RAIDE: **R**ootkit **A**nalysis **Id**entification **E**limination

By:

Jamie Butler & Peter Silberman

Agenda

- Overview
  - Rootkits
    - Hooks
      - KeServiceDescriptorTable
        - » Inline
        - » Overwrite
      - I/O Request Packet (IRP)
      - Interrupt Descriptor Table
      - Import Address Table
    - Hiding Processes
      - Detecting Hidden Processes
  - RAIDE
  - Demo using RAIDE to detect Shadow Walker, FUTo, Hacker Defender, and restore inline hook.

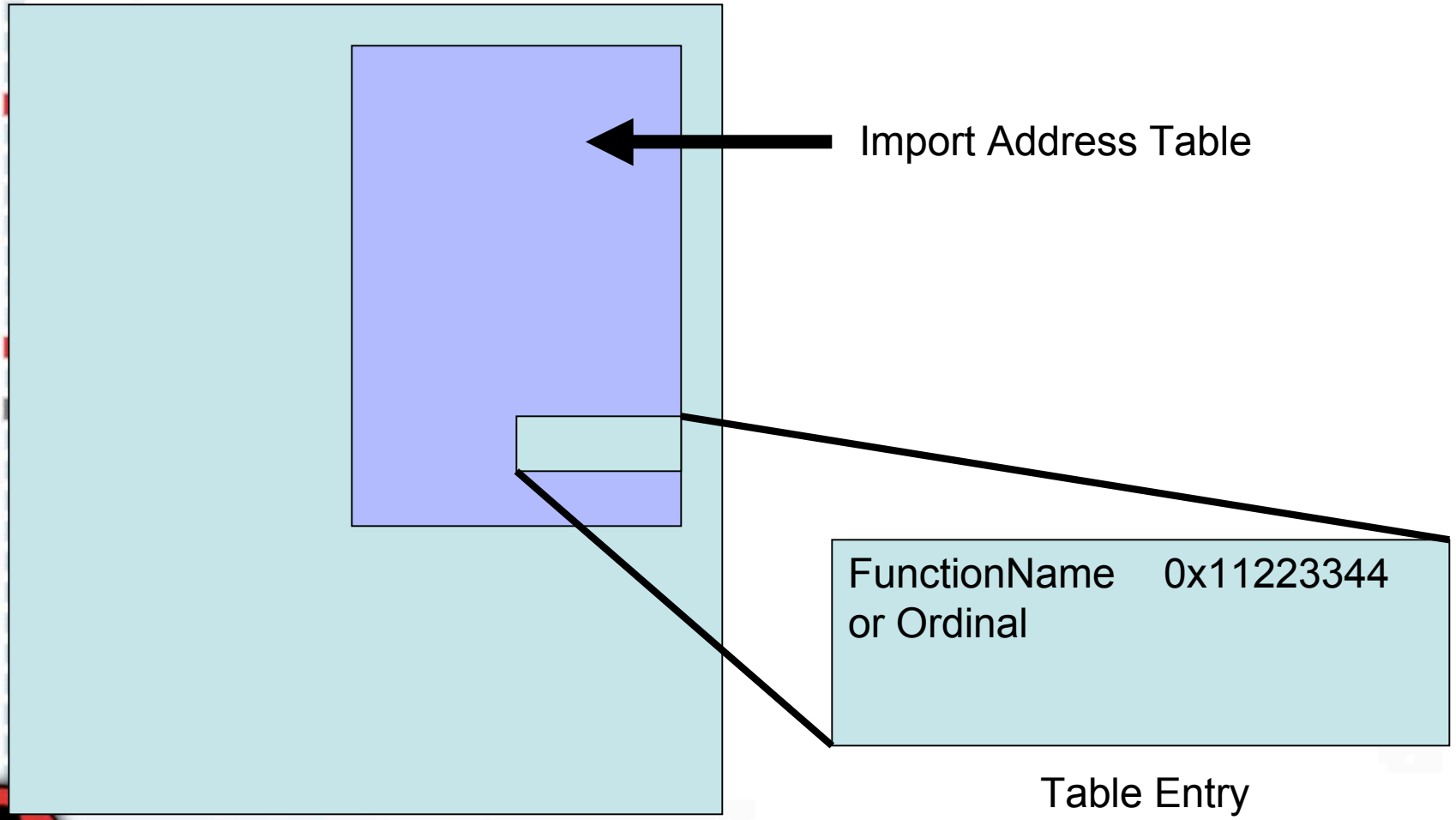
What is a rootkit

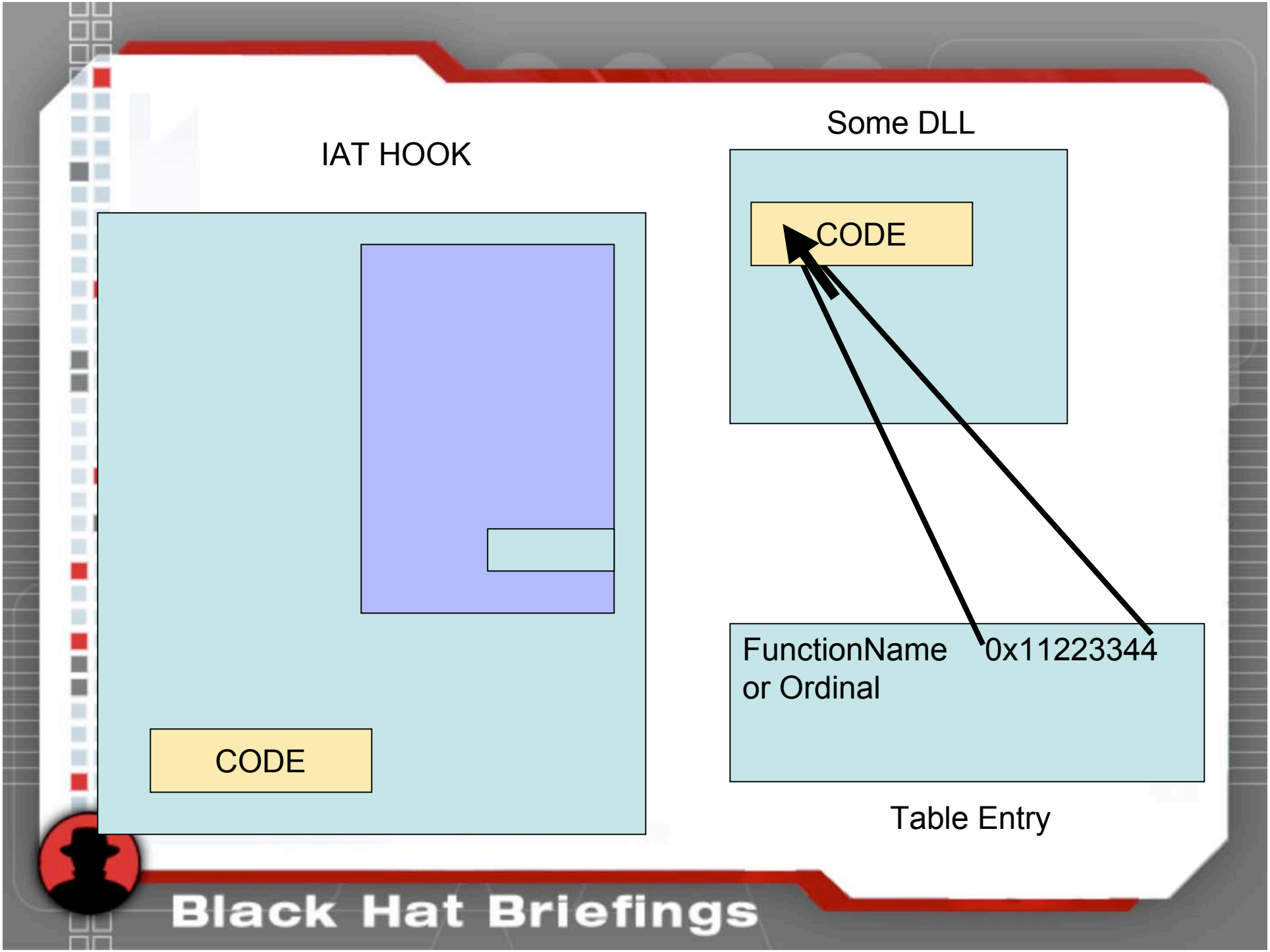
- Definition might include
  - a set of programs which patch and Trojan existing execution paths within the system
    - Hooks - Modifies existing execution paths of important operating system functions
  - The key point of a rootkit is stealth.
- History of Rootkits
  - Replace binaries like ls, ps, du, etc.
  - Bogus login program to steal passwords

