Lecture 8

Rootkits
Hoglund/Butler (Chapter 7-8)

Avoiding detection

- Two ways rootkits can avoid detection
  - Modify execution path of operating system to hide rootkit presence
  - Modify data that stores information about processes, files, etc. that would reveal presence of rootkit
- This chapter
  - Modifying data that stores information on rootkit

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
    - WinDbg makes it easier
  - Must know how object is used so that code doesn’t break after modification
  - Must know how object changes between versions of OS
  - Only objects the kernel keeps in memory and uses for accounting purposes can be modified
    - Can not be used to hide files
    - Can be used to hide processes, device drivers, ports
    - Can be used to elevate privilege levels

Determining OS version

- User-mode
  - Win32 API `OSVERSIONINFOEX` structure
  - Returned by `GetVersionEx`
- Kernel-mode
  - Old versions of Windows `PsGetVersion` API
  - New versions (XP) of Windows `RtlGetVersion`
  - Parse string that is returned
- Either mode
  - Windows registry query
    - HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\NT\CurrentVersion*
    - `RegQueryValueEx`

Making it happen

- From user-mode
  - Must create IOCTLs to communicate with driver that performs DKOM
    - I/O Control Codes
  - IOCTLs included within IRPs
  - Example in book

Process hiding

- Objects referenced by user process 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`

Process hiding

- Hiding done based on process name
  - PIDs are pseudo-random
  - Name is included in `EPROCESS` structure
  - Location of name obtained via `GetLocationOfProcessName`
  - 16 byte character string (first 16 characters of binary on disk)
- 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
- What about process scheduler?
  - Apparently does not rely on `FLINK/BLINK`

Device driver hiding

- drivers.exe utility
- Windows Device Manager
  - Rely on `ZWQuerySystemInformation` with a `SYSTEM_INFORMATION_CLASS` of 11
  - Modules also referenced via doubly linked list
    - Same trick used
    - Modify `FLINK` and `BLINK` again
  - 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`

Issues in list traversal

- Processes and modules may be added or deleted while traversing
  - Must grab `PspActiveProcessMutex`
  - Must deal with possible pre-emption while modifying
    - Must run at `DISPATCH_LEVEL` to prevent

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 privelege information in token
    - Stored in variable length portion of token
    - Example privileges: p 197
      - `SeCreateTokenPrivilege`
      - `SeAssignPrimaryTokenPrivilege`
      - `SeLockMemoryPrivilege`
      - `SeIncreaseQuotaPrivilege`
      - `SeUnsolicitedInputPrivilege`
      - `etc`

Token privilege and group elevation

- Major problem
  - Adding privileges to variable length part of token
  - Must avoid increasing token size
  - Look to modify in place
  - Many privileges are included but are in a DISABLED state
    - `SE_PRIVILEGE_DISABLED`
    - `SE_PRIVILEGE_ENABLED_BY_DEFAULT`
    - `SE_PRIVILEGE_ENABLED`

Token privilege and group elevation

- Group elevation
  - Privileges associated with group membership
    - Determined by group SID
    - Adding SIDs to a process token adds privileges
  - Much more complicated than adding privileges
    - Requires allocating new memory and updating pointers in `SID_AND_ATTRIBUTE` table
    - i.e. unlike privileges there are no “disabled” SIDs to fill in

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

Other DKOM targets

- Hiding network ports
  - Modifying tables of open ports in TCPIP.SYS
- Recommended tools
  - SoftIce
  - WinDbg
  - IDA Pro
  - Microsoft Symbol Server

Hardware manipulation

- Physical access allows for hardware/firmware changes to be made
  - BIOS modifications
    - CIH virus destroyed BIOS
    - No known public rootkit for BIOS
  - BIOS modifications to PCI devices
- Example in book 8259 keyboard controller
  - Modifies HAL.DLL (Hardware Abstraction Layer)
  - Technically not a hardware modfication, but adds exploit at interrupt processing level using assembly commands specific to hardware
- Microcode update for processors
  - Used to fix bugs
  - Stored in BIOS and uploaded to processor every time machine boots
  - Protected by strong encryption on Intel processors (but not AMD processors)
    - AMD K8 microcode update driver
    - IA32 microcode driver