Detecting Rootkits

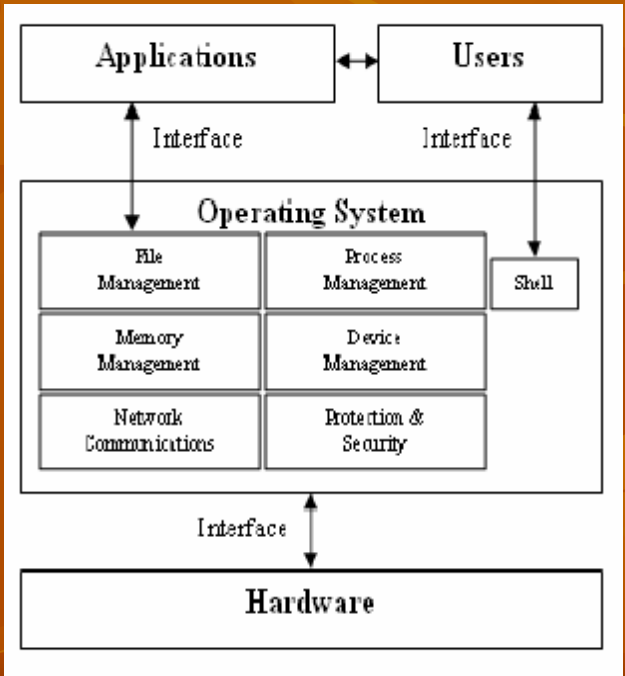
Cosmin Stejerean

What are rootkits

- A rootkit is a set of programs and code that allow a permanent or consistent, undetectable presence on a computer
- A rootkit is not an exploit, a trojan or a virus although it can make use of all of these technologies for delivery

How do rootkits work?

- A rootkit hides by intercepting and altering communications at the interfaces between various OS components.



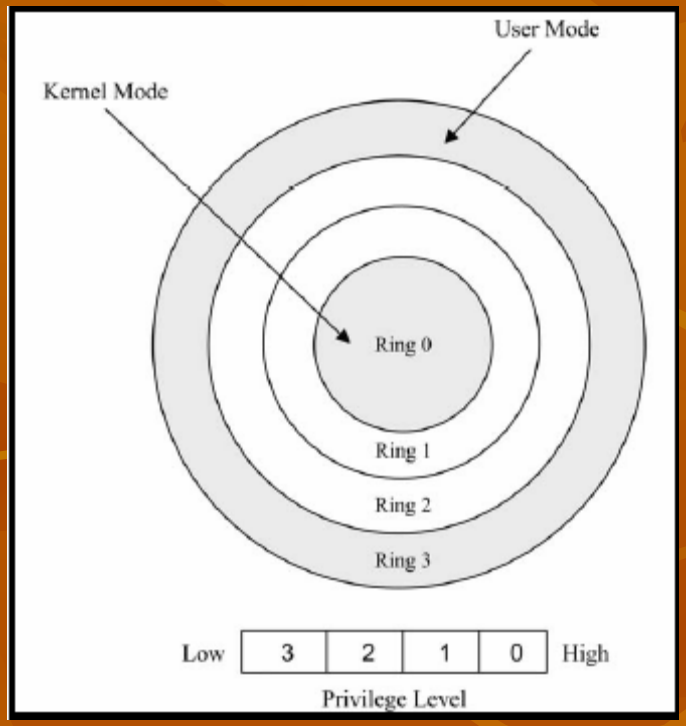
Affected Components

- I/O Manager
- Device & File System Drivers
- Object Manager
- Security Reference Monitor
- Process & Thread Manager
- Configuration Manager

Hardware Background

- x86 chips uses rings for access control
- Rings are 0 – 3 although Windows uses only Ring 0 and Ring 3
- Software cannot access any rings with lower numbers
- This is used for memory-access restrictions
- Ring 0 can execute privileged operations