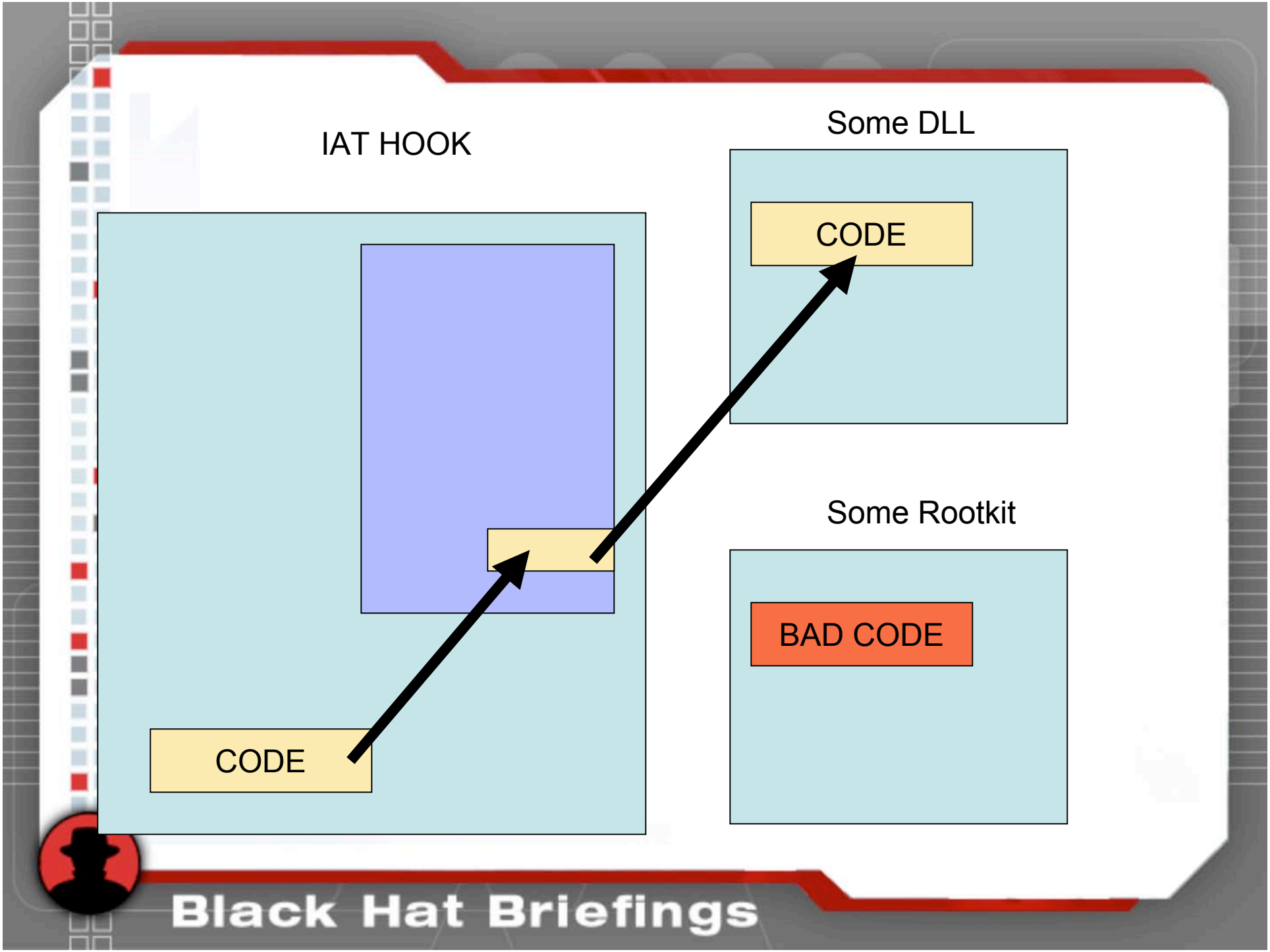
Hooking in User Land

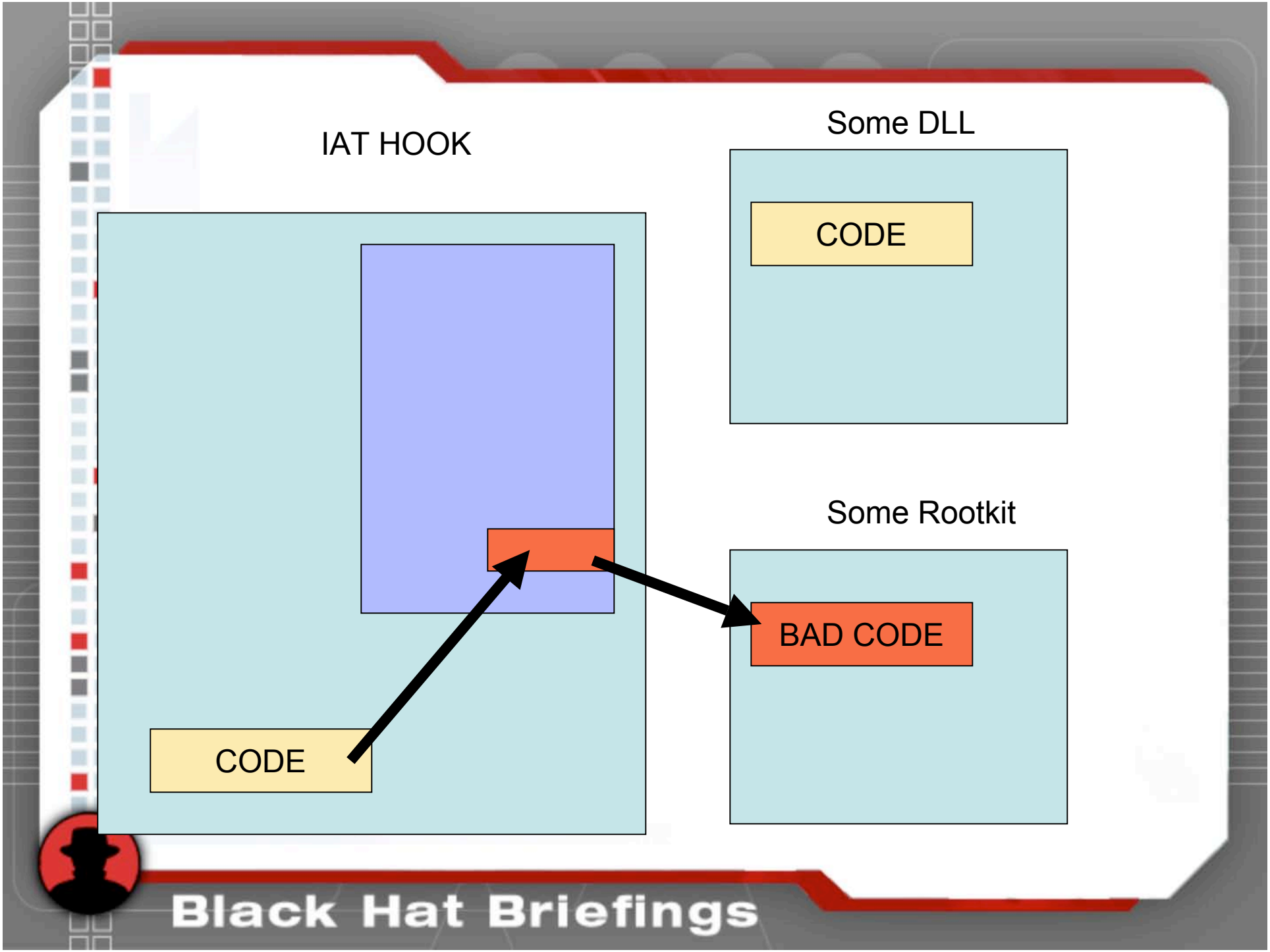
- IAT hooks
  - Hooking code must run in or alter the address space of the target process
    - If you try to patch a shared DLL such as KERNEL32.DLL or NTDLL.DLL, you will get a private copy of the DLL.
  - Three documented ways to gain execution in the target address space
    - CreateRemoteThread
    - Globally hooking Windows messages
    - Using the Registry
      - HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs

