Lecture 6

Rootkits
Hoglund/Butler (Chapter 4)

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
- Focus of chapter
  - Modifying execution path via “hooking”

Hooking Windows

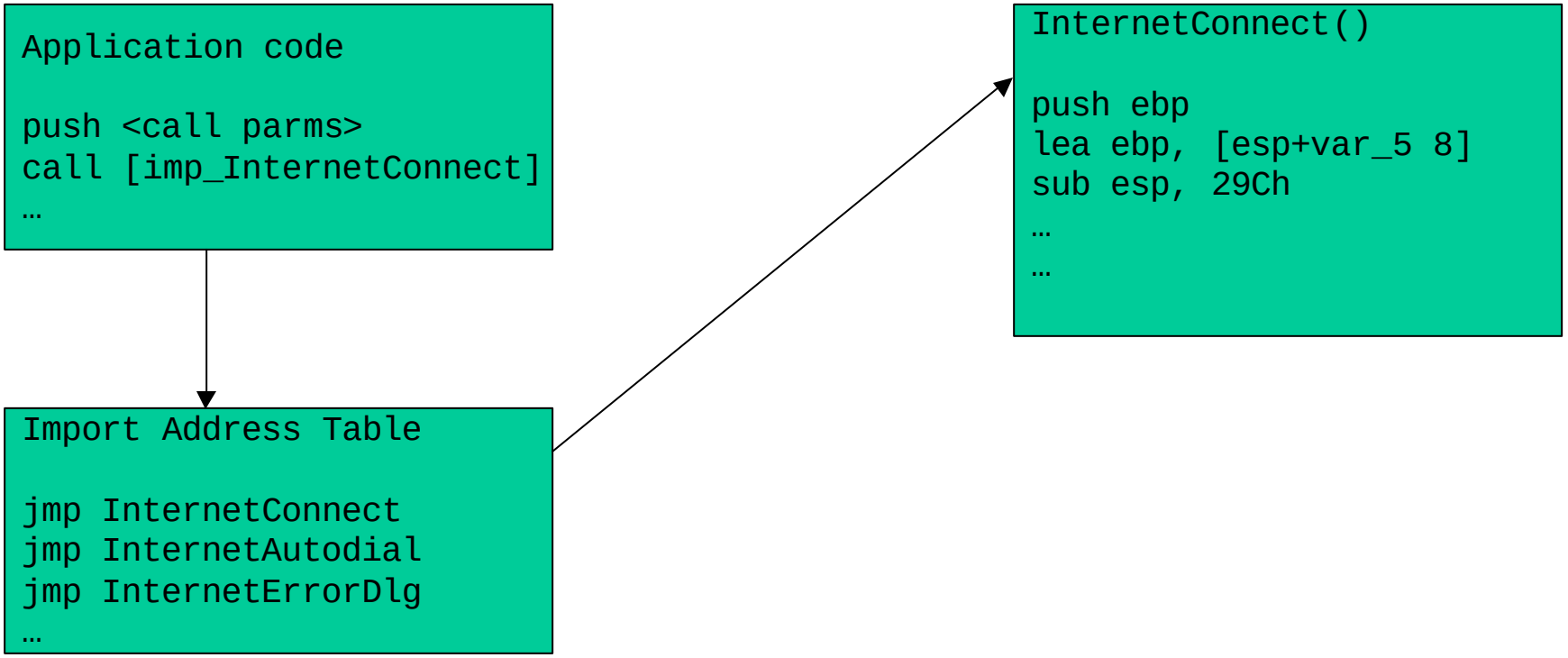
- Three OS subsystems processes depend on
  - Win32
  - POSIX
  - OS/2
- Processes rely on APIs provided by above
  - DLLs loaded at runtime into process address space
    - Kernel32.dll, User32.dll, Gui32.dll, Advapi.dll
    - Kernel32 loaded into private address space between 0x00010000 and 0x7FFE0000
    - Addresses of functions placed in Import Address Table (IAT)
  - Hooking
    - Modify code after it is loaded or modify IAT to point elsewhere
  - Example: Hiding files in a directory
    - Replace FindFirstFile(), FindNextFile() in Kernel32 to skip rootkit files

User hooks

- Modify execution path within process
- Run at a lower privilege level than most detection software
  - Thus, not as common nor as desirable
  - Kernel hooks described later

