Rootkits

Rootkits are trojan backdoor tools that modify existing operating system software so that an attacker can keep access to and hide on a machine.

Kernel-mode rootkits

Modifications to the operating system kernel that hide the attacker and/or provide backdoor access

Kernel mode

Rootkits typically try to run at Ring 0

Intel x86 supports 4 rings

- Ring 0 = highest-privilege (kernel code)
  - Access to all memory
  - Access to special processor instructions/registers that directly alter CPU behavior
- Ring 3 = lowest-privilege (user programs)
- Windows/Linux do not use Ring 1 and 2 typically
- Administrator programs running in Ring 3 will need to get Ring 0 privileges from the kernel to perform operations

Kernel mode rootkit targets

Process and thread control

- Process hiding
- Execution redirection

Interprocess and network communication control

Memory control

Filesystem control

- File and directory hiding

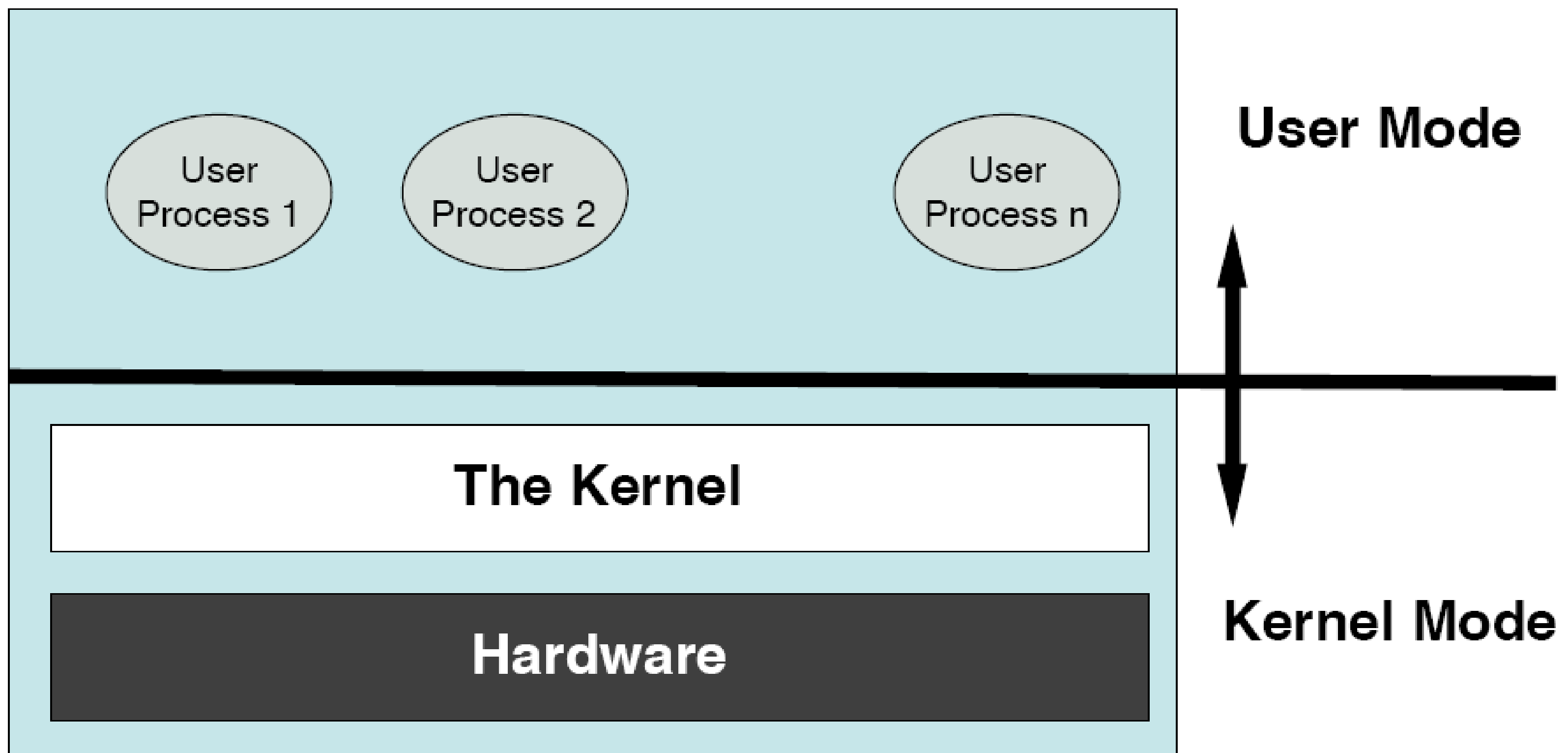
Operating system data

- Network port hiding
- Promiscuous mode hiding

Interrupts and hardware control

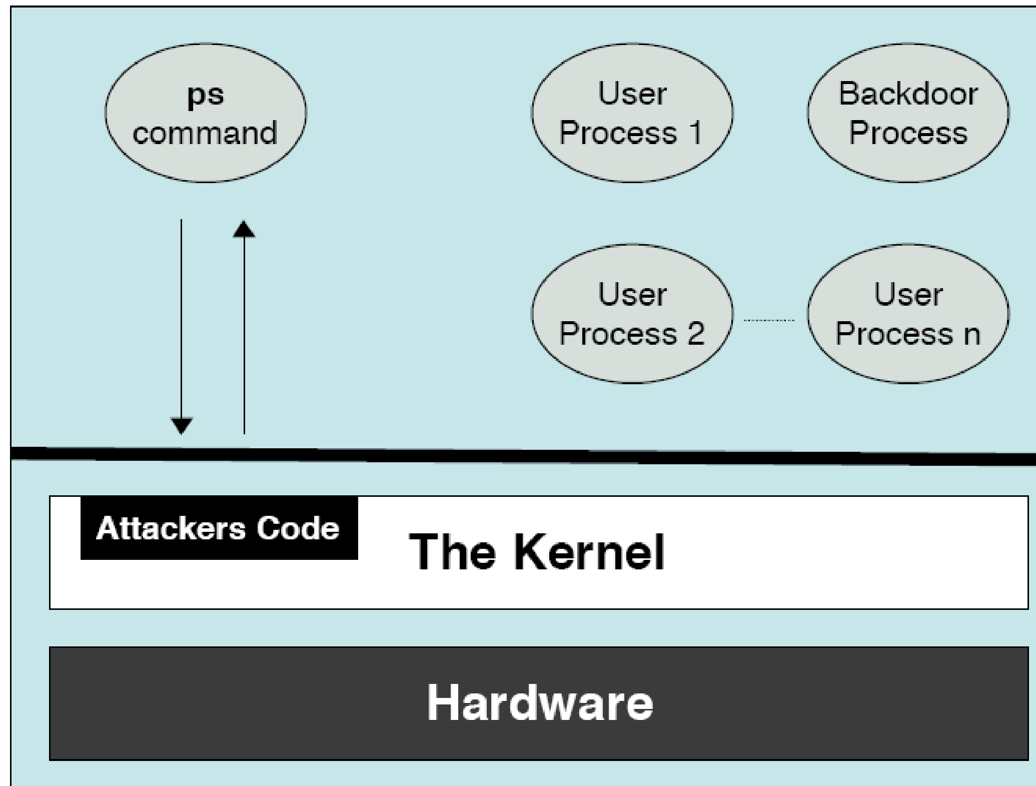
Review of kernels

View of Operating System



Kernel Manipulation

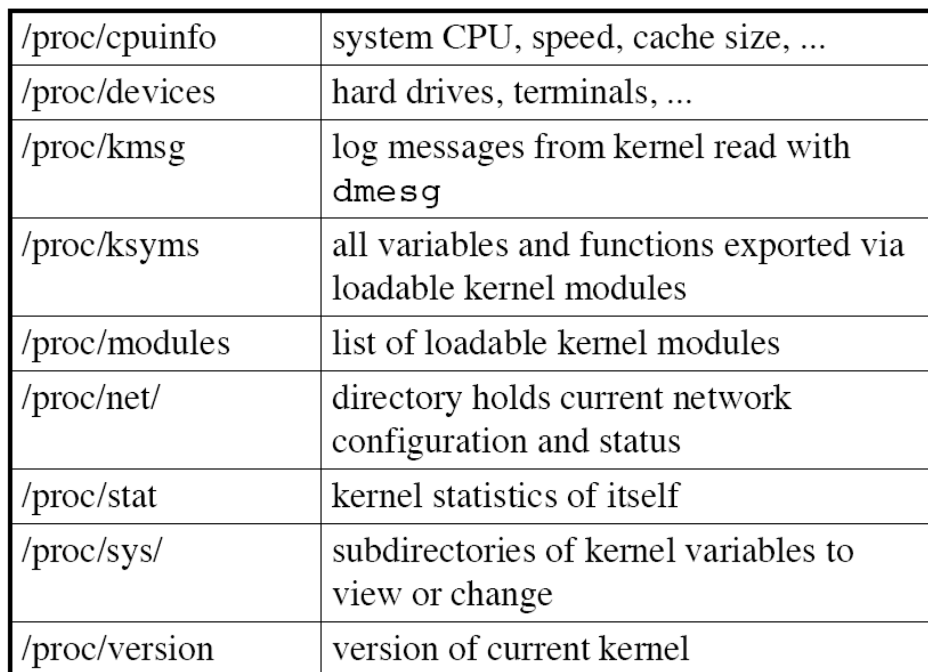
- **Malicious View**



Access via /proc

Mostly read only.