IAT HOOK


Import Address Table
FunctionName 0x11223344
or Ordinal
Table Entry

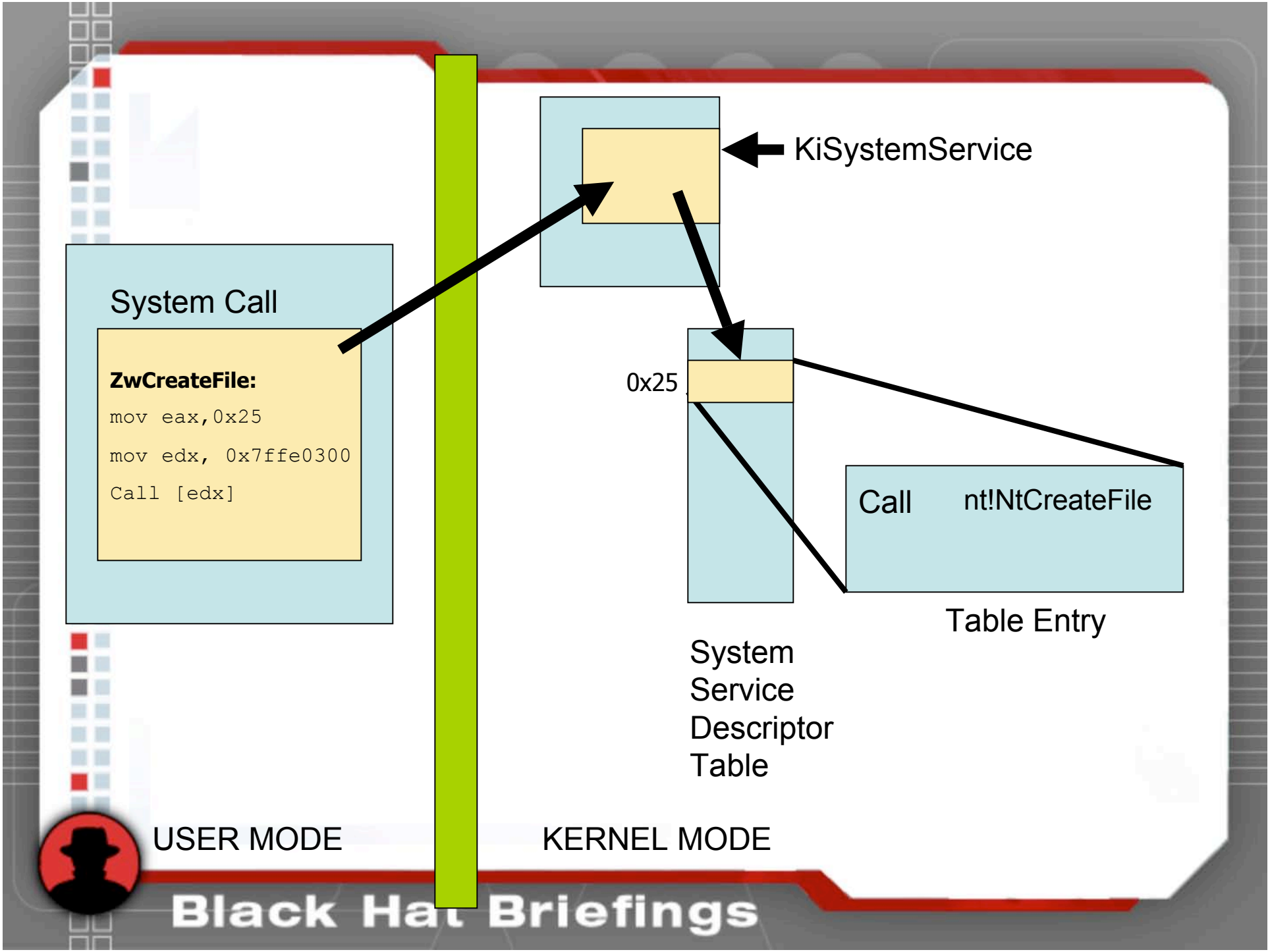

IAT HOOK
CODE
Some DLL
CODE
FunctionName
or Ordinal
0x11223344
Table Entry

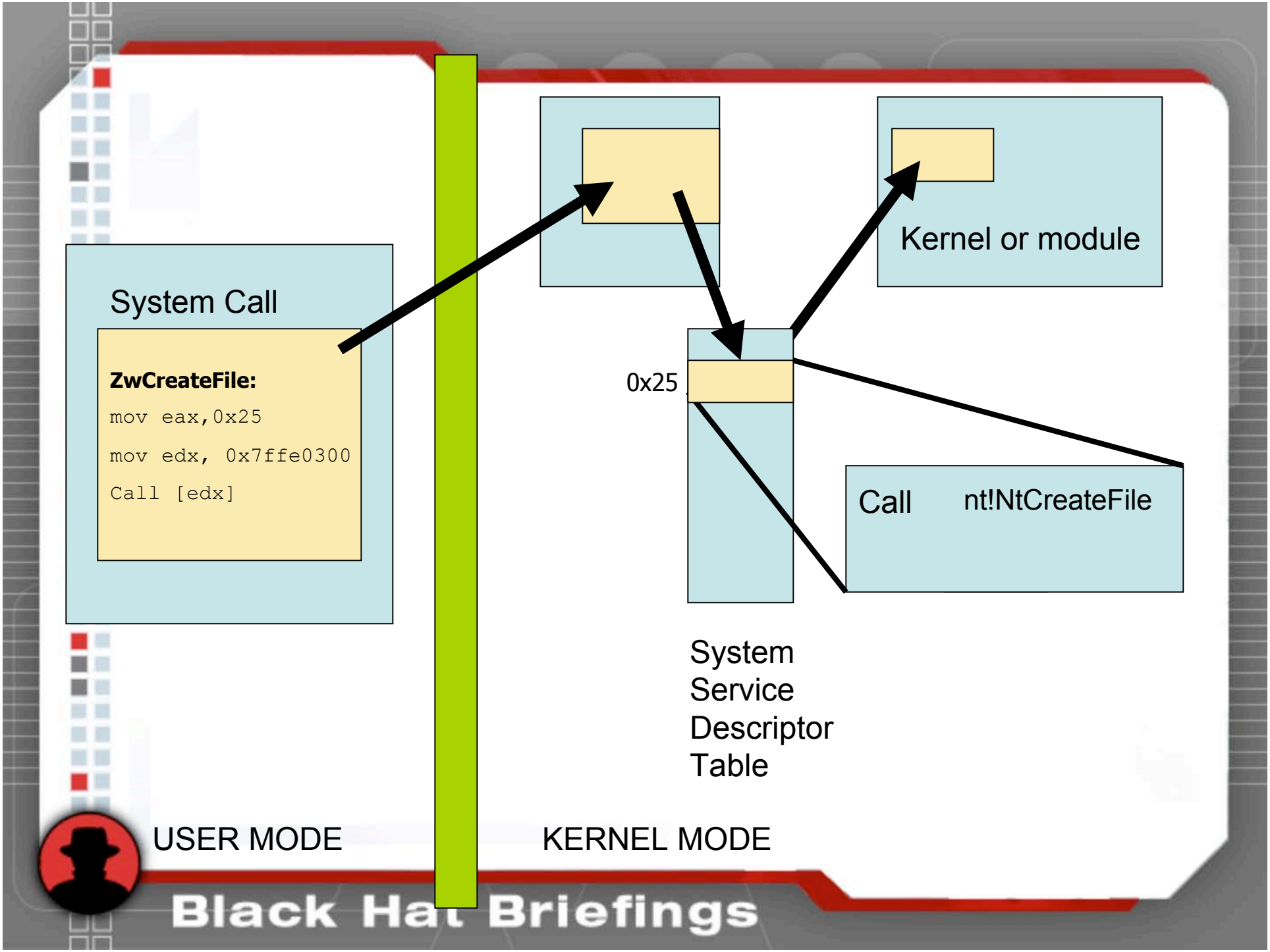

IAT HOOK
Some DLL
CODE
Some Rootkit
BAD CODE
CODE

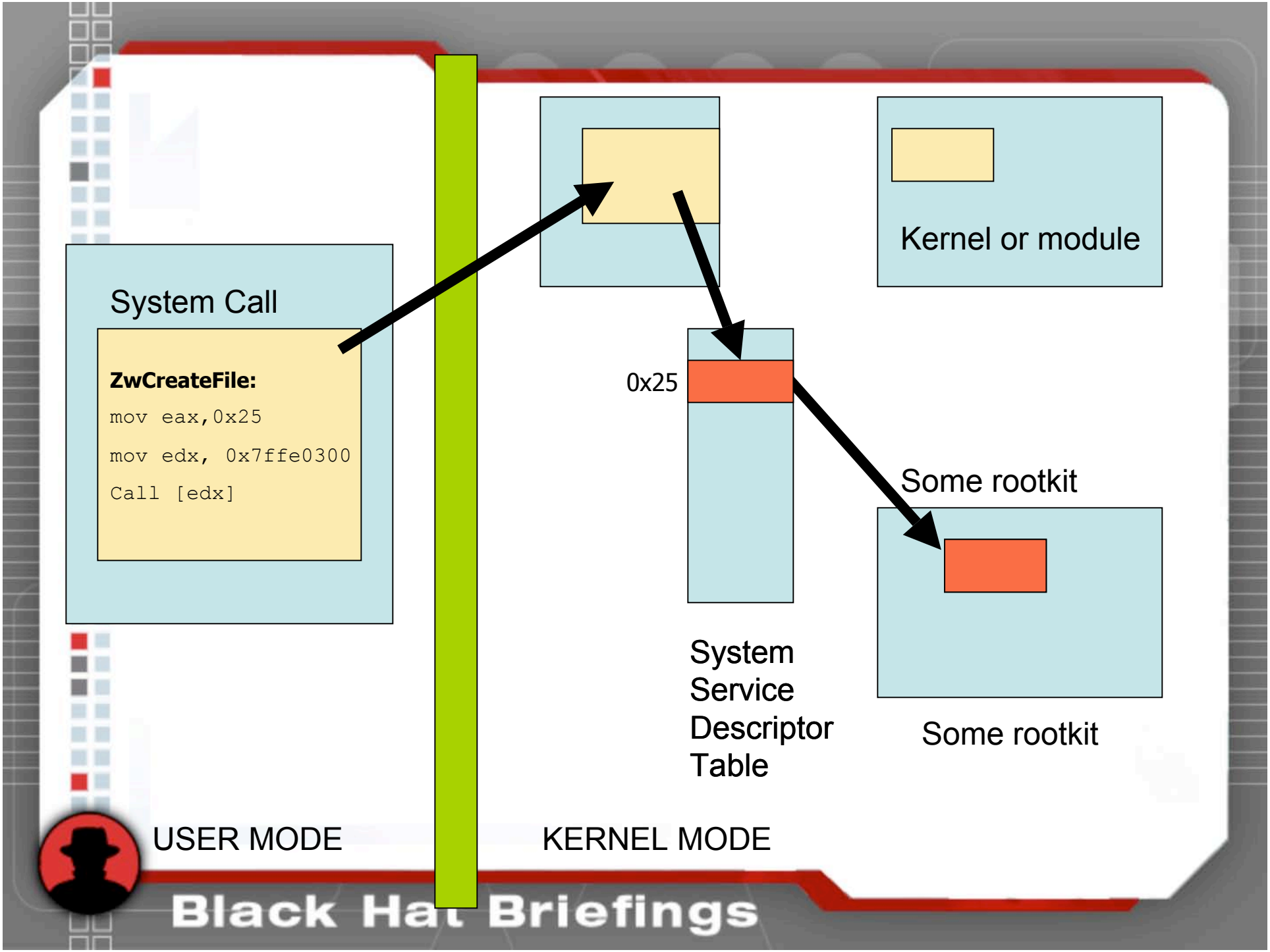

IAT HOOK
Some DLL
CODE
Some Rootkit
BAD CODE
CODE

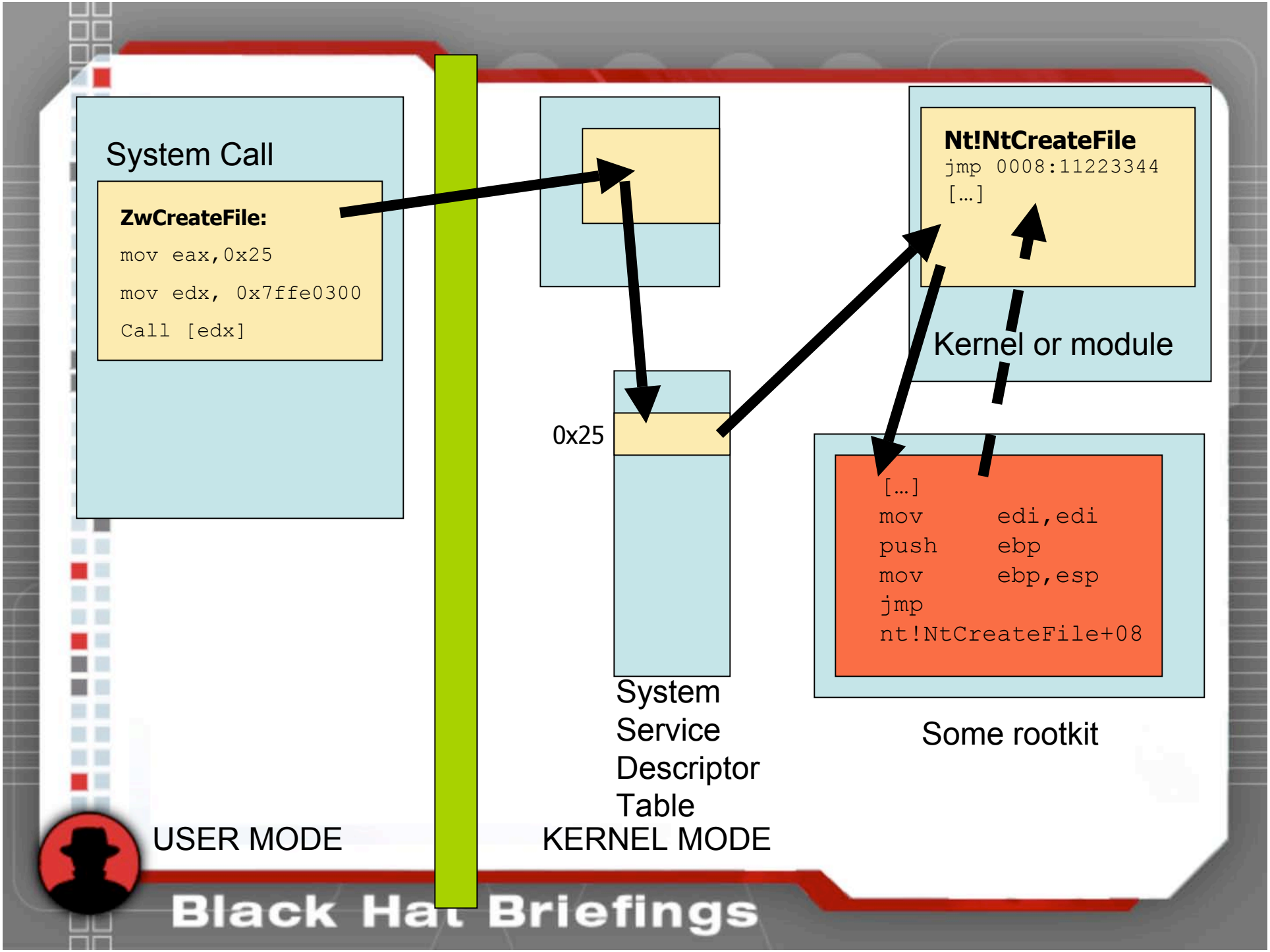
Hooking in Kernel Space

- The operating system is global memory
- Does not rely on process context
  - Except when portions of a driver are pageable
- By altering a single piece of code or a single pointer to code, the rootkit subverts every process on the system