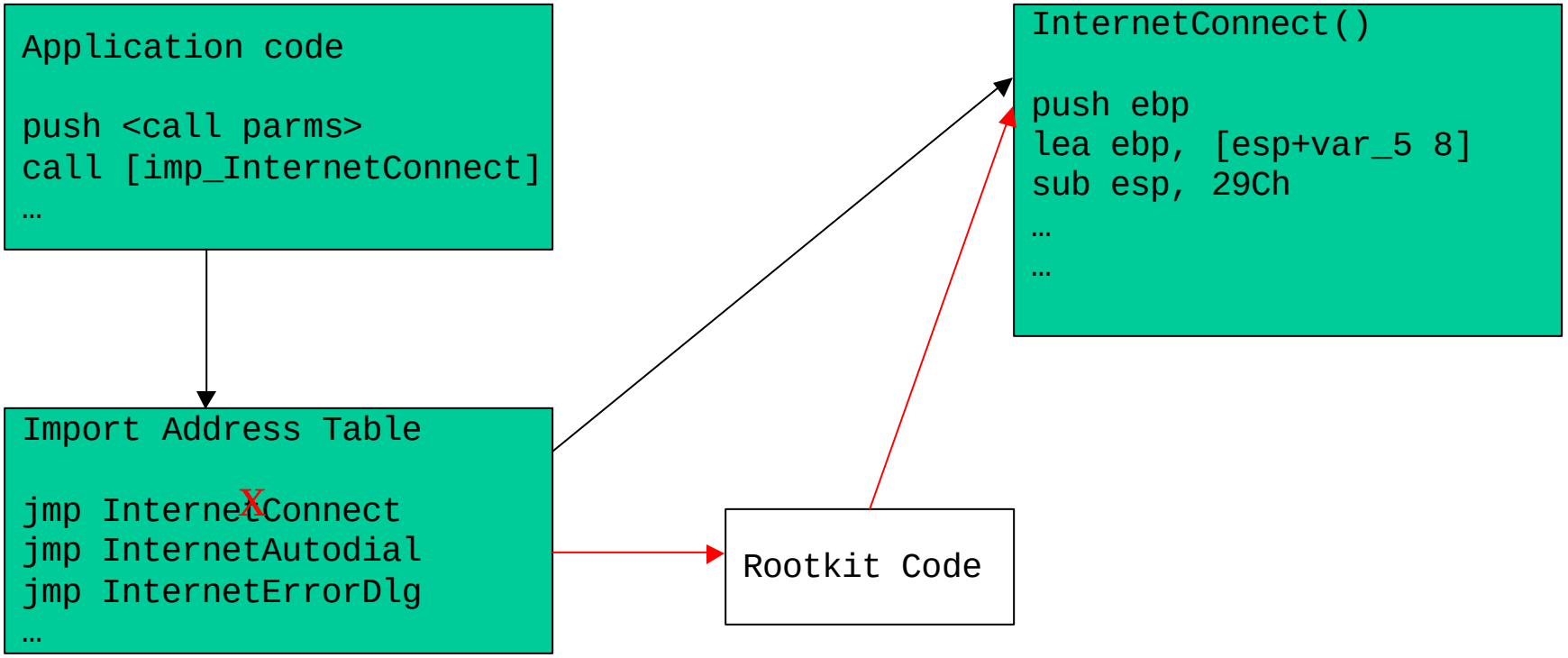
IAT hooking

- Normal operation for calling functions in system libraries



IAT hooking

- Load rootkit hook function into memory
- Replace target function's address in the IAT with address of hook function



IAT hooking

- Powerful and simple
- Easy to detect, but
  - Legitimate hooking common
    - Methods such as DLL forwarding makes benign vs. malicious hooks hard to discern
  - Late binding
    - Applications do late-demand binding where function addresses are not resolved until called
    - Reduces amount of memory used
    - Functions will not have addresses in IAT to hook!

Inline function hooking

- More powerful than IAT hooking
  - Do not have problems with binding time
  - Overwrite code bytes of target function so that no matter how it is resolved, your code will run
  - Can be used for both kernel and user functions

Inline function hooking

- Replace part of function preamble with a 5-byte unconditional jmp
  - Implement replaced instructions in rootkit code
  - Before XP

```
55      push ebp
8bec    mov ebp, esp
```

    - Hard to hook since you must disassemble user code
  - After XP

```
8bff    mov edi, edi
55      push ebp
8bec    mov ebp, esp
```

    - Easy to hook, exactly 5 bytes
    - MSFT intentionally did this to make hot patches easy

Inline function hooking

- Called a Detour
  - G. Hunt, D. Brubacker, “Detours: Binary Interception of Win32 Functions”, 3rd USENIX Windows NT Symposium, July 1999.
  - jmp instruction called a “detour”
  - original bytes of function saved in a “trampoline”
  - detour calls trampoline
  - trampoline implements 5 replaced bytes of original function, the function you want to execute and jmps back to original target function plus 5

Injecting a DLL

- Via the Registry
  - AppInit_DLL key
  - Add a DLL that hooks or modifies IAT, kernel32.dll or ntdll.dll
- Via Windows hooks
  - Windows allows you to hook window messages and events of another process
    - SetWindowsHookEx
    - Windows hook specifies Thread to hook to
    - Set to 0 and the system hooks all threads in the current Windows desktop!
  - Hook your DLL that modifies IAT, kernel32.dll, etc. to another process

J. Richter, “Load Your 32-bit DLL into Another Process’s Address Space Using INJLIB”, Microsoft Systems Journal/9 No. 5

Injecting a DLL

- Via remote thread
  - Windows allows you to create a thread on a remote process
  - CreateRemoteThread
    - Load rootkit DLL into remote process by specifying start routine as LoadLibrary and by giving it parameters that point to rootkit code using VirtualAllocEx

J. Richter, “Load Your 32-bit DLL into Another Process’s Address Space Using INJLIB”, Microsoft Systems Journal/9 No. 5

Kernel hooks

- More desirable as it places you on equal footing with detection software (Ring 0)
  - Kernel memory 0x80000000 and above
  - Cannot be accessed directly by processes unless through certain debugging APIs
  - Typically implemented as a device driver
  - Kernel hooks provide global scope

SSDT hooking

- System Service Descriptor Table
  - Kernel data structure that points to code which implements system calls in Win32, POSIX, and OS/2 subsystems
  - Indexed by system call number
- System Service Parameter Table
  - Specifies the number of bytes for the parameters of each call
- Hooking SSDT
  - Load rootkit as device driver
  - Replace SSDT entry to point to it 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

Using SSDT hooks

- 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

IDT hooking

- 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 SSDT entry with address of your hook
    - Hook calls KiSystemService upon completion
      - Execution does not return to IDT handler
  - Modern Windows uses faster SYSENTER
    - Addresses of functions stored in model-specific registers (MSR)
    - Require Ring 0 to modify

Hooking I/O

- Major I/O Request Packet Function Table
  - Function table contained in every device driver
  - Each IRP type has an entry in table for addresses of functions that handle it
  - Replace IRP of file system writes or TCP queries with rootkit
  - Good for hiding files and connections

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