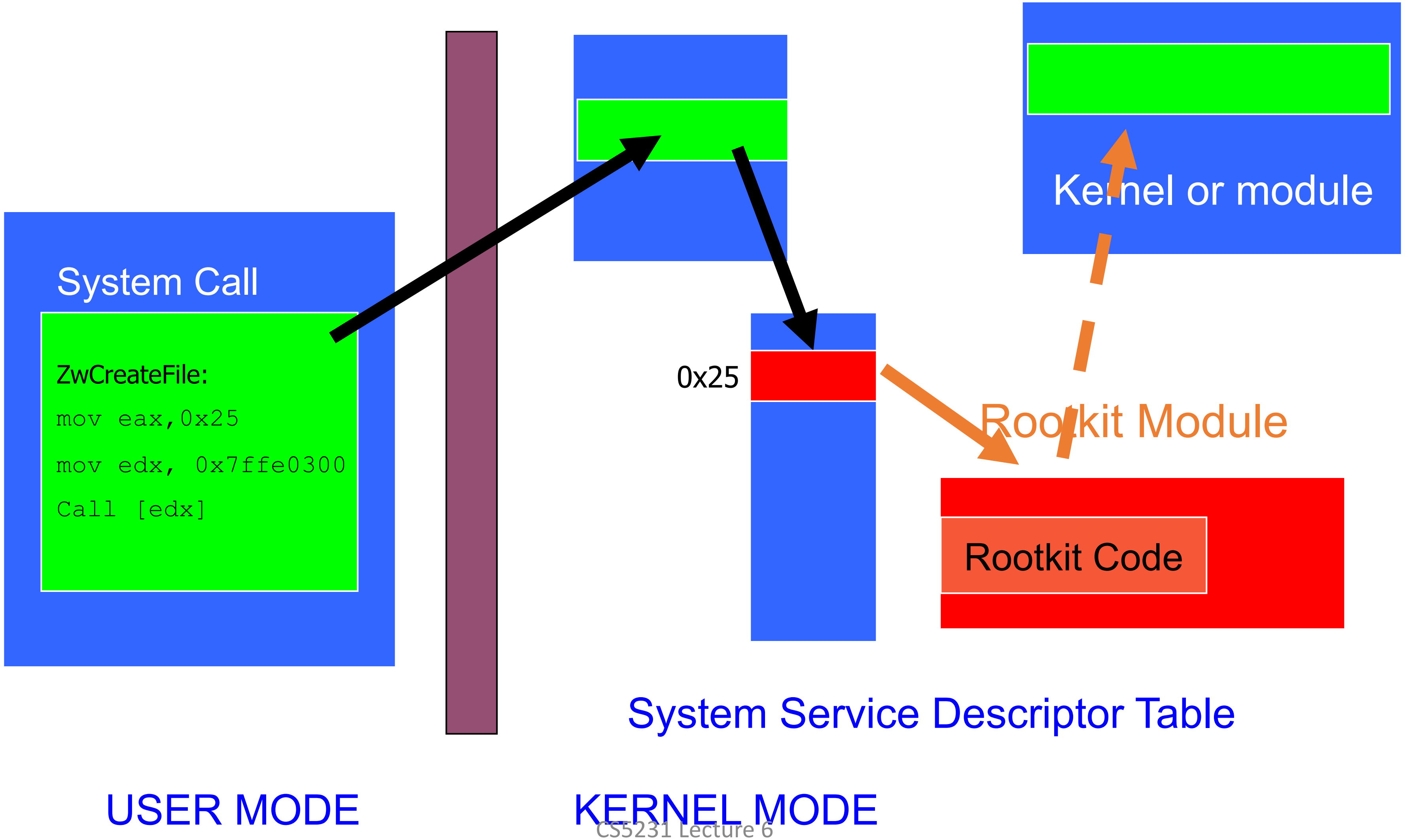
SSDT Hooking (1)



SSDT Hooking (2)



SSDT Hooking (3)



Places where can be subverted

File system:
- boot sectors
- file infections
- ASEPs (mostly registry keys)

BIOS flash, ?

persistent

CODE sections:
- processes
- kernel
- kernel drivers

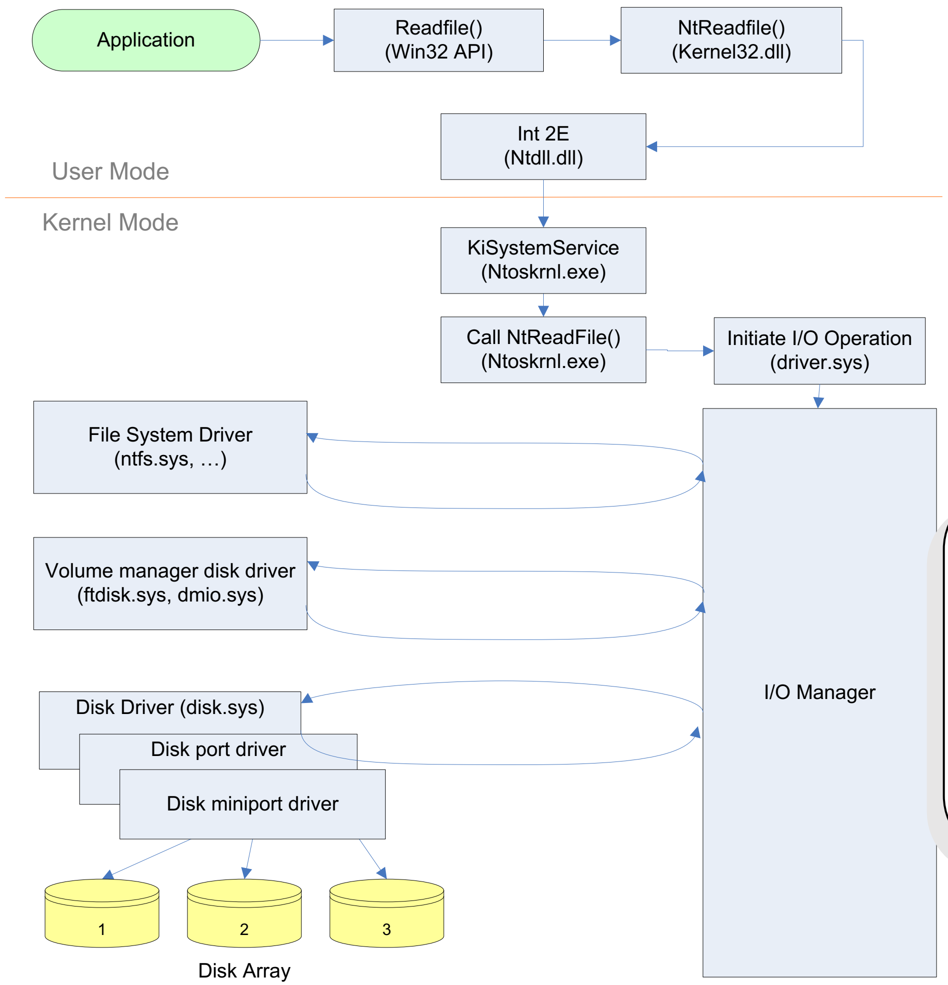
DATA sections:
- processes
- kernel
- kernel drivers