KiSystemService
System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
0x25
Call nt!NtCreateFile
Table Entry
System
Service
Descriptor
Table
USER MODE
KERNEL MODE
Black Hat Briefings


System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
Kernel or module
0x25
Call nt!NtCreateFile
System
Service
Descriptor
Table
USER MODE
KERNEL MODE
Black Hat Briefings


System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
Kernel or module
0x25
Some rootkit
System
Service
Descriptor
Table
Some rootkit
USER MODE
KERNEL MODE
Black Hat Briefings


System Call
ZwCreateFile:
mov eax,0x25
mov edx, 0x7ffe0300
Call [edx]
Nt!NtCreateFile
jmp 0008:11223344
[…]
Kernel or module
0x25
[…]
mov edi,edi
push ebp
mov ebp,esp
jmp
nt!NtCreateFile+08
System
Service
Descriptor
Table
Some rootkit
USER MODE
KERNEL MODE
Black Hat Briefings

I/O Manager and IRP Hooking

- System Calls
  - NtDeviceIoControlFile
  - NtWriteFile
  - Etc.
- Requests are converted to I/O Request Packets (IRPs)
- IRPs are delivered to lower level drivers

I/O Manager and IRP Hooking

- Every driver is represented by a DRIVER_OBJECT
- IRPs are handled by a set of 28 function pointers within the DRIVER_OBJECT
- A rootkit can hook one of these function pointers to gain control