- /proc abstraction for easy access to kernel parameters



/proc/cpuinfo	system CPU, speed, cache size, ...
/proc/devices	hard drives, terminals, ...
/proc/kmsg	log messages from kernel read with `dmesg`
/proc/ksyms	all variables and functions exported via loadable kernel modules
/proc/modules	list of loadable kernel modules
/proc/net/	directory holds current network configuration and status
/proc/stat	kernel statistics of itself
/proc/sys/	subdirectories of kernel variables to view or change
/proc/version	version of current kernel

- `/proc/[pid]`
  - Per-process information

Writing to /proc

May alter kernel configuration in real time via /proc abstraction

```
root@morbo[1012]# cd /proc/sys/net/ipv4
root@morbo[1012]# cat ip_forward
0
root@morbo[1012]# echo 1  >ip_forward
root@morbo[1012]# cat ip_forward
1
```

System Calls

- Transition from user mode to kernel mode
- User-mode processes don’t access system calls directly
  - Done via libraries
- System call table is essential for invoking the right call
  - /usr/include/sys/syscall.h
  - /usr/include/bits/syscall
  - /usr/include/asm/unistd.h
- Linux System.map contains kernel symbols

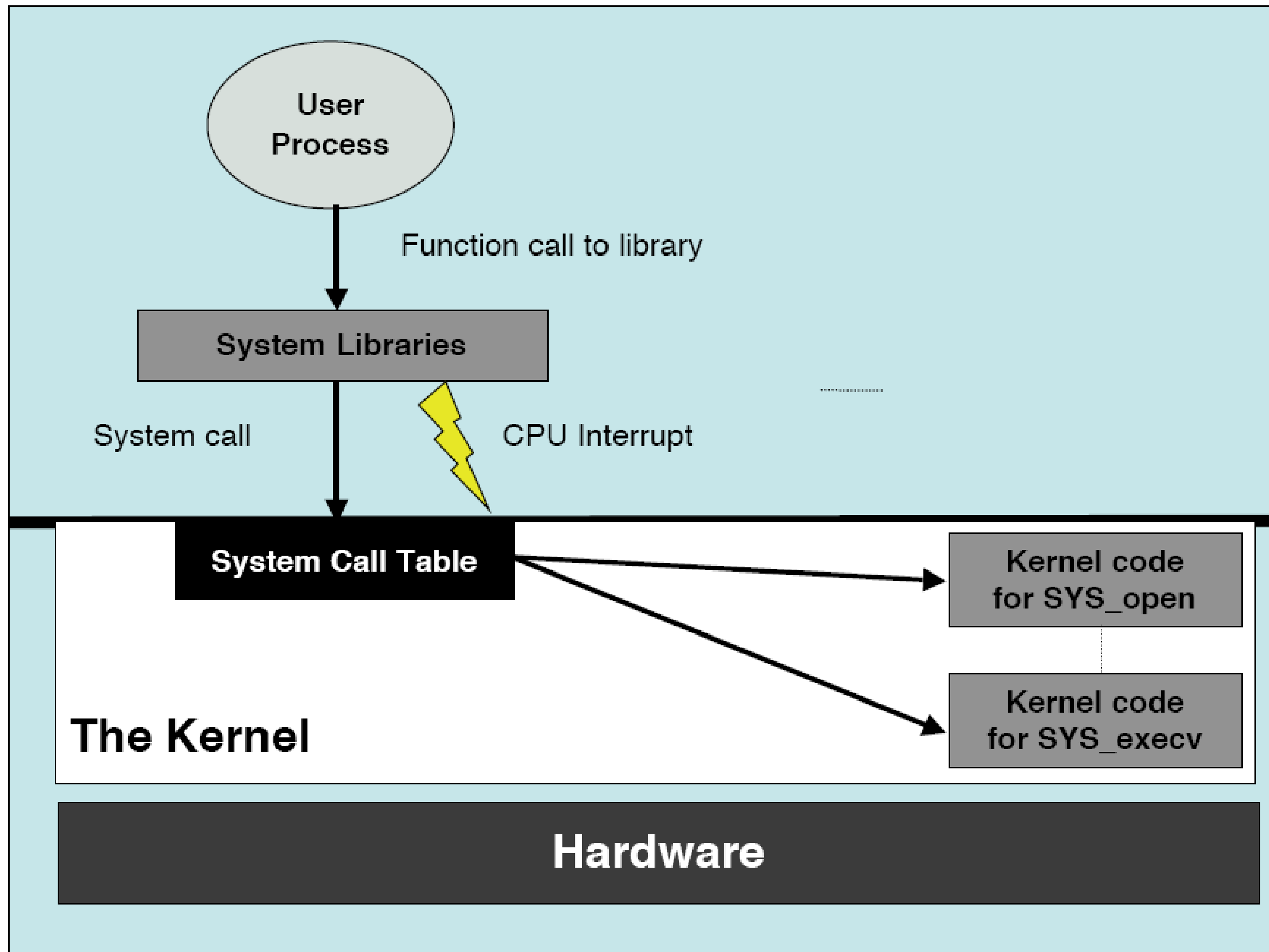
System Call Table

SYS_open	Opens a file
SYS_read	Reads a file from the file system
SYS_write	Writes to the file system
SYS_execve	Executes a program
SYS_setuid	Sets the permissions of a running program
SYS_get_kernel_syms	Accesses the system table
SYS_query_module	Helps insert a loadable kernel module into the kernel

Symbol table

```
root@morbo[1021]# cat /boot/System.map
               *snip*
c0107510 T copy_thread
c01075c0 T dump_thread
c01076e0 T __switch_to
c0107820 T sys_clone
c0107860 T sys_vfork
c0107890 T sys_execve
c0107910 T get_wchan
c0107990 t __constant_memcpy
c0107aa0 t __constant_c_and_count_memset
               *snip*
```

System Call In Action



Watching User Syscalls

strace (Linux)

ktrace (OSX)

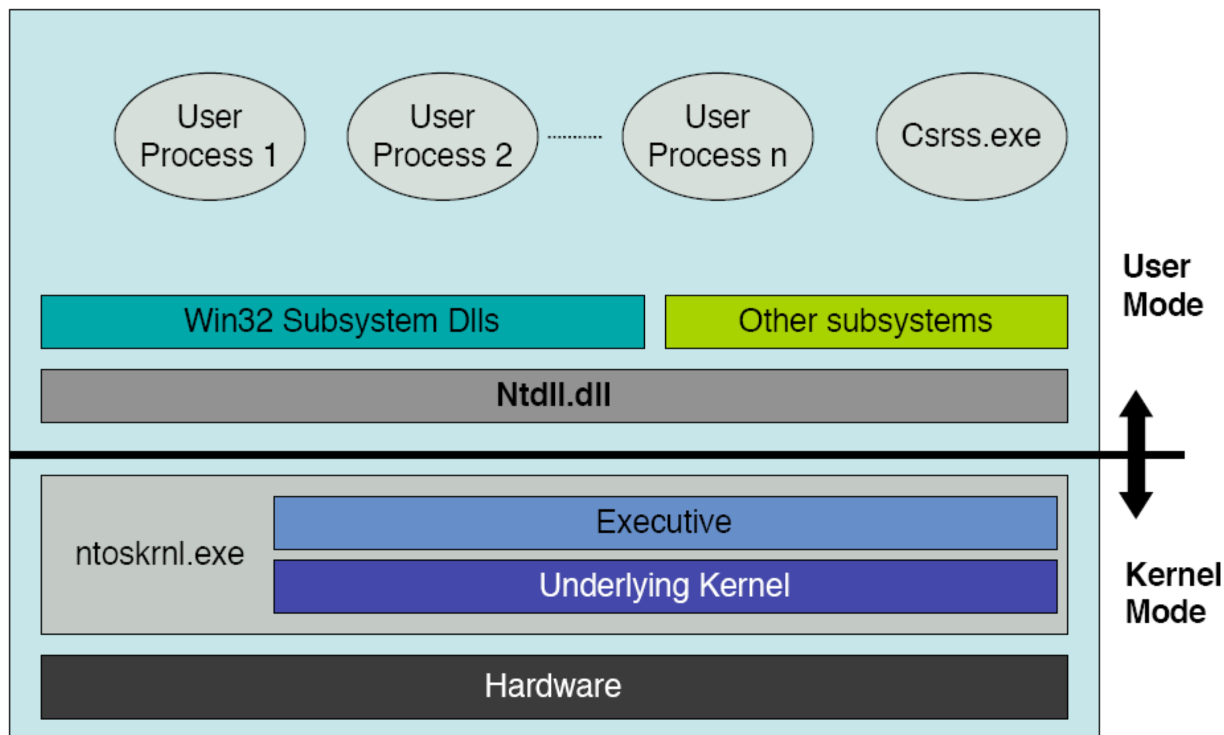
truss (Solaris)

```
root@morbo[1021]# strace cat /etc/passwd
execve("/bin/cat", ["cat", "/etc/passwd"], [/* 31 vars */]) = 0
uname({sys="Linux", node="morbo.cs.pdx.edu", ...}) = 0
brk(0)                                    = 0x804c3a4
open("/etc/ld.so.preload", O_RDONLY)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)        = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=84457, ...}) = 0
old_mmap(NULL, 84457, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40014000

                          *snip*

open("/etc/passwd", O_RDONLY|O_LARGEFILE) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=2092, ...}) = 0
read(3, "root:x:0:0:root:/root:/bin/bash\n"..., 4096) = 2092
write(1, "root:x:0:0:root:/root:/bin/bash\n"..., 2092root:x:0:0:root:/root:/bin

                          *snip*
```

Windows kernel

User-mode processes make calls to Win32 subsystem DLLs

- APIs for user programs to call the kernel
  - WIN32 Subsystem: csrss.exe, kernel32.dll, gdi32.dll, user32.dll
  - POSIX Subsystem: psxss.exe, psxdll.dll, posix.exe
  - Displaying information on screen, opening files, running programs
- Must go through ntdll.dll to get to kernel



