Lecture 5

Rootkits

Hoglund/Butler (Chapters 1-3)

Rootkits

- A set of small and useful programs that allow an attacker to maintain access to “root” on a computer
  - Remote command and control (bots)
  - Software eavesdropping
- What they are not
  - Not an exploit
  - Not a virus

How do they work?

- Modifications to software
  - Patching binaries on disk
  - Easter eggs (for developers to access later)
  - Source-code modifications
- Two goals
  - Maintain access even through firewalls
  - Remain hidden from Host IDS, Network IDS, and forensic tools

The kernel

- Process management
- File system
- Security
- Memory management

Rootkit functions

- Hide files
- Hide registry entries
- Hide processes
- Modify boot service to start rootkit
- Modify network operations and services

Getting into the kernel

- Loadable modules
  - commonly used for third-party hardware support on Windows and Linux
  - device driver or kernel driver
    - runs in kernel
    - has access to all of the privileged memory of kernel
    - registers a name for access from user space (i.e. \\Device\\MyDevice)
- Used in conjunction with user-mode code
  - easier to debug and support functionality in user mode
  - open device name (i.e. \\Device\\MyDevice)
  - use ioctl/read/write to communicate to driver

Surviving reboot

- Using the “run key” in registry
  - Can be checked at boot time by anti-virus
  - Rootkit hides the value after being loaded
- Using a trojan or infected file
  - Replace a different .sys or executable that is run at boot-time
  - Modify search path to change DLL being used
- Using .ini files
  - Initialization files that specify executables to run and DLLs to load (win.ini)
- Registering as a driver
  - Loaded on boot, but also visible at boot since it needs registry key
  - Rootkit hides key after being loaded

Surviving reboot

- Registering as an add-on to existing application
  - Like a toolbar on a web browser
- Modifying on-disk kernel
  - Modify boot loader to allow new kernel to pass checksum integrity check
- Modify boot loader
  - Have boot loader apply patches to kernel before loading
- Other possibilities?

Hardware interaction: rings

- 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
- Rootkits typically try to run at Ring 0

Hardware interaction: tables

- Some CPU conditions require software routines to be handle them
  - Interrupts, exceptions, page faults, etc.
  - Too many conditions to keep track of all of them in hardware
  - CPU contains base address for tables that contain pointers to routines
- CPU tables
  - Global Descriptor Table (used to map addresses)
  - Local Descriptor Table (used to map addresses)
  - Page Directory (used to map addresses)
  - Interrupt Descriptor Table (used to find interrupt handlers)
- OS tables
  - OS-implemented tables not directly supported by CPU
  - System Service Dispatch Table (handling system calls)

Hardware interaction: tables

- Global and local descriptor tables
  - Code segment register (CS) points to where program is stored
    - Can be modified by any program via use of “far call” “ far jump” or “far return”
  - Call gates
    - Special descriptor that allows a new ring level to be specified when “far call” used
    - Useful for allowing user-mode programs to make a function call into kernel mode

Hardware interaction: tables

- Interrupt descriptor tables
  - 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)
  - Other gates
    - Trap gates = can be interrupted by maskable ints
    - Task gates = outdated, hardware support for switching task
      - Not used by windows or linux (which do it in software)

Hardware interaction: memory

- Page directory tables
  - CPU handles memory access
    - CPU checks whether process can open book (descriptor check)
      - Check to see if segment being accessed has sufficient privilege
    - CPU checks whether process can read certain chapter in book (page directory check)
      - Check to see if page table being accessed has sufficient privilege
    - CPU checks whether process can read particular page in chapter (page check)
      - Check to see if page being accessed has sufficient privilege
  - 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

Hardware interaction: SSDT

- System Service Dispatch Table
  - System calls
  - Two ways
    - Use int 0x2e
    - Call SYSENTER instruction

Subverting tables

- Overwriting SSDT and IDT entries
  - 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 (more later)

Multiprocessor issues

- Each CPU contains its own interrupt table
  - Hooks should be done across all CPUs
  - Drivers must perform synchronization to avoid system crash