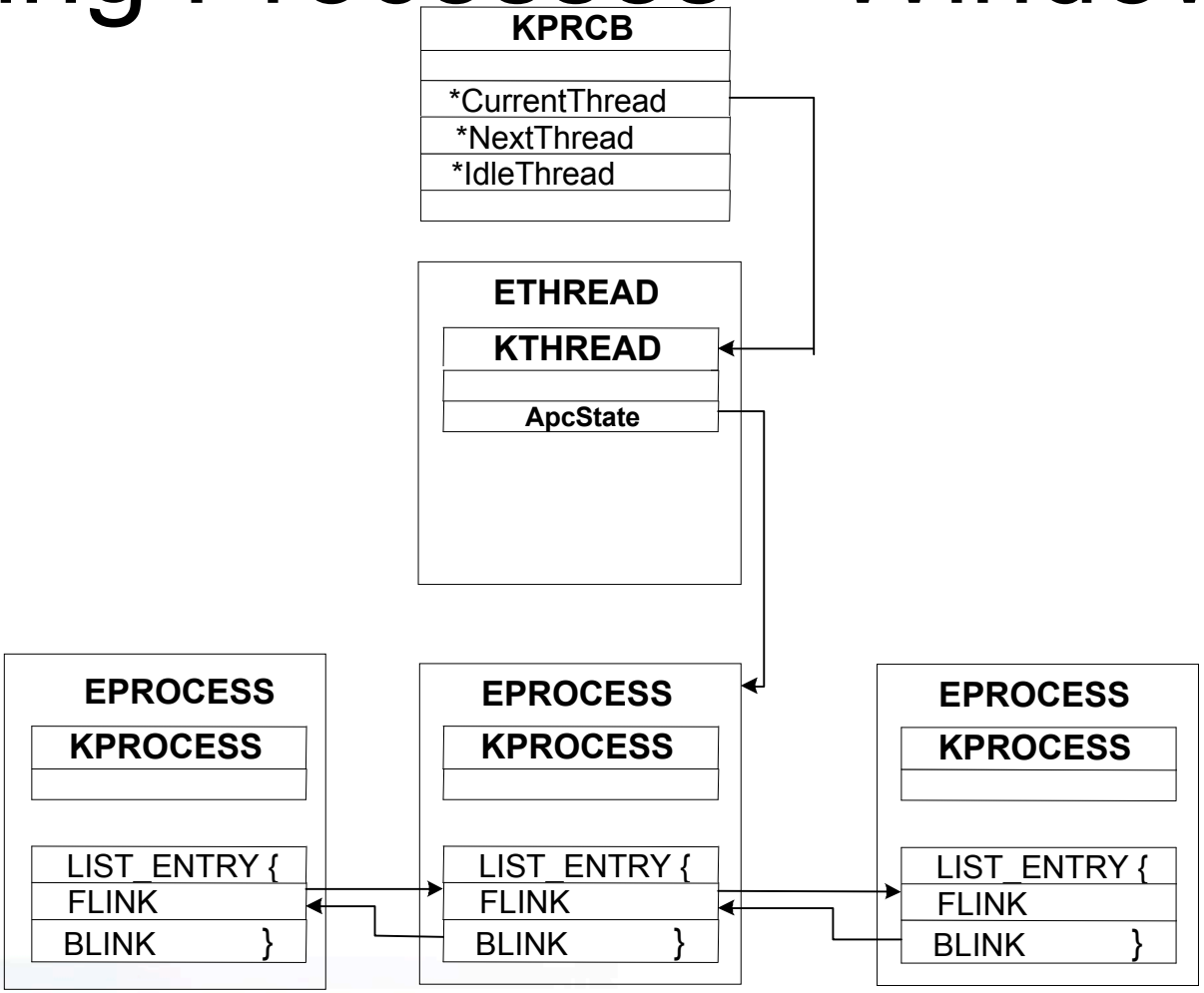
Interrupt Descriptor Table Hooks

- Each CPU has an IDT
- IDT contains pointers to Interrupt Service Routines (ISRs)
- Uses for IDT hooks
  - Take over the virtual memory manager
  - Single step the processor
  - Intercept keystrokes