Windows kernel

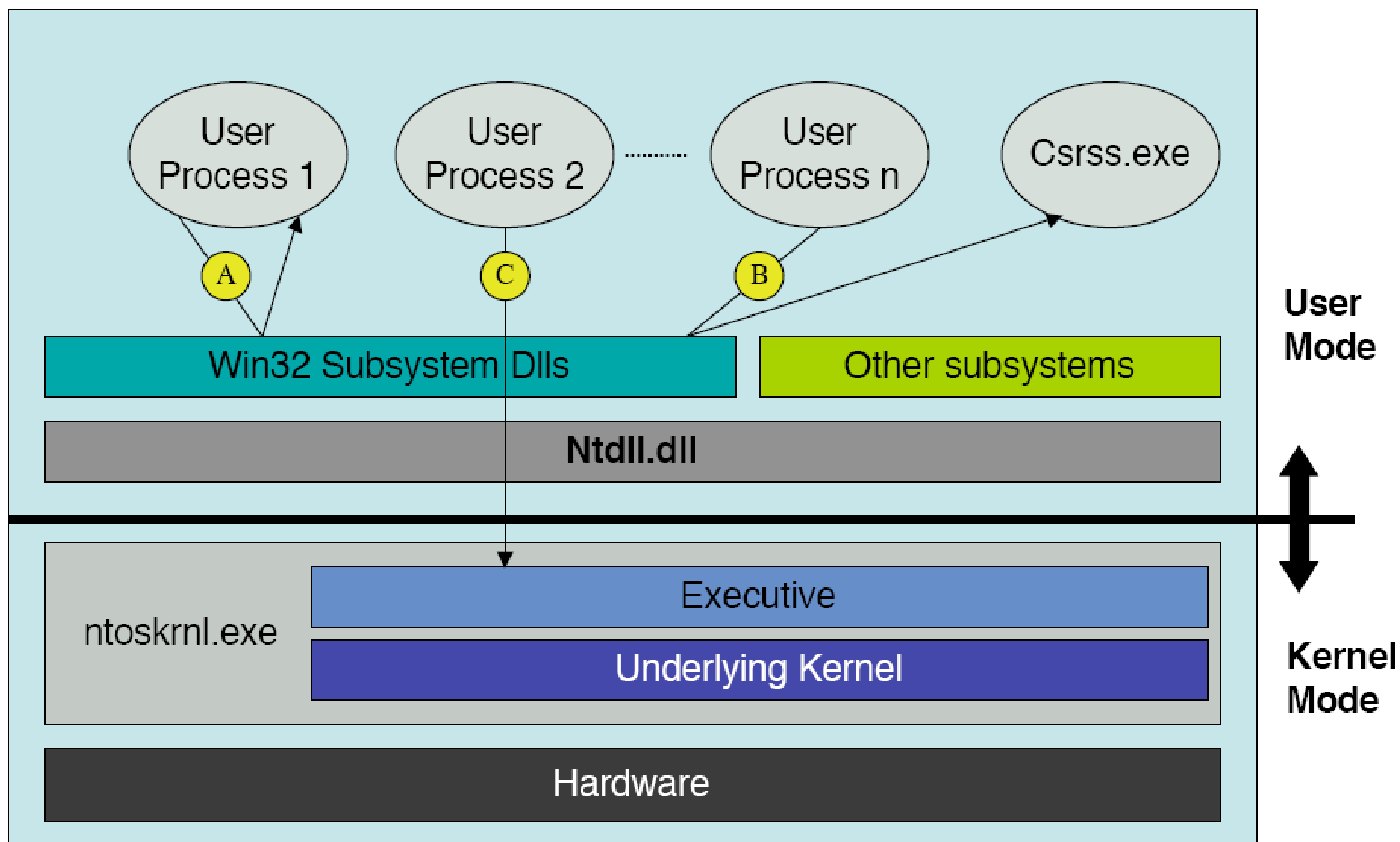
NTdll.dll

- Maps function calls for transition to the kernel.
- System Service Dispatching (system call)
  - Interrupt 0x2E on x86 processors.
  - Registers (call numbers) indicate location in dispatch table (SSDT).
- Invokes kernel-level functionality called the Executive within NTOSKRNL.EXE

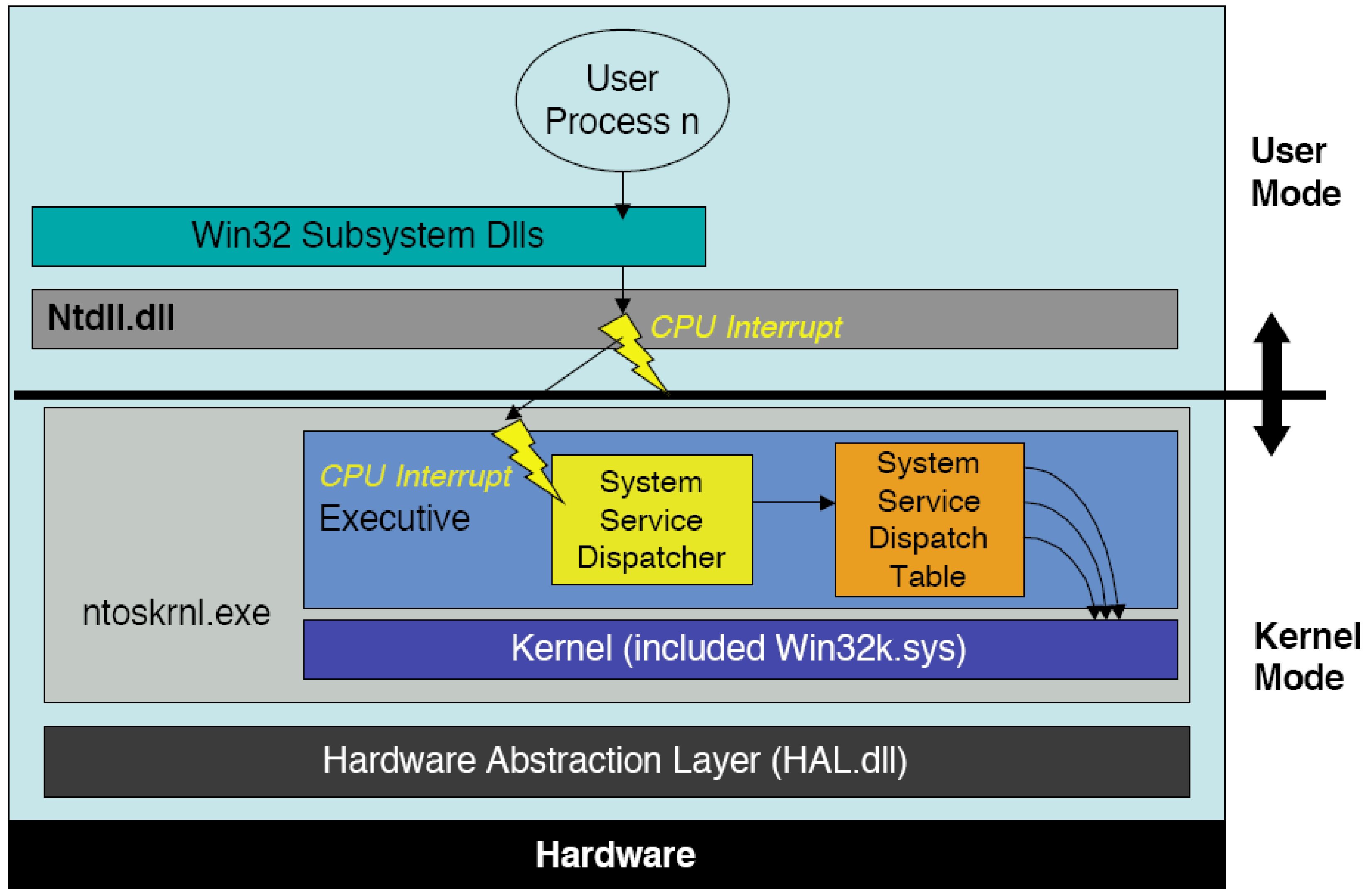
NTOSKRNL.EXE

- Executive
  - Makes kernel function calls available to user-mode.
- Virtual Memory Manager (VMM)
- Security Reference Monitor (SRM)
- I/O Manager
- Object Manager
- Process and Thread Manager
- Kernel Services (Exception handling and runtime library)
- LPC Manager (Local Procedure Call)

Windows kernel



Windows kernel (system calls)



Manipulating the kernel

Kernel modules/drivers

Hooking kernel functions and data

Direct kernel object modification

Running “underneath” kernel

Patching the kernel image

Kernel modules/drivers

Kernel modules/drivers

Augment or replace kernel features (e.g. system calls)

- In Linux: insmod

Add device drivers

Linux kernel modules

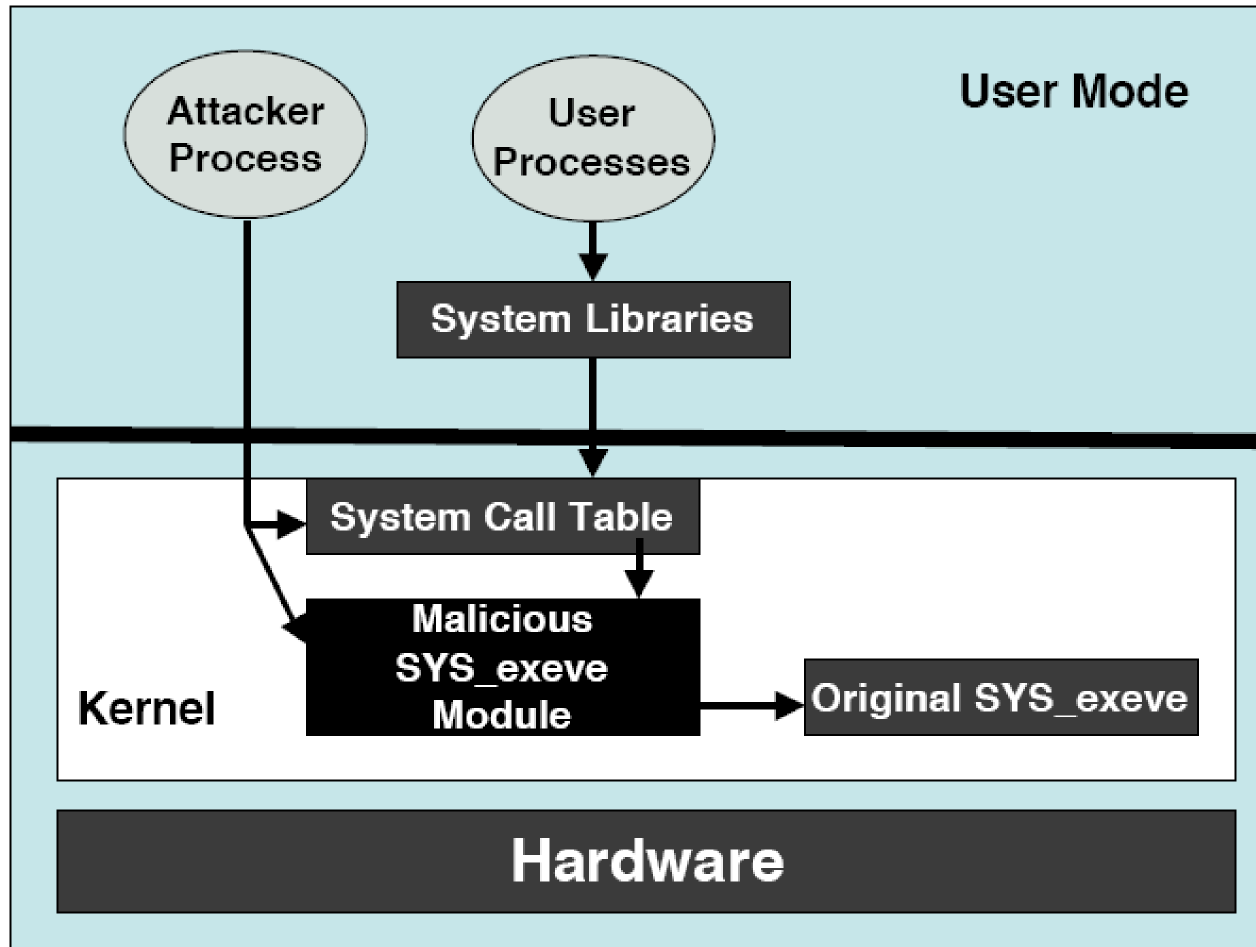
Allows administrator to dynamically include only code that is needed