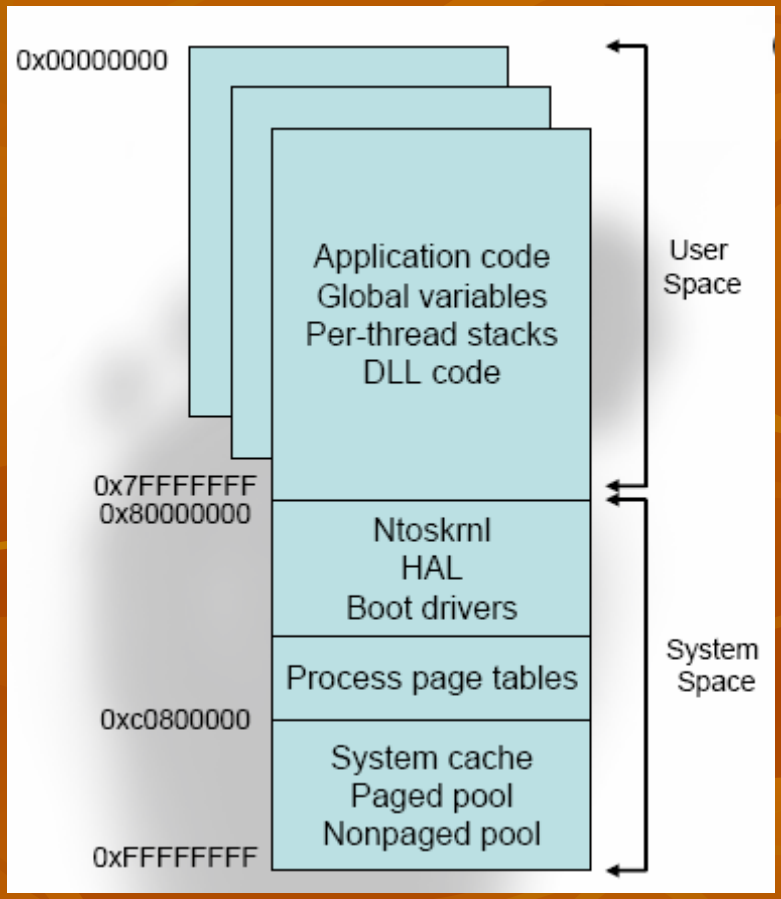
Rings


User Mode
Kernel Mode
Ring 0
Ring 1
Ring 2
Ring 3
Low
3
2
1
0
High
Privilege Level

Important Tables

- Global Descriptor Table (GDT)
- Local Descriptor Table (LDT)
- Page Directory
- Interrupt Descriptor Table (IDT)
- System Service Dispatch Table (SSDT)