CPU registers: Debug Registers, Some MSRs, ?

volatile

Joanna Rutkowska, invisiblethings.org, 2006.

Discussion



Question I:
How to detect user-mode rootkits?

Question II:
How to detect kernel-mode rootkits?

Defense

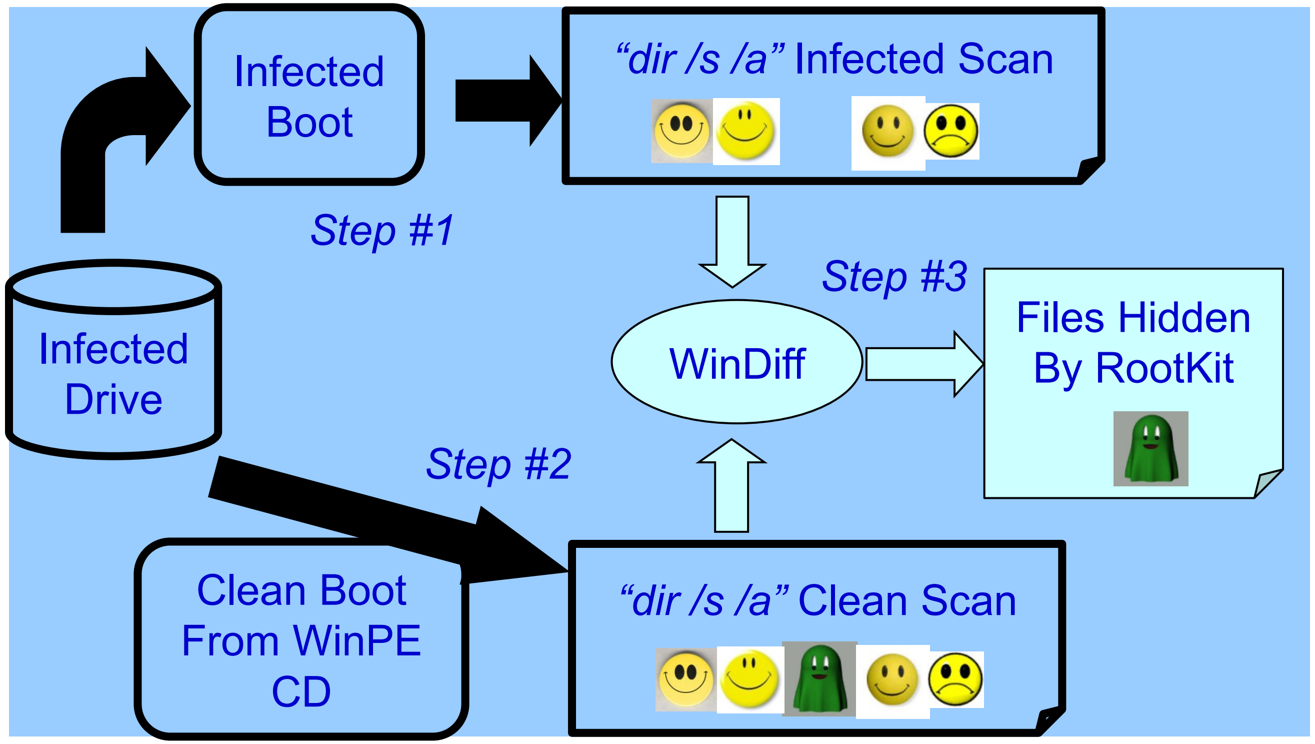
- Cross-View Detection
  - Microsoft Ghostbuster
  - Execution Path Analysis (EPA)
- Integrity Checking
  - CoPilot
  - Virtual Machine Introspection

Microsoft Ghostbuster

- **Motivation:** Rootkits cause & hide some persistent state changes
  - Rootkit-related files

- **Idea:** detecting persistent state changes
  - Compute a cryptographic has of every file on infected disk and match it against previous known database