- Modules run in ring 0, ideal for attacker

/proc/modules

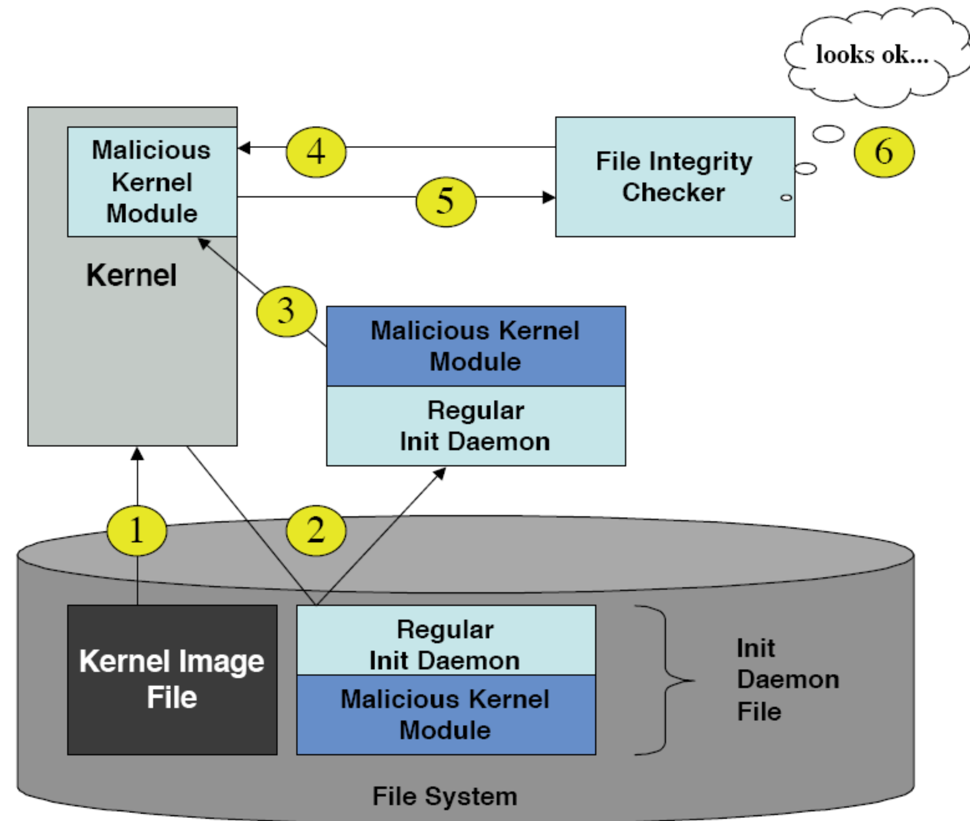
```
root@morbo[892]$ lsmod
Module                  Size  Used by    Not tainted
cdrom                  33728   0  (autoclean) [ide-cd]
soundcore               6500   0  (autoclean)
nfsd                   80272   0  (autoclean)
autofs                 13332   0  (autoclean) (unused)
iptable_filter          2412   0  (autoclean) (unused)
ip_tables              15096   1  [iptable_filter]
loop                   12184  15  (autoclean)
mousedev                5588   1
keybdev                 2976   0  (unused)
usbcore                78944   1  [hid usb-uhci]
eepro100               22740   1
ext3                   70912   4
raid5                  19080   2
```

Linux kernel modules



Example module: File redirections

1. Load kernel
2. Kernel starts init process
3. Modify init to load malicious kernel module into kernel
4. When process opens init files, malicious module captures request
5. Malicious module lies about contents so it is not revealed
6. File integrity checker thinks everything is fine



Loadable device drivers

Modules easy to detect

Kernel rootkits often load themselves as device drivers instead

- Runs in kernel mode with full access to privileged memory
- Examples: Slanret (27KB device driver), Krei (27KB TCP backdoor)

Can be used in conjunction with user-mode code

- Easier to debug and support functionality in user mode
- Registers a name for access from user space (i.e. \\Device\\MyDevice)
- Use ioctl/read/write to communicate to driver

Examples

Network drivers

- Network driver establishes communication with remote attacker while hiding it from applications above
- Network driver redirects or suppresses client traffic (e.g. Drop Windows Update DNS query)
- Can add as a filter in chain or via direct calling of NDIS

File system drivers

- Insertion within file system filter chains (see previous)

Can be made "stealthy"

- Inject interceptor functions in driver memory
  - NonPagedPool memory since driver memory should not be paged out
- Unload driver after injection so it can no longer be detected
- Kernel driver only loaded long enough to apply patch

Hooking kernel functions and data

Hooking within kernel

Can be done directly or via device drivers and kernel modules

More desirable than library hooks

- Places you on equal footing with detection software (Ring 0)
- Kernel memory 0x80000000 and above
- Kernel hooks provide global scope

Attractive targets

- Memory management
- Exception processing
- CPU registers that contain base addresses for many function tables

Memory management

Global and local descriptor tables

- Used to map addresses being used by process
- Code segment register (CS) points to where currently executing instruction is stored
- Changes via use of “far call” “ far jump” or “far return”
- Tampering with tables to hijack memory accesses

Memory management

Page directory tables

- Also instrumental for mapping addresses
- CPU uses special register CR3 to point to an array of 1024 32-bit values called the page directory
  - Each process has its own unique value of CR3 (its own page directory)
  - Threads of a process share CR3 value
  - Each 32-bit value specifies base address of a page table in physical memory
- CPU handles memory access via table lookups and checks
  - Descriptor check
    - » Check to see if segment being accessed has sufficient privilege
  - Page directory check
    - » Check to see if page table being accessed has sufficient privilege
  - Page check
    - » Check to see if page being accessed has sufficient privilege
- Rootkits tamper with tables to allow modification and hiding of code pages

Interrupt processing

Interrupt descriptor tables

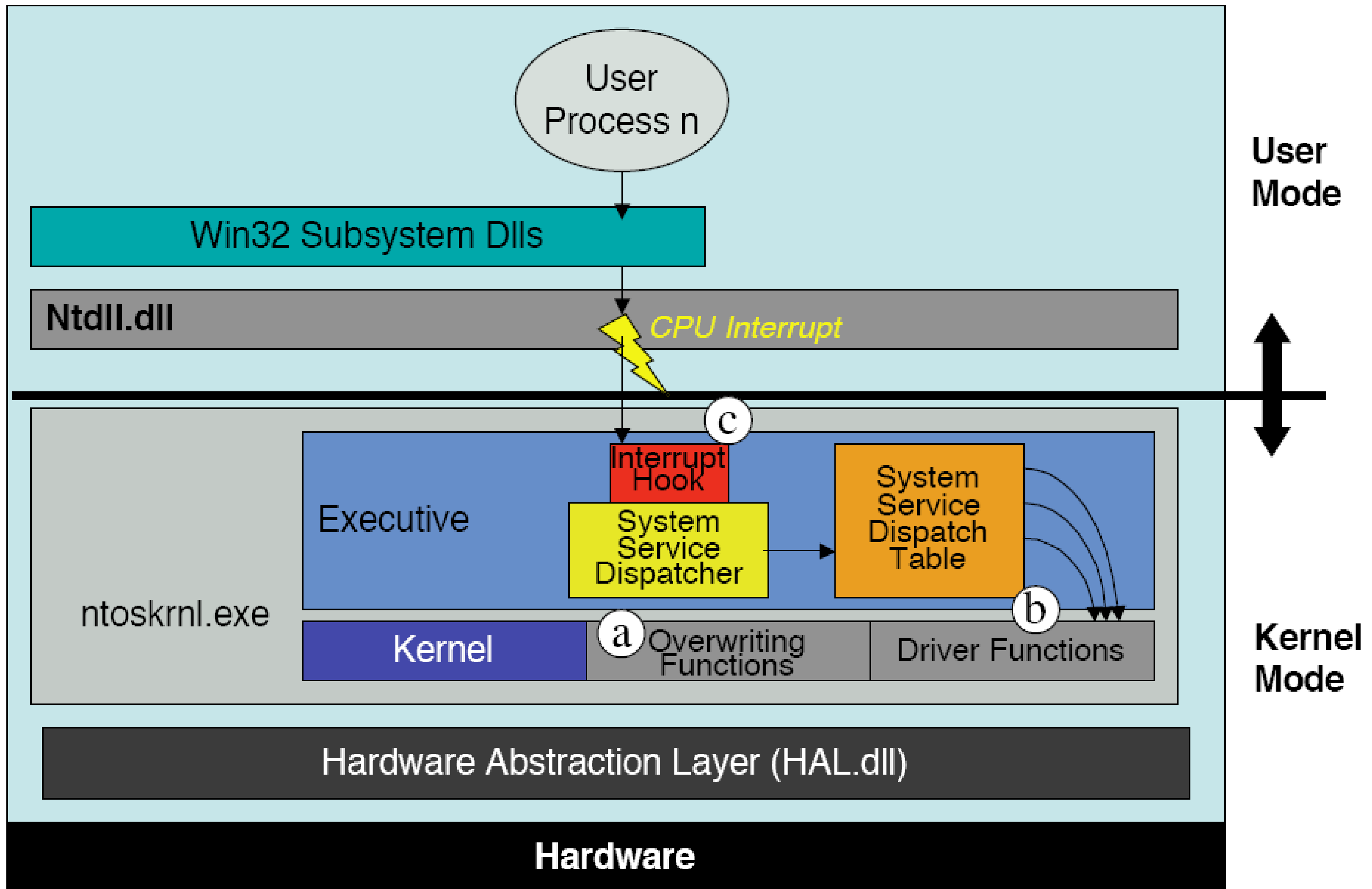
- Used to find interrupt handlers
- IDT register stores base of IDT
  - One IDTR per CPU
- IDT contains array of 256 entries (one for each interrupt)
  - IDT entry can specify privilege level to run at ("interrupt gate")
  - Useful in getting to kernel mode via interrupts (i.e. system calls generate interrupts)

Hooking the IDT

Interrupt Descriptor Table

- Numerous software and hardware interrupts
- Page faults (Entry 0x0e), timers, system calls (Entry 0x2e), etc.
- Hooking most useful on system call interrupts
  - i.e. int 2e
  - Store original int 2e function handler (KiSystemService) into global DWORD
  - Replace IDT entry for system call interrupts with address of your hook
  - Hook calls KiSystemService upon completion
- Modern Windows uses faster SYSENTER
  - Addresses of functions stored in model-specific registers (MSR)
  - Require Ring 0 to modify