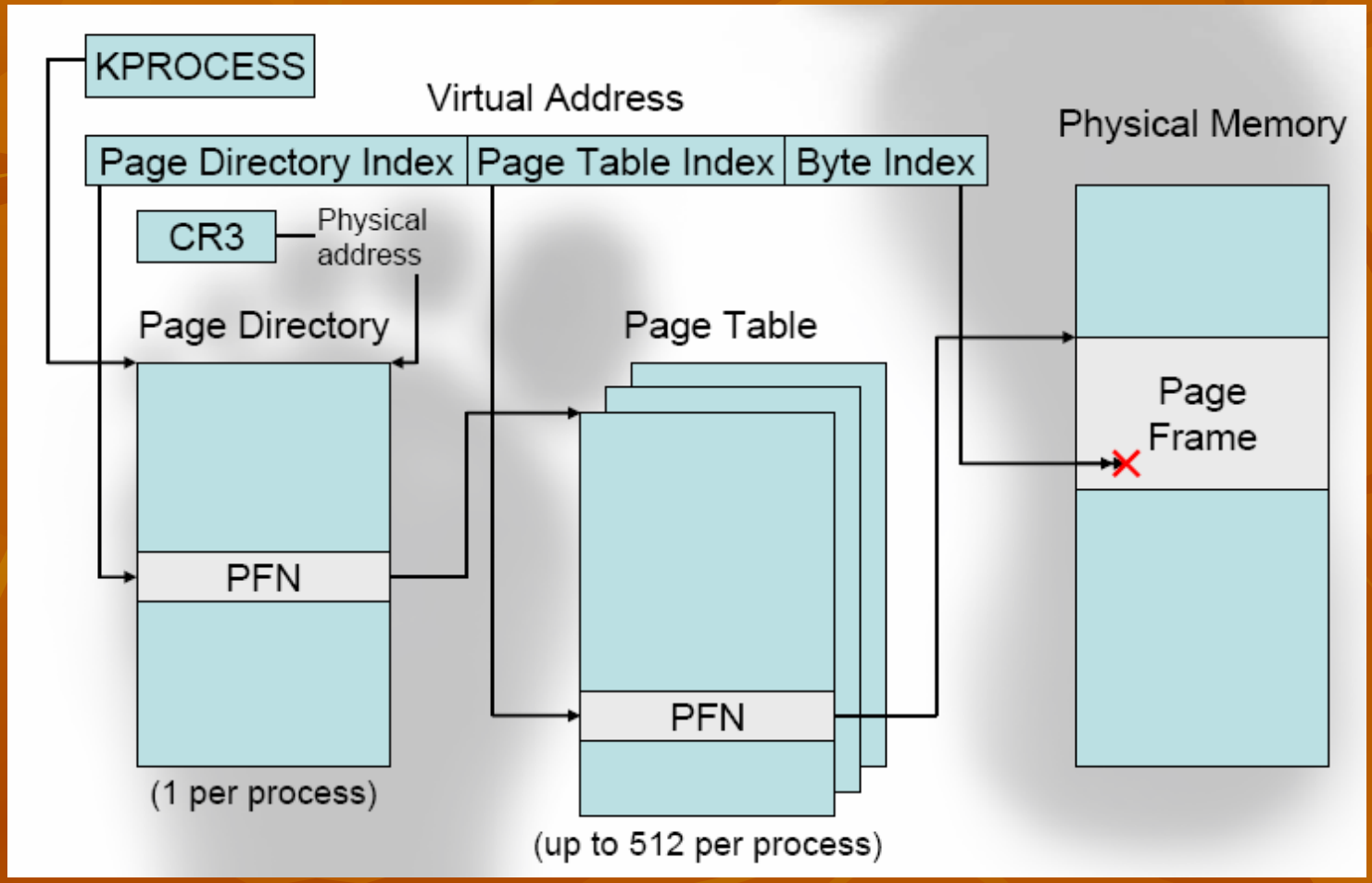
Virtual Memory



Virtual Memory

- Separate virtual and physical address spaces.
- •Virtual & physical address spaces are managed by dividing them into fixed size blocks.
- •The OS handles virtual to physical block mappings.
- •Virtual address space may be larger than physical address space.
- •Virtually contiguous memory blocks do not have to be physically contiguous.

Virtual Memory Lookups



Userland Hooking

- Import Address Table hooking
  - Simpler but easier to detect
  - Some problems due to DLL binding time
- Inline Function Hooking
  - More powerful but more complicated

IAT Hooking

- When code uses external DLLs the IAT stores a pointer to the location of each imported external function
- Once a rootkit binds to the memory space of a process it can replace some entries in this table to point to rootkit functions
- Most API calls are in this table and can be hooked

Inline Hooks

- The rootkit saves the first several bytes of a function and replaces it with a “jump” instruction
- This transfers control to the rootkit’s function and the rootkit can the call the original function and filter the results

Kernel Hooks

- Hooking the SSDT

SSDT

- Used to lookup functions to handle a given system call
- Rootkits can hook into this processing to sniff data, alter arguments or redirect the system call

IDT

- The IDT is used to locate the function that will handle a given interrupt
- This can be modified to allow user level programs to communicate with the rootkit running in the kernel or to sniff regular interrupts
- Problem because these are passthrough functions

Driver Hooking

- Can hook the major IO functions provided by a driver
- Also pass-through functions but can use callback functions

Other Methods

- Runtime patching by replacing code in memory
- Layered drivers
- Modifying in kernel memory structures such as the linked list of active processes or threads
- Hardware manipulation

Fighting Rootkits

- Many ways for rootkits to operate
- The OS cannot be trusted
- This makes rootkit detection nearly impossible
- It doesn’t mean we don’t have to try…

Prevent rootkits

- Need to detect rootkits being installed
- Involves hooking a large number of functions to detect all possible entry points
- Even if this can be done it is still hard to tell that the software is malicious
- This would require signatures but this cannot guard against new attacks

Detection

- Scan the memory for patterns (there are already counter measures for this)
- Check the operating system for hooks
- Locating inline hooks can be very painful since it requires disassembling the functions

Detecting hooks

- Get the address range of known kernel modules
- Check for functions that should fall within this address space but do not
- However not only malicious software will have hooks

Detecting inline hooks

- It is possible to check the first few bytes of each function for unconditional jumps and check if the destination is within acceptable address range

More Detection…

- Tracing
  - Have a clean baseline of the system
  - Trace system calls in the future and watch for extra instructions
- Behavior
  - Most promising method
  - Looks for odd behavior in the OS (“lies”)

Detecting Hidden Processes

- Hook SwapContext and when a new thread is being swapped in verify that it exists in the linked list of active threads or verify against call from API
- Get an entry for a process and use this to get the list of all handles. Each handle references the process it belongs to.

Detecting other items

- Similar techniques can be used to find modifications to the registry and file system

Other solutions for detection

- Scan from infected system and compare with scan from clean system
- Direct memory access using UDMA while system is running
- There are still many problems with these approaches

Conclusion

- Don’t get hacked in the first place and if you do WIPE the system.