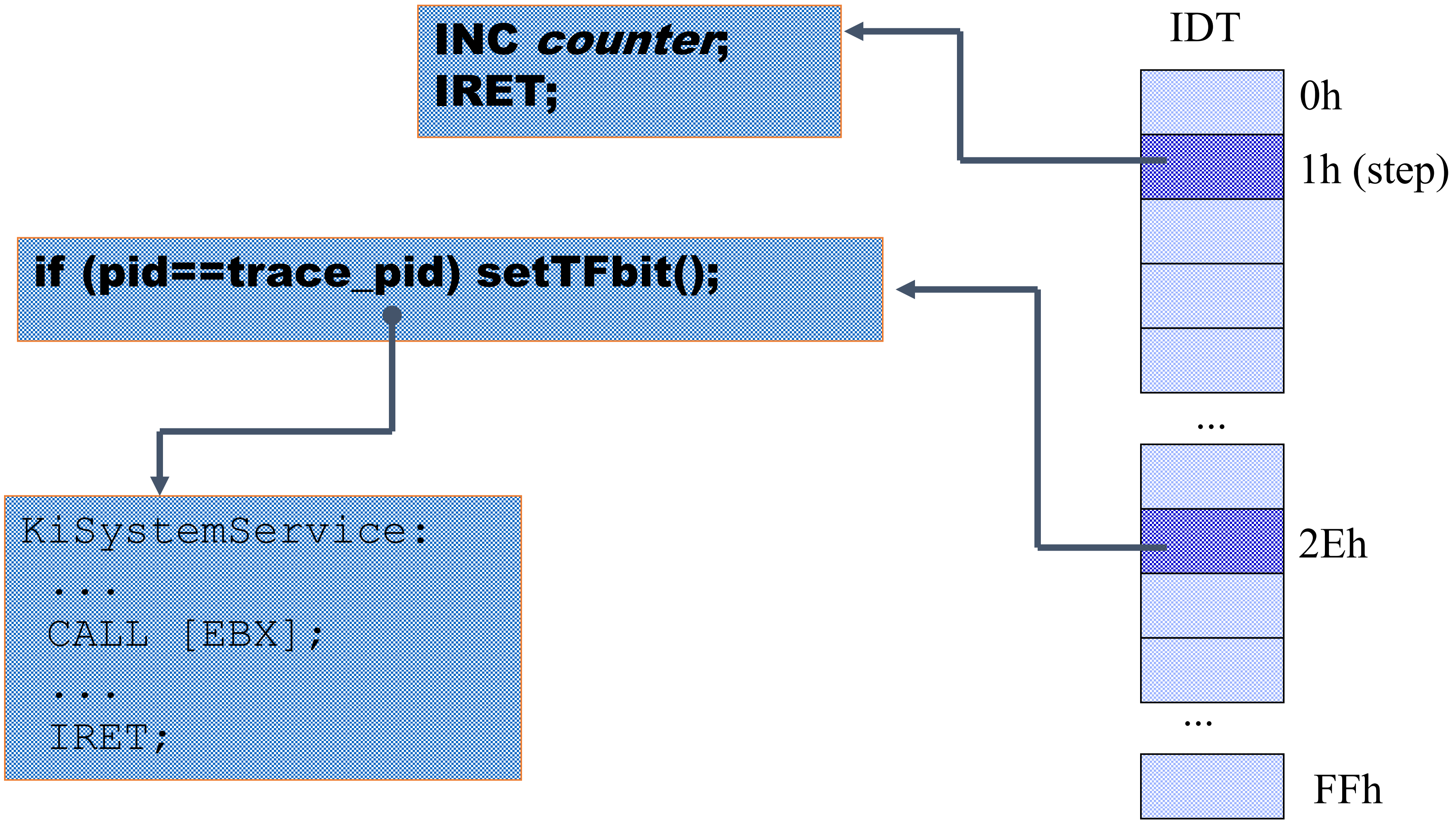
Microsoft Ghostbuster



Execution Path Analysis (EPA)

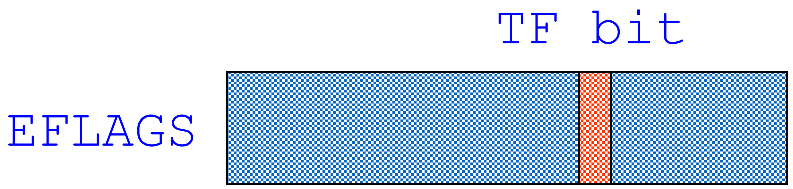
- **Motivation:** Rootkits cause execution path changes
  - Rootkit hooking (IAT/EAT, SSDT, IDT)
  - Raw code change (inline hooking)
- **Idea:** detecting execution path changes
  - Trace the execution path for some typical system activities (like system services)
  - Compare the trace with the trace saved after the installation of clear system
  - So, we need a baseline, but it is mostly acceptable requirement (exactly as in case of most integrity checkers).

EPA – implementation



Step Mode on IA-32

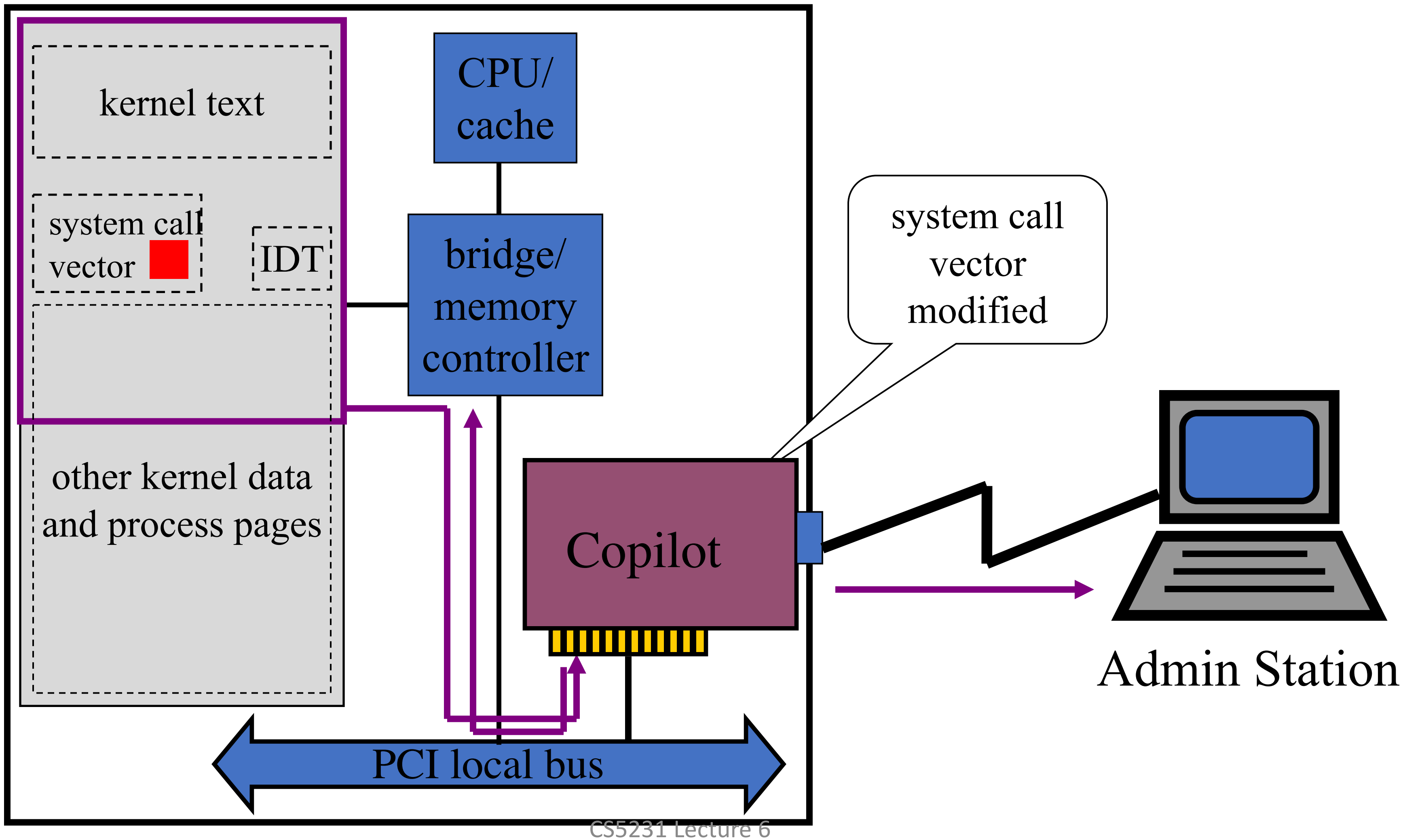
- TF bit in EFLAGS register
- When enabled, an exception is raised after every machine instruction
  - Exception hander is stored in IDT[1]
- TF bit is cleared when int 2eh instruction is executed to enter the kernel mode