Hooking system call interrupts

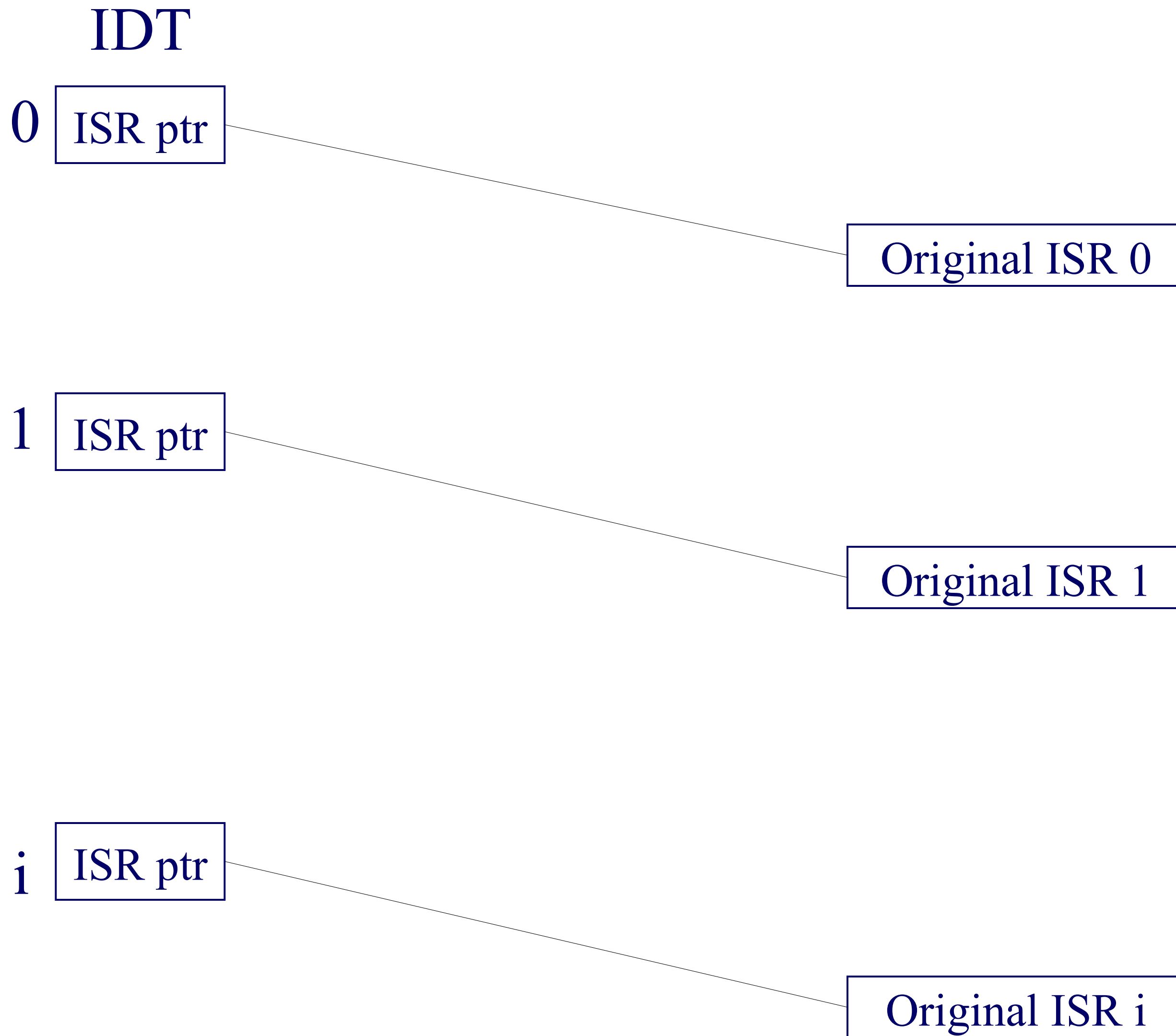


Hooking the IDT

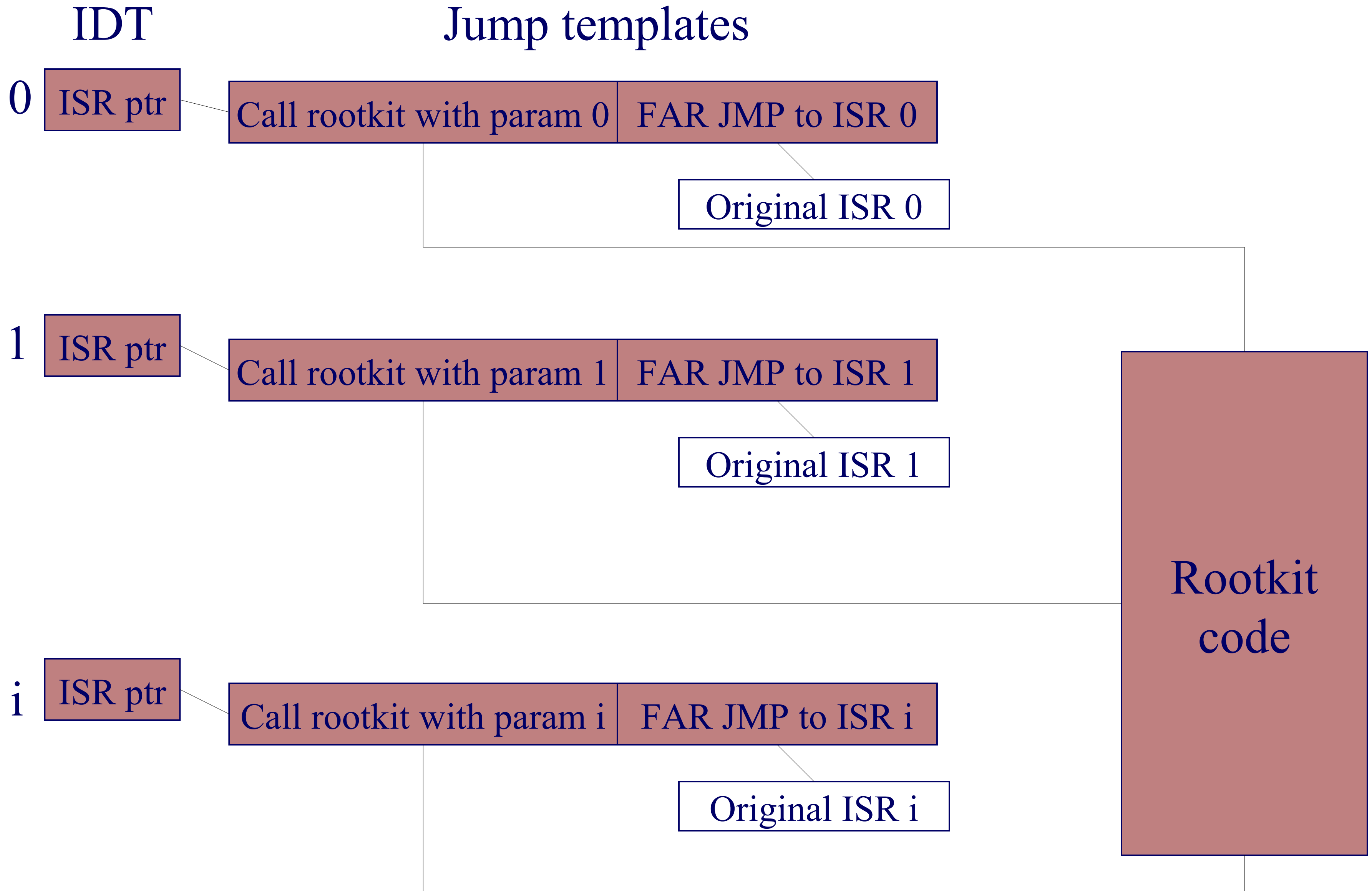
Patch entries in IDT to point to common hook table

- Easier than patching every interrupt service routine (ISRs)
- Hook every IDT entry, but include unique jump details to call back to original ISR
  - See code in Rootkit book

Hooking the IDT: Original IDT



Hooking the IDT: IDT with jump templates



System call processing

System Service Dispatch Table

- Kernel data structure that points to code which implements system calls in Win32, POSIX, and OS/2 subsystems
- Indexed by system call number
- Two ways for invoking calls
  - Use int 0x2e
  - Call SYSENTER instruction

Hooking the SSDT

Load rootkit as device driver

Replace SSDT entries to point to code in device driver instead of Ntoskrnl.exe or Win32k.sys

Later versions of Windows XP make memory that stores SSDT read-only (BSOD if you try to write)

- Change CR0 to disable memory protection in kernel
- Use Memory Descriptor Lists to change flags
- HOOK_SYSCALL, UNHOOK_SYSCALL macros

Hooking the SSDT

Example: Hiding processes

- Replace NTQuerySystemInformation function in SSDT
- Hook calls original function and filters results to remove rootkit entries from SystemInformationClass buffer that is returned
- Must update execution time statistics across all processes in list
  - If CPU doesn’t add up to 100%, someone will be suspicious

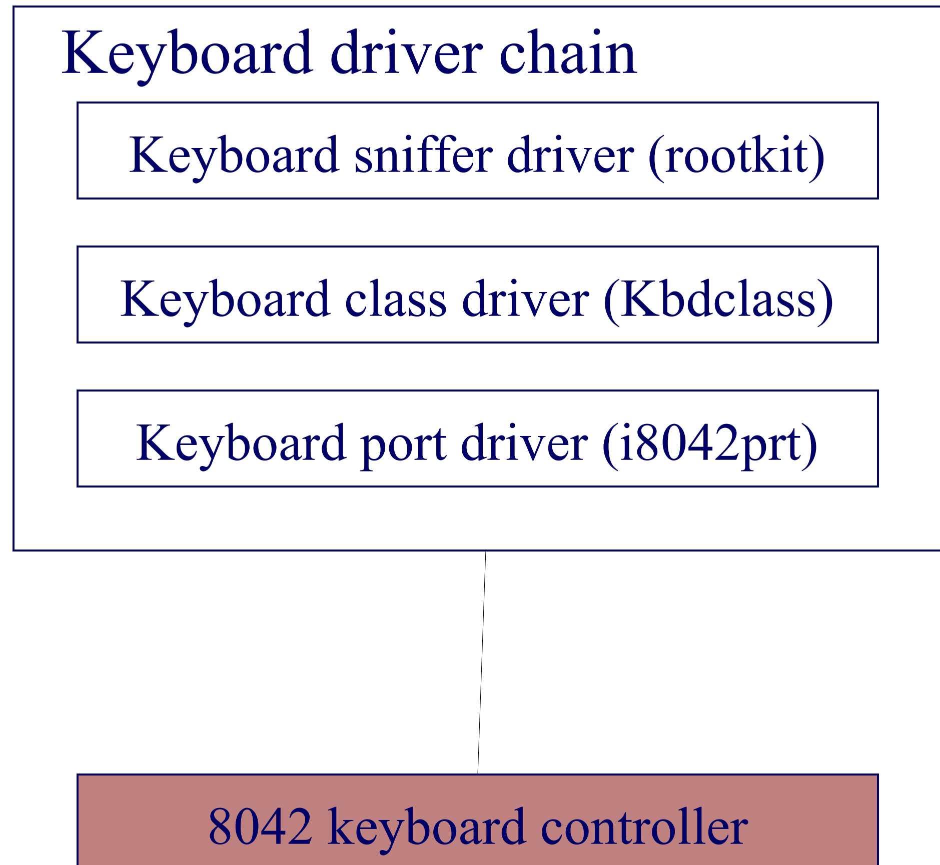
I/O processing

Drivers typically organized in chains to avoid re-implementing functions that can be shared