CoPilot

- Remove reliance on system software correctness
  - Use hardware access to resources (e.g., memory)
  - Run protection code on a coprocessor (NOT the host)
  - Provide a secure reporting mechanism
- Basic model:
  - Collect data based on monitor's observables
  - Determine if data violates monitor policy
  - Take some action (e.g., report, recover, etc.)

Copilot Integrity Protection



Copilot Protection Strategy

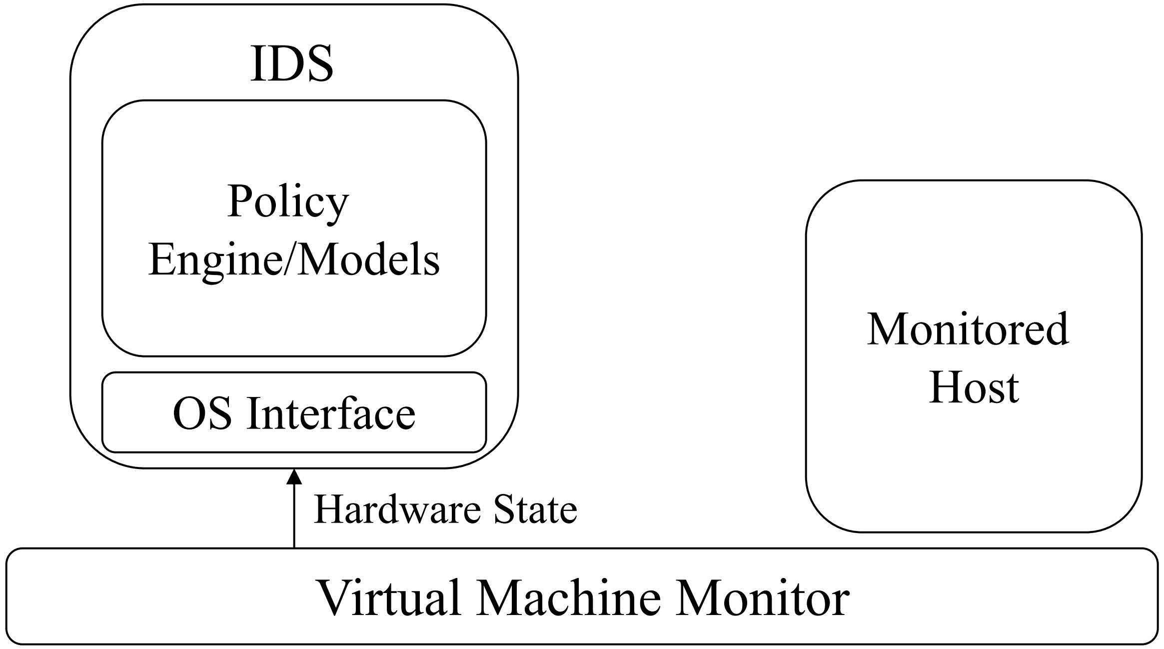
- Copilot currently uses the following traditional methods
  - Hash of Linux kernel text
  - Linux system call vector
  - Linux interrupt descriptor table
  - Linux module list/text
- Compare the above with a “known-good” state
- Copilot improves these methods by providing an isolated and independent platform for kernel monitoring

Virtual Machine Introspection (VMI)

- Remove reliance on system software correctness
  - Use VMI to access resources (e.g., memory)
  - Run protection code on VMM
  - Provide a secure reporting mechanism
- Basic model:
  - Run the monitored host in a sandbox (Virtual Machine) on some host
  - Run the IDS outside the VM
  - Allow the IDS to pause the VM and inspect the hardware state of host
  - Policy modules determine if the state is good or bad, and how to respond

Virtual Machine Introspection (VMI)

- VMM has access to ALL of the monitored host's virtual hardware
- VMM can pause guest OS to see a consistent state



Discussion

- Detection
  - Microsoft Ghostbuster
  - Execution Path Analysis (EPA)
  - CoPilot
  - Virtual Machine Introspection
- Prevention

Rootkit Commonalities

Question:
Any Commonalities?

New code
Persistent code
Called on demand.

adore-ng

- Linux 2.4/2.6
- Kernel module
- Adds "custom" functions
- Hooks VFS

SucKIT

- Linux 2.4
- /dev/kmem
- Adds "custom" functions
- Hooks system calls

FU

- Windows 2k/XP
- Device Driver
- Modifies kernel objects
- Installs custom driver code