Example chain: keyboard drivers

- Drivers organized as a chain to implement key remappings, keyboard macros, etc.
- Lowest-level driver deals with direct access to bus and hardware device
- Next level deals with data formatting and error codes
- Each level intercepts data from lower level, modifies it, and passes it on to higher level
- Perfect for rootkits
  - Insert into chain to suppress events or eavesdrop on them

Keyboard chain example



Hooking I/O processing

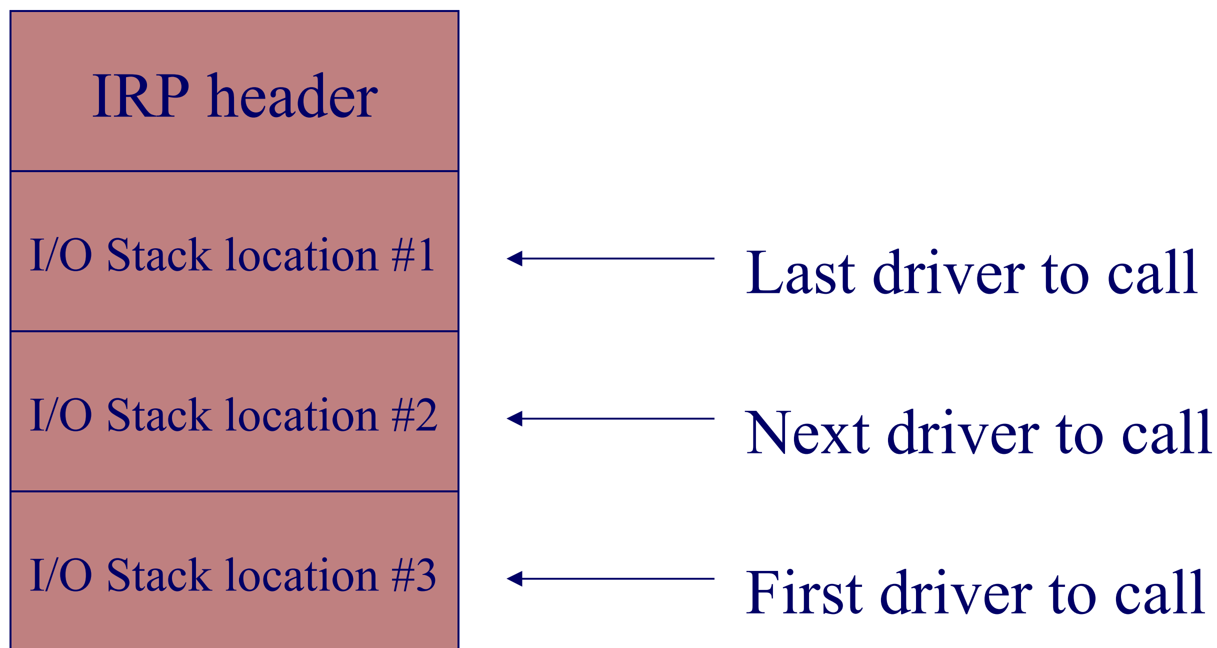
I/O Request Packet Function Table

- Function table contained in every device driver
- Lists IRP types it handles and addresses of functions that handle it
- Replace function table entries of file system or TCP queries
  - Insert new functions, modify existing functions
  - Good for hiding files and connections

Windows IRP (I/O request packet)

- Contains stack specifying routines of the driver chain
- I/O manager creates IRP and fills in IRP based on number of drivers in driver chain
- Inject keyboard sniffer in chain, IRP automatically updated
- Example: KLOG

I/O Request Packet



Other I/O hooks

File system drivers

- Rootkits store files in file system that must remain hidden
- Use layered file system drivers to hide all rootkit files
- In Windows, install hook on all available drive letters (HookDriveSet in book)
- Rootkit parses file name in QueryDirectory.FileInformationClass QueryBuffer
  - Deletes entries associated with rootkit

Kernel hooking issues

Memory pages containing SSDT and IDT are set to read-only in the page table

- Attacker must change pages to read/write in order to alter the pages
- Rootkits do this using CR0 trick or via registry key modifications
  - CR0 controls whether memory access protection in the kernel is enforced
  - WP bit = controls whether processor will allow writes to memory pages marked as read-only
- Counter-measure
  - scanners check integrity of original IDT
- Counter-counter-measure
  - Hackers create copy of IDT somewhere else, modify it, and change IDTR to point to modified one

Hybrid hooking

Use kernel to hook user process

- **Why?**
  - **Userland hooks are easier to implement functionality in**
  - **But, run at lower privilege level and can be detected by detection software running at Ring 0**
  - **Most detection looks at inclusion method**
- **Hook IAT without opening a handle to target process (which can be detected)**
  - **Kernel-based inclusion using PSSetImageLoadNotifyRoutine**
    - » Driver callback routine that is called every time an image is loaded into memory
    - » OS sends notification when your target process or DLL is loaded
    - » Driver callback is executed when load happens
  - **Use on kernel32.dll to be notified when all processes load**
  - **Modify IAT of processes in callback**

Hybrid hooking

Memory space for hooks

- **Must allocate additional memory in remote process for hooks**
- **New trick**
  - **User address 0x7ffe0000 and kernel address 0xffdf0000 map to same physical page**
    - » Kernel address writable, but user address is not
    - » Shared region is 4K, but kernel uses only 1K
    - » 3K available for rootkit on every process

Direct Kernel Object Manipulation

Direct Kernel Object Manipulation

Hooking disadvantages

- If someone knows where to look, hooks can usually be detected
- Modern kernel/hardware memory protection mechanisms may make some hooks unusable (read-only, no-execute protection)

DKOM

- Directly modify objects the kernel relies upon for its bookkeeping and reporting
- Normally, modifications to processes or tokens is done via Object Manager in kernel
  - Performs protection checks
- DKOM bypasses Object Manager and its checks

DKOM

Disadvantages

- Must disassemble format of object
  - Must know how object is used so code doesn't break after modification
  - Must know how object changes between versions of OS
- Works only for in-memory objects
  - Can not be used to hide files
  - Good for hiding processes, device drivers, ports
  - Good for elevating privilege levels

DKOM example: Process hiding

Process hiding from programs such as `Taskmgr.exe`

`ZwQuerySystemInformation` call lists running processes

- Traverses doubly linked list in the `EPROCESS` structure of each process
  - `FLINK` = pointer to process in front
  - `BLINK` = pointer to process in back
- Find a reference to `EPROCESS` of current process by calling `PsGetCurrentProcess`

Traverse list and update `FLINK` and `BLINK` pointers to point around process to be hidden

- Must ensure that hidden process has valid `FLINK` and `BLINK` pointers when hidden process exits via `PspExitProcess`
- Have them point to itself

Jamie Butler's FU Rootkit

- See Greg Hoglund's Rootkit book or BlackHat podcasts for details

DKOM example: Device Driver hiding

Device driver hiding from `drivers.exe` utility

- Windows Device Manager

`ZWQuerySystemInformation` with a `SYSTEM_INFORMATION_CLASS` of 11

- Modules also referenced via doubly linked list
  - Same trick used, modify `FLINK` and `BLINK`
- Finding the list is hard
  - Scan memory manually for `MODULE_ENTRY` object structure
  - Use Kernel Processor Control Block (KPRCB) for Windows XP and beyond
  - Use WinDbg to view members of the `DRIVER_OBJECT` structure (contains an undocumented field 0x14 into structure that is a pointer to driver's `MODULE_ENTRY`

DKOM example: Token creation

Token privilege and group elevation

- Process token derived from login session of user that spawned process
- Every thread within process has its own token

Use modifications to token to gain elevated privileges to install rootkit

- Win32 API: `OpenProcessToken`, `AdjustTokenPrivileges`, `AdjustTokenGroups`
- One can modify token privileges without elevated privileges by directly modifying privilege information in token
  - Stored in variable length portion of token
  - Example privileges: p 197
    - `SeCreateTokenPrivilege`, `SeAssignPrimaryTokenPrivilege`
    - `SeLockMemoryPrivilege`, `SeIncreaseQuotaPrivilege`
    - `SeUnsolicitedInputPrivilege`

DKOM example: Port hiding

Hiding network ports

- Modifying tables of open ports in TCPIP.SYS

Hiding while performing DKOM

Events generated upon all actions

- Registered callbacks upon certain events must be disabled to ensure stealth
- Example: Windows Event Log
  - Process being created
  - Parent PID
  - Username that owns process
  - Must change values in process token to other users to hide tracks

Running “underneath” kernel

Running “underneath” kernel

Restart victim virtualized

- Modify victim from below
- Examples: VMWare, VirtualPC, Plex86, Bochs, BluePill

Issues

- Complex implementation
- Fooling Administrator at console is difficult

User Mode Linux

Runs a Linux kernel inside a normal user-mode processes

- All UML instances run as guest operating systems
- Each UML is independent of others
- Similar to VMware or VirtualPC, but only for hosting Linux