NICKLE

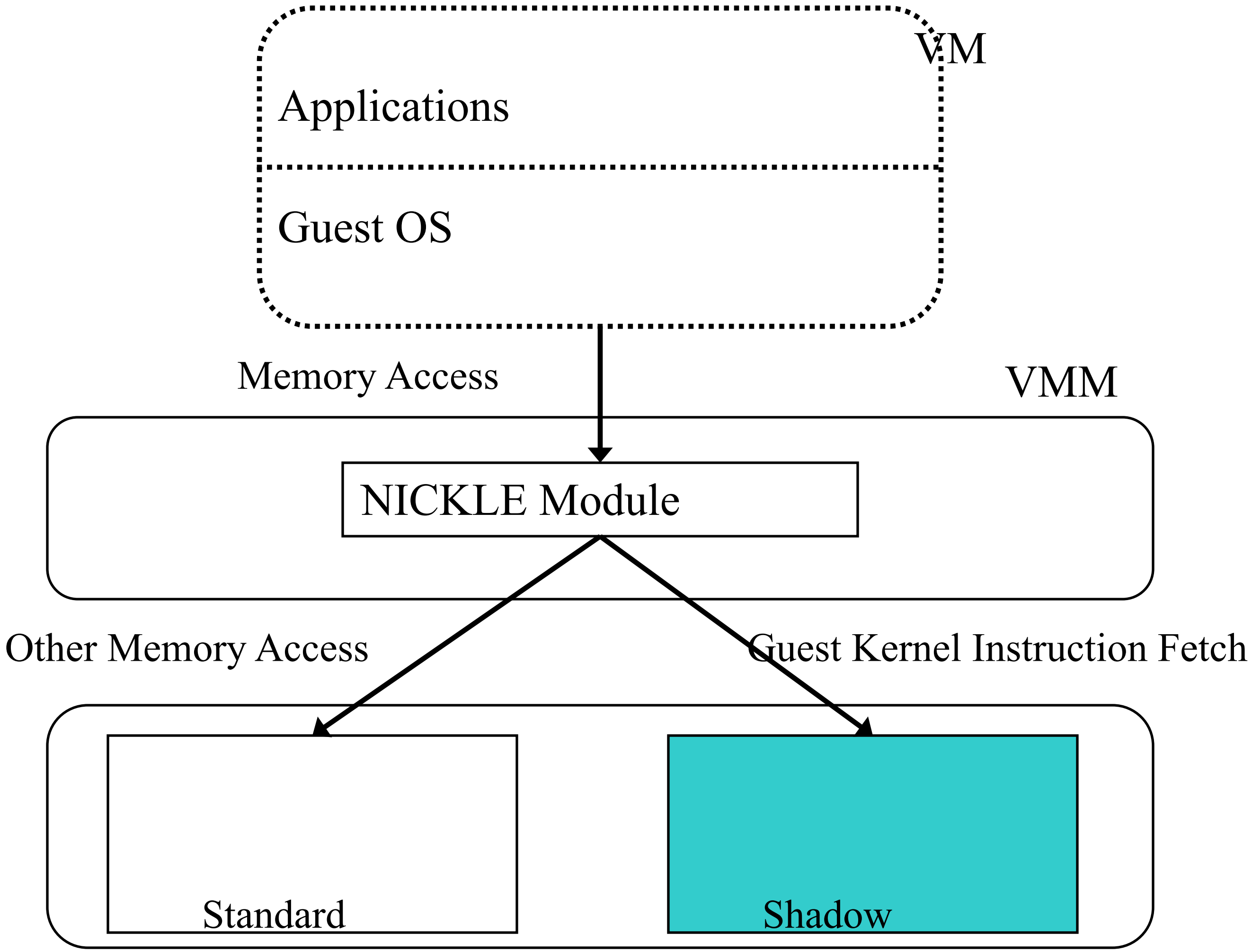
- Kernel Code Integrity:
  - Tracking run-time kernel code layout
  - Enforcing the following properties
    - Only loading authenticated kernel code
    - Only executing authenticated kernel code

NICKLE

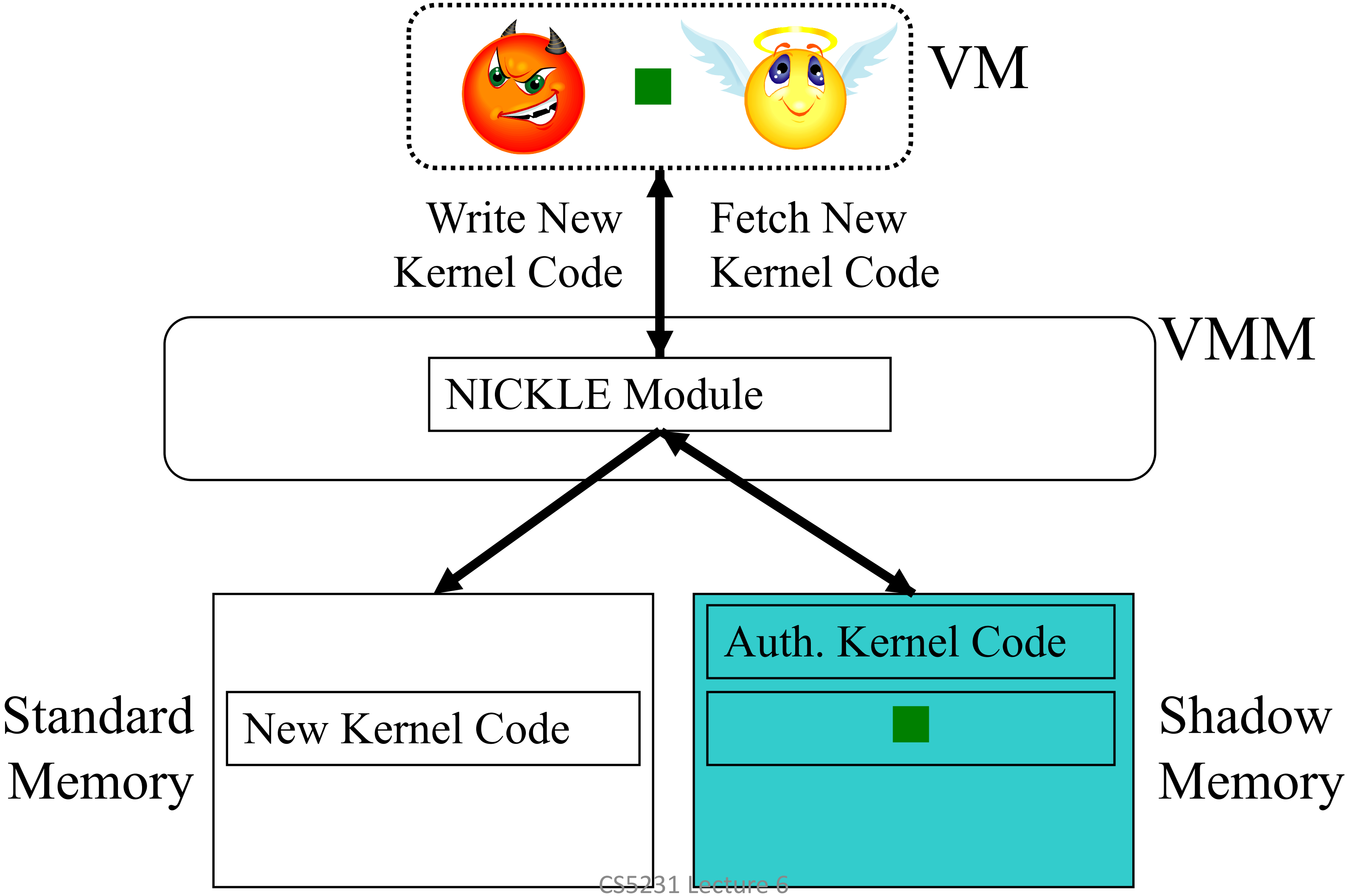
- Two memory spaces
  - Shadow: Authenticated kernel code
  - Standard: Everything else
- Use a VMM to manage the two
- Dynamically reroute memory accesses

- Side note: Unmodified OS

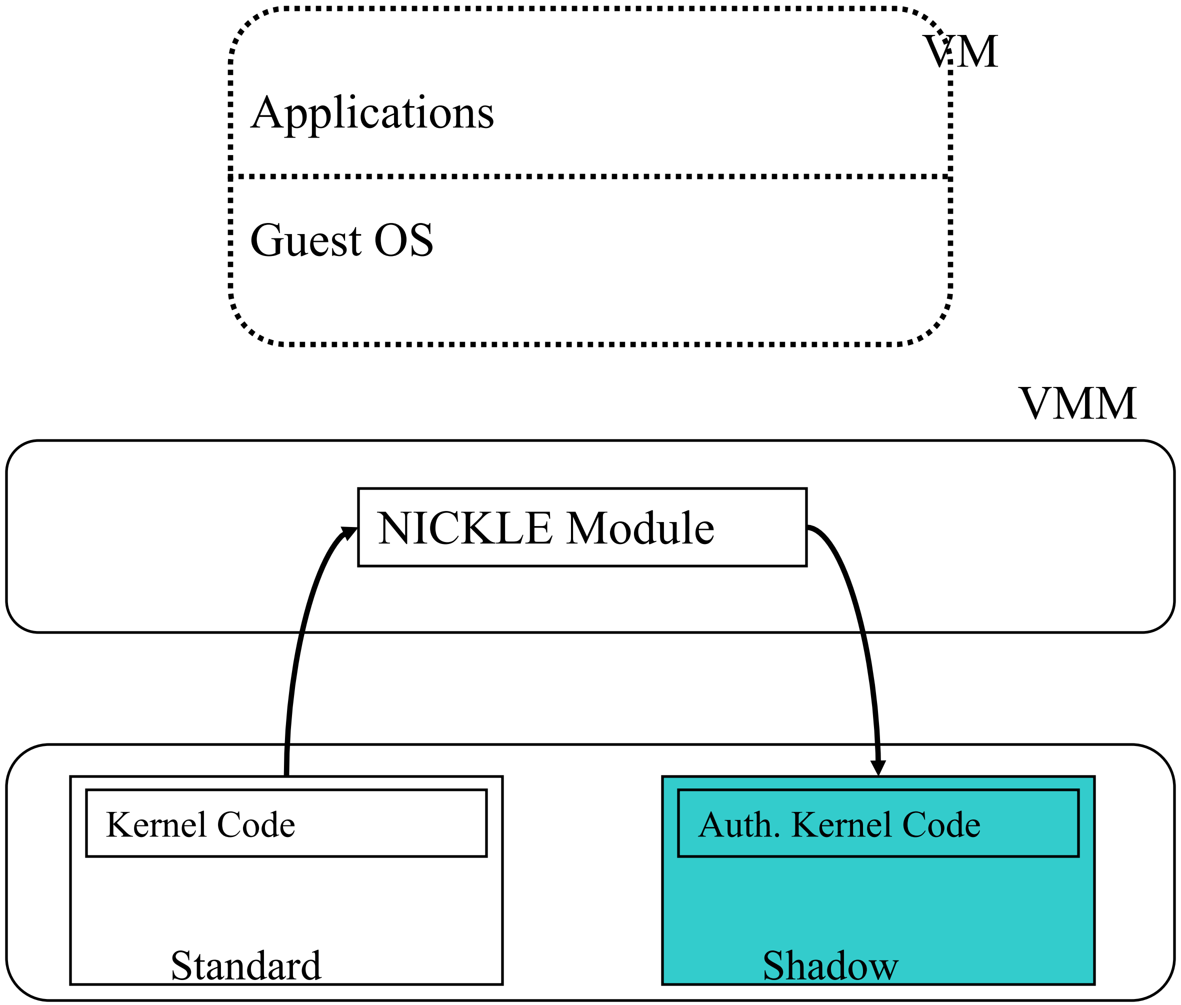
Memory mirroring



Attack scenario