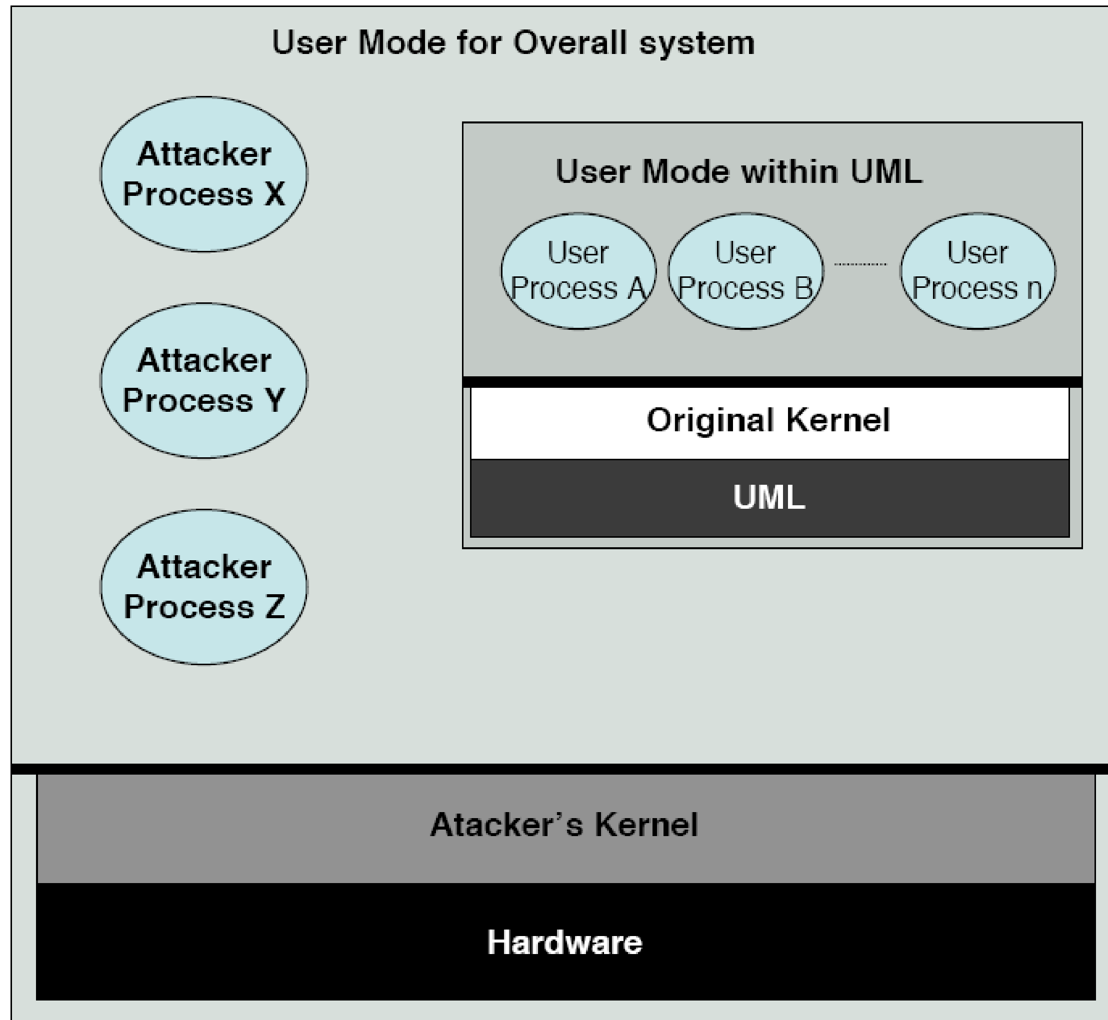
Originally for security

- Can compromise a single virtual machine without risking host

Malware purpose!

- In a cage you can’t see!
- Like … the matrix

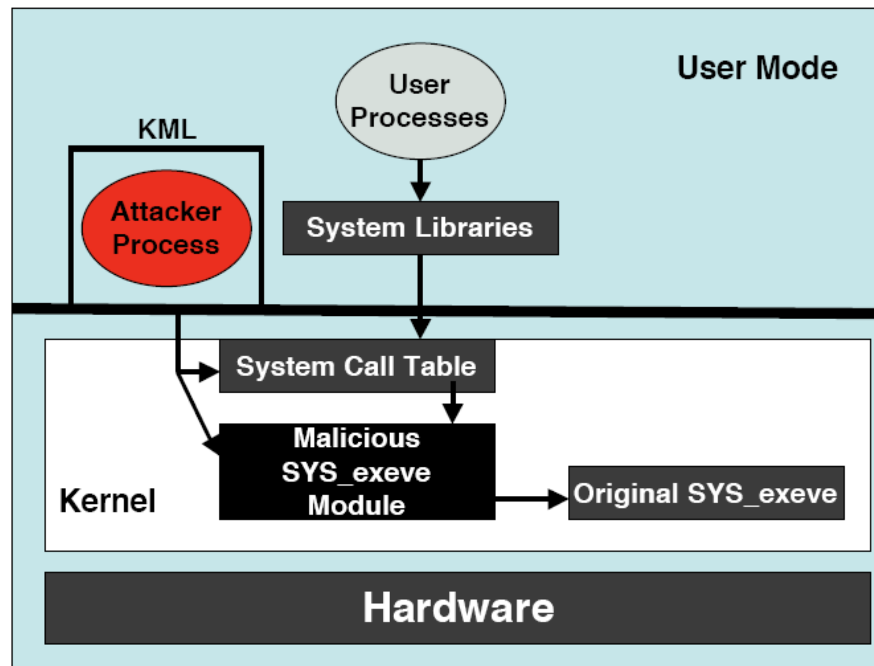
User Mode Linux



Kernel Mode Linux Project

Designed for software developer or administrator to run user code in kernel mode

- Installed in /trusted
- Runs in ring 0 for better performance
- Attacker could use KML for kernel-level attacks in previous section



Hardware virtualized rootkits

Red Pill, Blue Pill

- Place rootkit in hypervisor
- See BlackHat 2006 talks Rutkowska, Dai Zovi

Detecting hardware virtualized rootkits

- See BlackHat 2007 talk from Ptacek

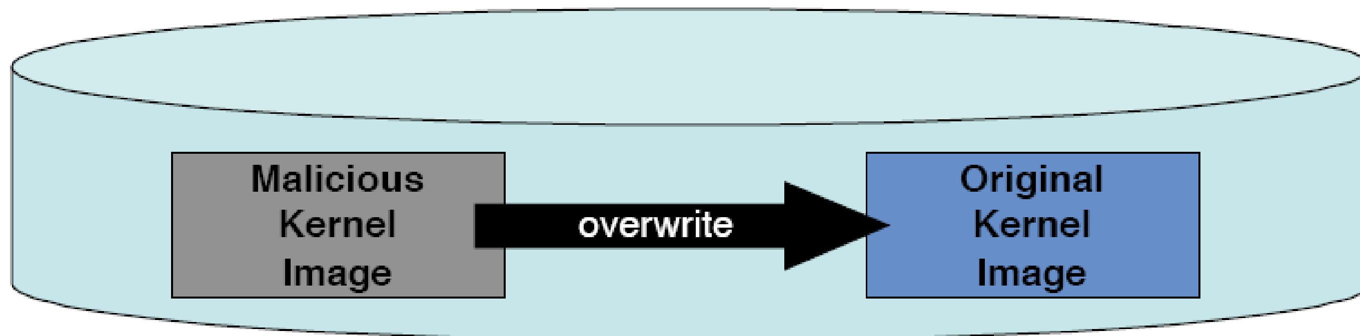
Patching the kernel image

Patching the kernel in file system

"Simple", easy, but requires reboot

- Problem: How to get a working kernel?
- Victim machine could have custom kernel or a specially compiled one
- Solution: Jbtzhm kernel modification kit
  - Appends itself to existing kernel
  - Adjust pointers to point to new code

Problem: Kernel integrity checking



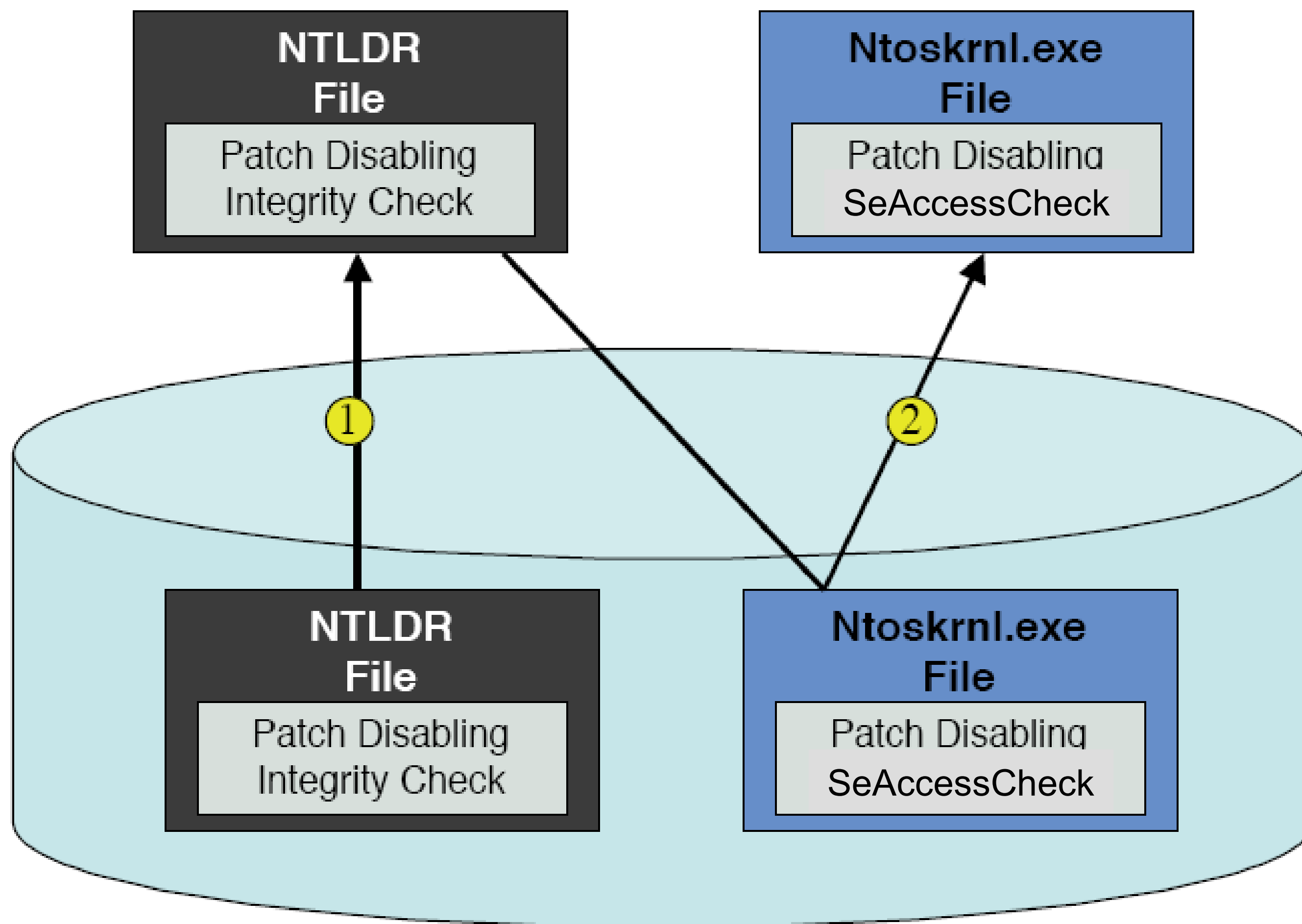
Patching the Kernel on disk

NTLDR verifies integrity of Ntoskrnl.exe before loading into memory

Bypassing kernel integrity checks

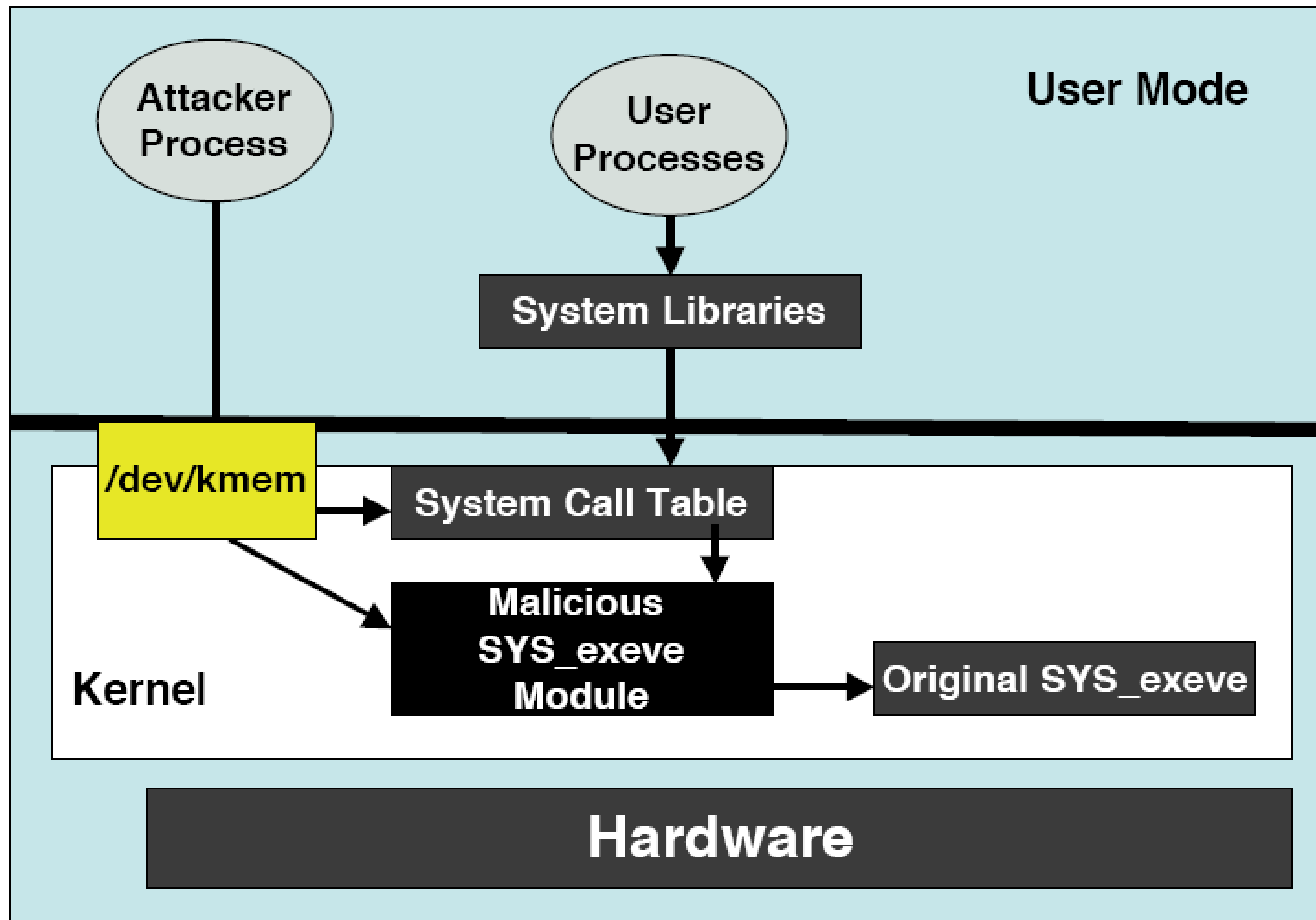
- Tamper with both NTLDR and Ntoskrnl.exe
- Overwrite machine instructions inside NTLDR to skip integrity check.
- Modify Ntoskrnl.exe to add rootkit
- Exploited in Bolzano and fun Love viruses

Windows NT Kernel example



At runtime on Linux, /dev/kmem

Attack at run-time instead…



At runtime in Windows

Global Descriptor Table (GDP)

- How memory is divided into segments
- Allocated to user programs and kernel
- Maps memory from 0x0000000 to 0xFFFFFFFF
- Kernel memory between 0x80000000 - 0xC0000000

GDP Attack (Hoglund)

- Add a new entry to GDT
- Modify GDT to allow someone running in Ring 3 to read/write from overlapping memory space in kernel region
- So... by writing machine language code that adds an entry to the GDT, the attacker can read/write kernel memory directly!

At runtime in Windows

Patch the running kernel disabling all security checking features

- Overwriting 4 bytes of kernel memory or image
- SeAccessCheck function of kernel verifies the user has permission.

Crazylords extension

- Uses Hoglund’s work and PhysMem by Mark Russinovich
- Manipulates \Devices\PhysicalMemory
- Allows attacker to view, search, and alter memory

Rootkit counter-measures

Detection

File integrity checking tools

- Good against the unknown

System anomaly detection tools

- Eg. Link counts on directories, suspicious network activity
- KSTAT
  - Detects changes to the system call table
    - scans /dev/kmem & compares System.map
  - Generates syscall fingerprints
- Intrusion Detection Systems
  - Cisco's Security Agent, Network Associates' Entercept, Watchguard's Server Lock

Anti-virus tools

- Signature based detection
- Good against the known
- What about the unknown?

Prevention

Prevent superuser privilege escalation

Kernel patches and updates

System configuration hardening

Non-modular kernels

Do not export system call table (or randomize it)

Examples

- Systrace
  - Tracks and limits system calls (Niels Provos)
- SELinux
  - Enhanced security controls
  - Applied mandatory access controls to critical system components and functionality (versus discretionary)

Response

Bring a CD-ROM with

- Fresh copy of antivirus tool installation and latest signatures.
- FIRE and Knoppix bootable Linux with forensic tools.

Reinstall…

What about backups?

Tools

Microsoft Detours library

- Library that provides routines for easy trampoline insertion

Debuggers and Disassemblers

- Ollydbg, SoftIce, WinDbg, IDA Pro

Tsearch

- Automated in-memory resource finder/modifier

Winject

- Automatic DLL injection and ejection

Process Explorer Tool

- Mark Russinovich

Microsoft Symbol Server

Example Rootkits

Adore Kernel Rootkit

Linux & FreeBSD kernel module

- File and directory hiding
- Process hiding
- Socket-hiding
- Hide promiscuous mode
- Backdoor
- Syslog filtering
- wtmp/utmp/lastlog filtering
- Hides malicious module (lsmod)
- Hidden privilege escalation

Kernel Intrusion System (KIS)

Linux kernel module

- Hides files, ports, processes, network ports, promiscuous mode
- Executable file redirection
- Nice GUI configuration
- Attach to any daemon, such as initd
- Configurable through the network, communication without listening ports
- KIS – hides processes/files only from unhidden processes

SucKit (Super User Control Kit)

Attacks /dev/kmem directly

- /dev/kmem contains the memory image of the running kernel
- No need for modules

NT Kernel Rootkit

Hoglund's Rootkit for WindowsNT, 2000, and XP ....

- http://www.rootkit.com
- Registry key hiding
- Execution redirection for any user-mode executable process
- Built-in keystroke logger
- Simple configuration... any file, registry key, or process with name _root_ automatically hidden.
- Hidden processes can see other hidden items.

Rustock.B

Details of rootkit

- Hooks MSR_SYSENTER
  - System call entry routine
- Changes the following APIs
  - ZwOpenKey
  - ZwEnumerateKey
  - ZwQueryKey
  - ZwCreateKey
  - ZwSaveKey
  - ZwDeviceIoControlFile
  - ZwQuerySystemInformation
  - ZwInitializeRegistry
  - Hides based on process name issuing call (RookitRevealer, Rkdetector, etc)
- Scans Windows kernel image for string
  - FATAL_UNHANDLED_HARD_ERROR
  - Replaces it with rootkit functions
- Creates bypasses for network communication
  - Modifies tcpip.sys, wanarp.sys, ndis.sys

End

Stealth techniques

Hiding processes/windows

- Window list scanning (Randomize name of exe)

Hiding DLL injection

- Removal from loaded module lists (MEM_ORDER_TYPE, LOAD_ORDER_TYPE, INIT_ORDER_TYPE)

Hiding in kernel

- Memory cloaking
  - modifying page tables and page mappings dynamically
- Alternative Data Stream (ADS)
  - Use of : instead of \
  - C:\windows\system32:lzx32.sys
  - Hides the driver from easy detection
  - Windows Explorer and cmd.exe do not show ADS
  - Change memory to replace ":" (0x3a) to a "\" (0x5c)
  - Attach ADS to directory since ADS viewers do not show this

ADS filename

Address	Hex dump	ASCII
0006FEA0	F4 5B 6F F6 F4 5B 6F F6 00 00 00 00 90 5B 6F F6	¶[o÷¶[o÷....É[o÷
0006FEB0	44 5C 6F F6 07 B2 54 80 00 0D DB BA 30 F1 3E 82	D\o÷•▓TÇ..█‖0±>é
0006FEC0	43 3A 5C 57 49 4E 44 4F 57 53 5C 73 79 73 74 65	C:\WINDOWS\syste
0006FED0	6D 33 32 3A 6C 7A 78 33 32 2E 73 79 [illegible] 00 BF E1	m32:lzx32.sys.┐ß