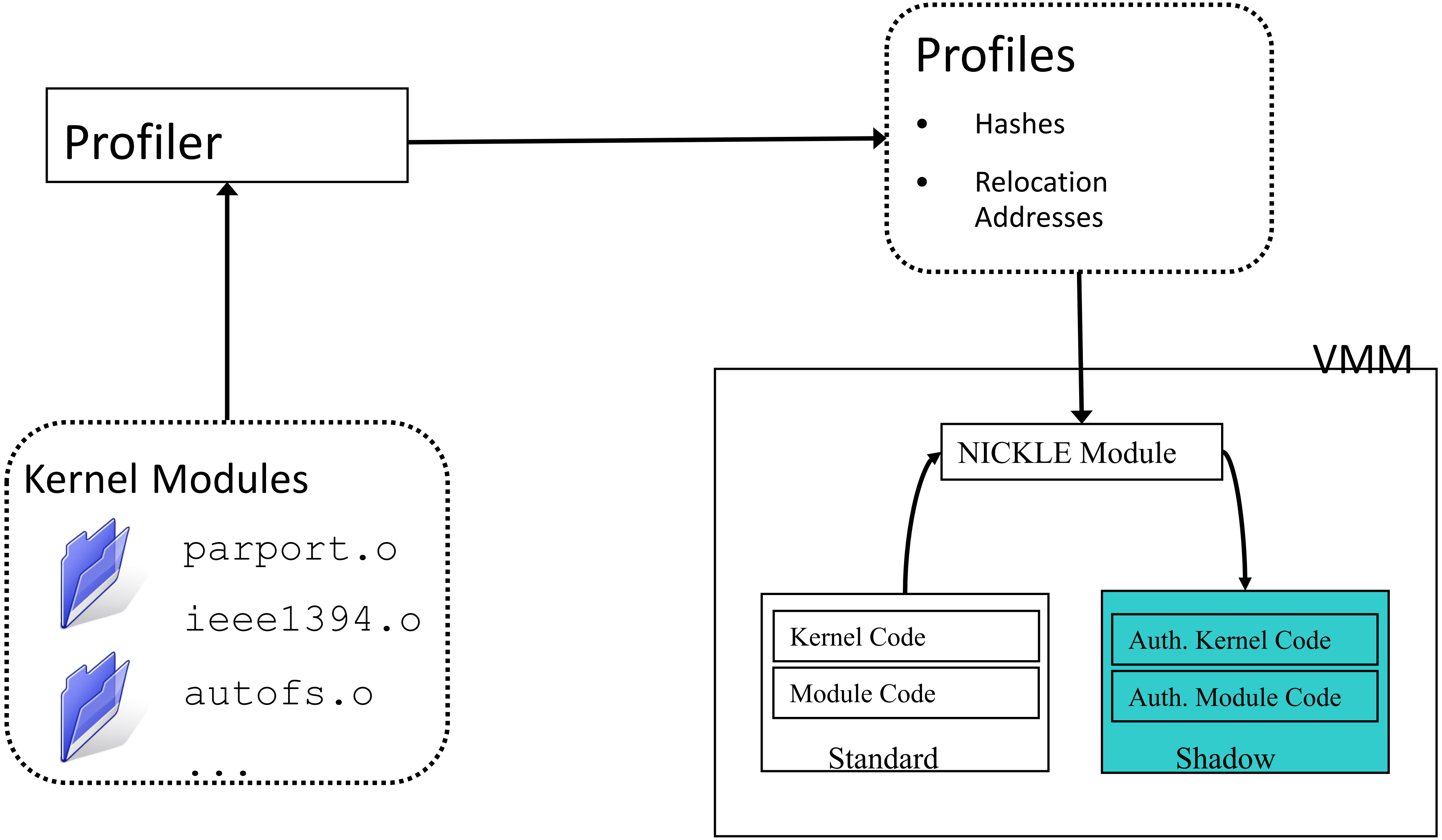
Filling the shadow memory



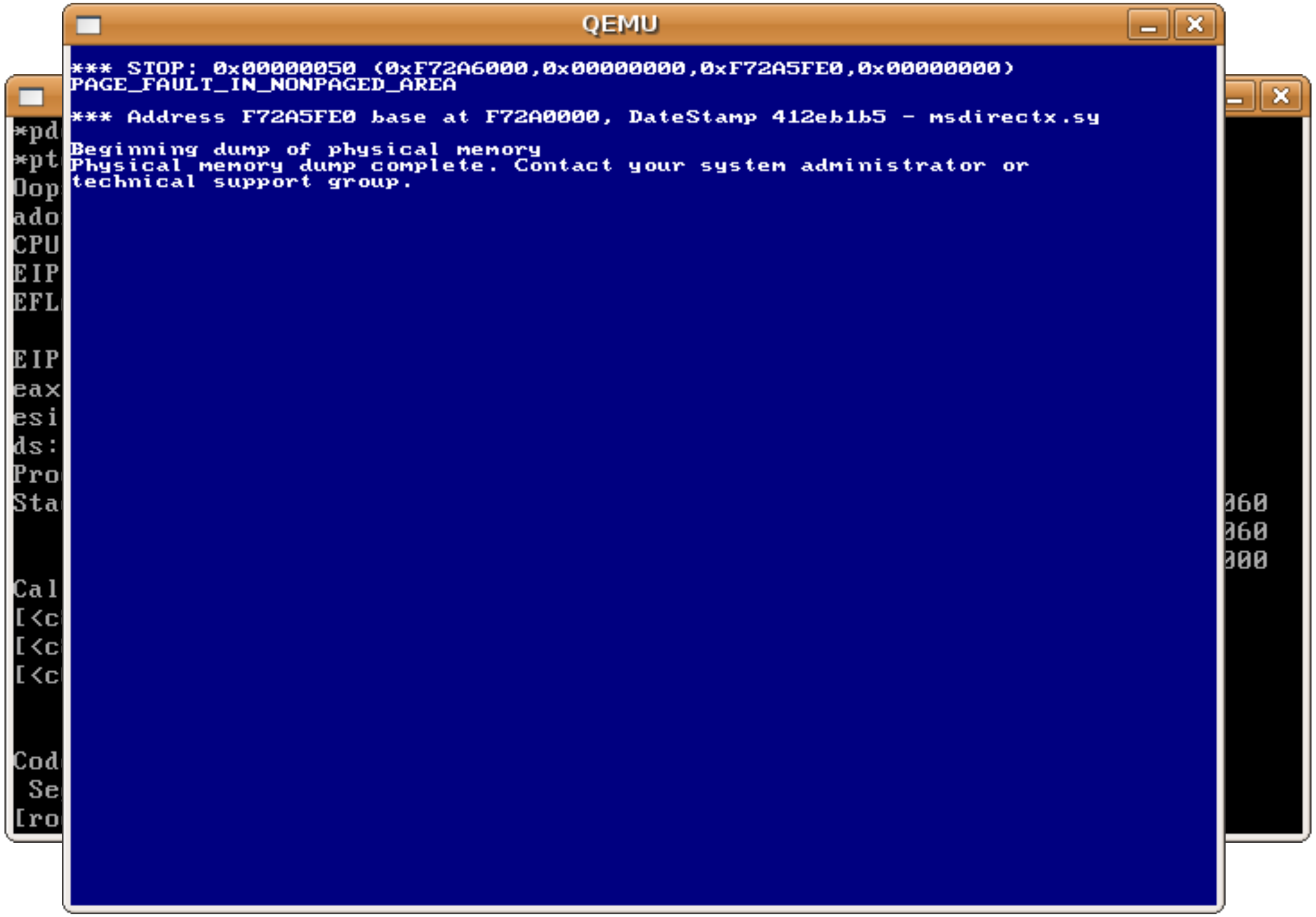
Not so fast… kernel modules!

- Technique:
  - Detect when module has been loaded
  - Hash and verify module
  - Copy to shadow memory
- Pain:
  - Dynamically relocatable
  - Lots of them

Kernel modules



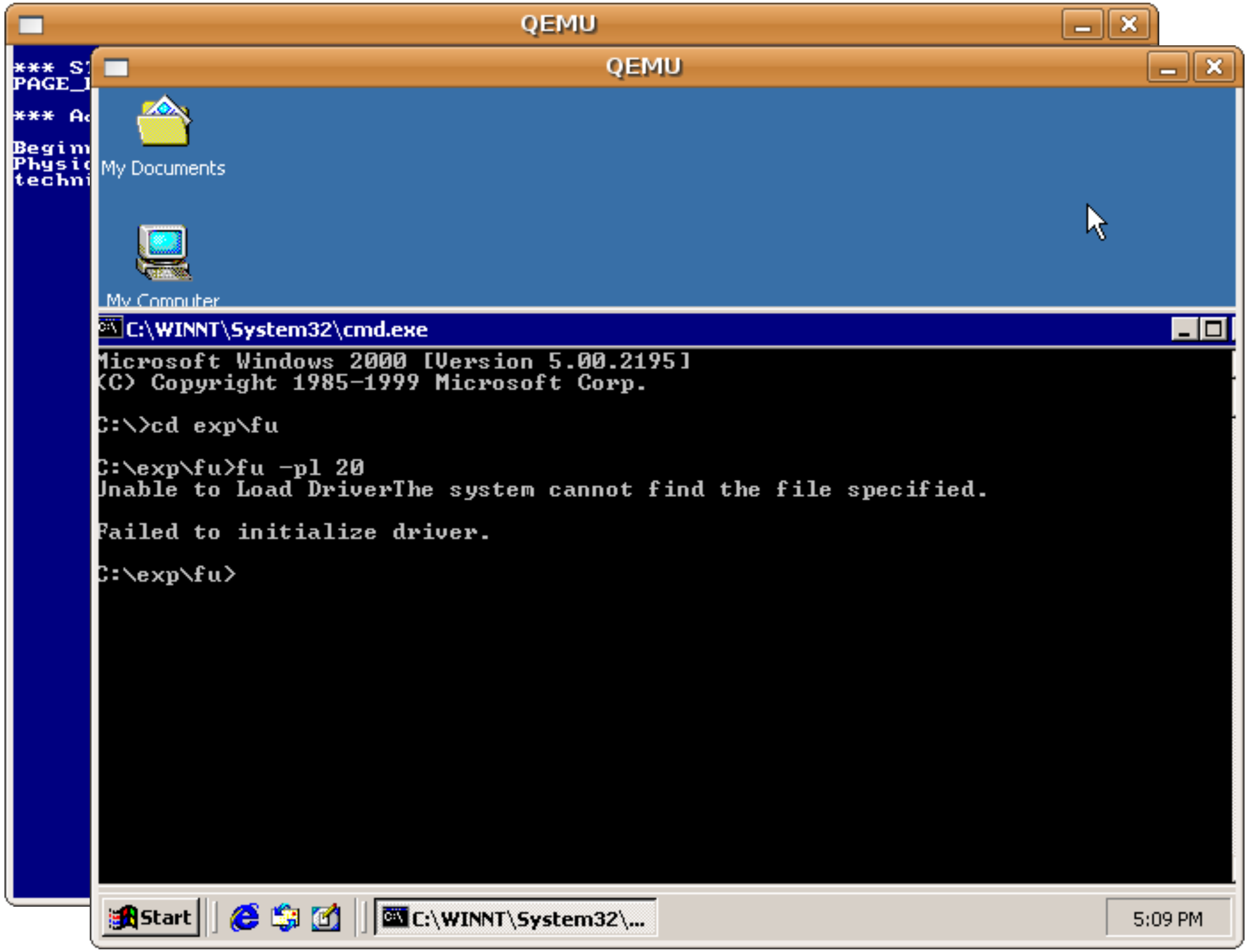
Improvements



Code rewriting

- Rewrite malicious code
- Rootkits we surveyed `call` malicious code during installation
- Given rootkit installation scenario…
  - Replace bad code with `return -1;`
  - `b8ffffffffc3`

Code rewriting



Demonstration of Effectiveness

Guest OS	Rootkit	Attack Vector	Outcome of NICKLE Response				
			Observe Mode	Rewrite Mode		Break Mode	
			Detected?	Prevented?	Outcome	Prevented?	Outcome
Linux 2.4	adore 0.42, 0.53	LKM	✓	✓	insmod fails	✓	Seg. fault
	adore-ng 0.56	LKM	✓	✓	insmod fails	✓	Seg. fault
	knark	LKM	✓	✓	insmod fails	✓	Seg. fault
	rkit 1.01	LKM	✓	✓	insmod fails	✓	Seg. fault
	kbdv3	LKM	✓	✓	insmod fails	✓	Seg. fault
	allroot	LKM	✓	✓	insmod fails	✓	Seg. fault
	rial	LKM	✓	✓	insmod fails	✓	Seg. fault
	Phantasmagoria	LKM	✓	✓	insmod fails	✓	Seg. fault
	SucKIT 1.3b	/dev/kmem	✓	✓	Installation fails silently	✓	Seg. fault
Linux 2.6	adore-ng 0.56	LKM	✓	✓	insmod fails	✓	Seg. fault
	eNYeLKM v1.2	LKM	✓	✓	insmod fails	✓	Seg. fault
	sk2rc2	/dev/kmem	✓	✓	Installation fails	✓	Seg. fault
	superkit	/dev/kmem	✓	✓	Installation fails	✓	Seg. fault
	mood-nt 2.3	/dev/kmem	✓	✓	Installation fails	✓	Seg. fault
	override	LKM	✓	✓	insmod fails	✓	Seg. fault
	Phalanx b6	/dev/mem	✓	✓	Installation crashes	✓	Seg. fault
Windows 2K/XP	FU	DKOM†	✓	✓	Driver loading fails	✓	BSOD§
	FUTo	DKOM	✓	✓	Driver loading fails	✓	BSOD
	he4hook 215b6	Driver	✓	✓	Driver loading fails	✓	BSOD
	hxdef 1.0.0 revisited	Driver	✓	partial‡	Driver loading fails	✓	BSOD
	hkdoor11	Driver	✓	✓	Driver loading fails	✓	BSOD
	yyt_hac	Driver	✓	✓	Driver loading fails	✓	BSOD
	NT Rootkit	Driver	✓	✓	Driver loading fails	✓	BSOD

Successfully preventing 23 real-world kernel rootkits!

Limitations

- Data-only attacks
  - Process hiding, privilege escalation, etc.
- Return-to-kernel
- Self-modifying code
- SMM, VMM rootkits