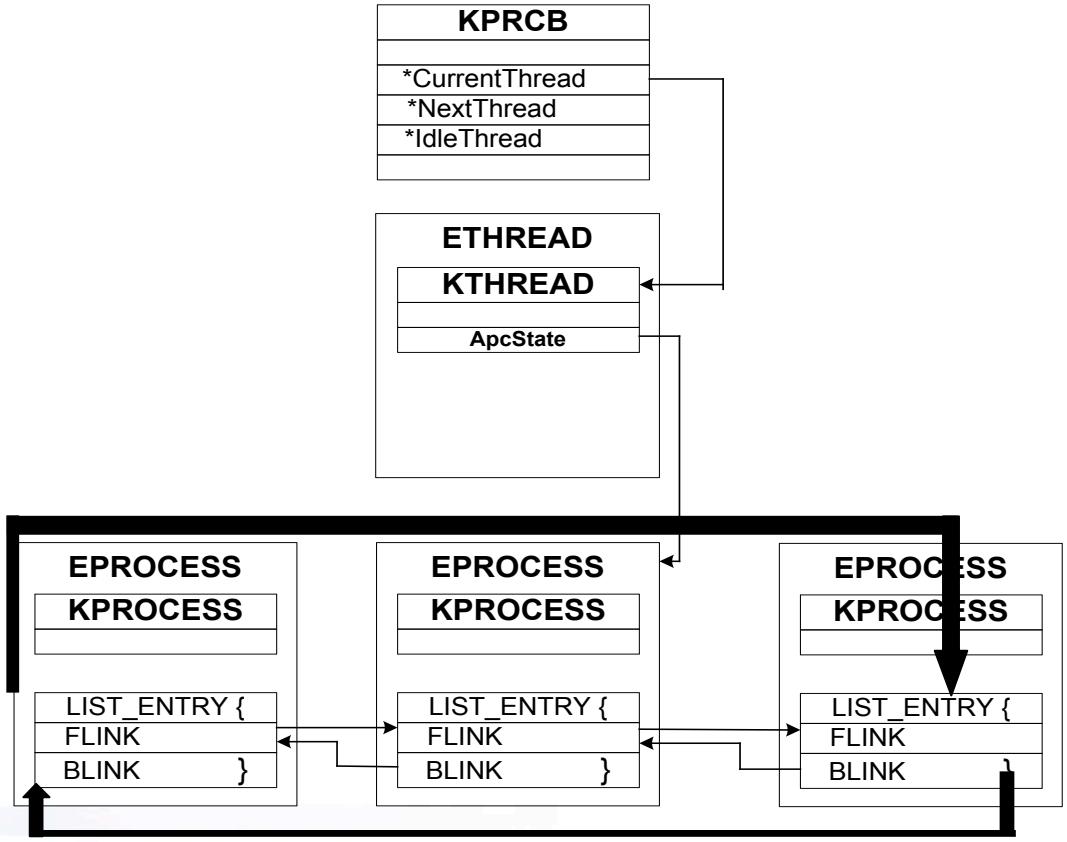
Hiding Processes

- DKOM Uses
  - Hide Processes
  - Add Privileges to Tokens
  - Add Groups to Tokens
  - Manipulate the Token to Fool the Windows Event Viewer
  - Hide Ports

Hiding Processes - Windows



Hiding Processes - Windows



FUTo – Hiding In The Handle Table

- FUTo
  - Uninformed Journal Vol. 3
  - New version of FU
  - Hides from IceSword and Blacklight
- Let's understand the handle table

Kernel Structures: Handle Tables

- Handles are an index into the Handle Table for a particular object
- Objects represent processes, threads, tokens, events, ports, etc.
- The kernel/object manager must do the translation from a handle to an object
  - Single point of access ensures security checks can be performed

Kernel Structures: Handle Tables

- Handle Table entries are 8 bytes each
- lkd> dt nt!_HANDLE_TABLE
- +0x000 TableCode : Uint4B
- +0x004 QuotaProcess : Ptr32 _EPROCESS
- +0x008 UniqueProcessId : Ptr32 Void
- +0x00c HandleTableLock : [4] _EX_PUSH_LOCK
- +0x01c HandleTableList : _LIST_ENTRY
- +0x024 HandleContentionEvent : _EX_PUSH_LOCK
- +0x028 DebugInfo : Ptr32 _HANDLE_TRACE_DEBUG_INFO
- +0x02c ExtraInfoPages : Int4B
- +0x030 FirstFree : Uint4B
- +0x034 LastFree : Uint4B
- +0x038 NextHandleNeedingPool: Uint4B
- +0x03c HandleCount : Int4B
- +0x040 Flags : Uint4B
- +0x040 StrictFIFO : Pos 0, 1 Bit

PspCidTable

- PspCidTable
  - Job of PspCidTable is to keep track of all the processes and threads
  - Relying on a single data structure is not a very robust
  - Alterating one data structure
    - OS has no idea hidden process exists

Removing From PspCidTable

- To hide from PspCidTable scanners:
  - Obtain PspCidTable by scanning PsLookupProcessByProcessId or GetVars
  - Parse PspCidTable for references to rogue process' EPROCESS
  - Set those values to 0
  - Setup process notify routine
    - Safely restore PspCidTable as process is terminated
- Other tables to remove references from:
  - CRSS
  - EPROCESS Handle Table
  - Beyond the scope of this talk (Read the Uninformed article)

Detecting Processes

- Blacklight Beta
  - Released in March 2005
  - Good hidden process and file detection
- IceSword 1.12
  - Robust tool offering:
    - SSDT Hook Detection
    - Hidden File and Registry Detection
    - Hidden Process
    - Hidden Ports and socket communication Detection
- Common flaw
  - Both application uses the Handle Table Detection method

Detecting Hidden Processes PID Bruteforce

- Blacklight
  - Bruteforces PIDs 0x0 - 0x4E1C
    - Calls OpenProcess on each PID
      - If Success store valid PID
    - Else Continue Loop
  - Finished looping, take list of known PIDs and compare it to list generated by calling CreateToolhelp32Snapshot
  - Any differences are hidden processes

RAIDE

- What is RAIDE?
- What makes RAIDE different than Blacklight, RKDetector, Rootkit Revealer, VICE, SVV, SDTRestore?
- What doesn't RAIDE do?

What is RAIDE

- RAIDE is a complete toolkit offering:
  - Forensic Capabilities (RKDetector)
    - Dumping Process
  - Hidden Process Detection (Blacklight)
  - Hook Restoration (SDTRestore, SVV)
  - Hook Detection (SDTRestore, SVV)
  - Memory Subversion Detection
  - Hidden Process Restoration
    - Relink process to make it visible
    - Close Hidden Process (not implemented yet)

What Makes RAIDE Different?

- RAIDE combines most existing tools
  - RAIDE detects Memory Subversion
  - RAIDE does **not** use IOCTL's to communicate

What Doesn't RAIDE Do?

- RAIDE does not detect hidden files, folders, and registry keys
- RAIDE does not restore Driver hooks
- RAIDE does not restore IDT hooks
- RAIDE is not going to keep you Rootkit Free!

RAIDE Communication

- RAIDE uses Shared Memory segments to pass information to the kernel
  - Shared Memory contains only encrypted data
  - Communication uses randomly named events

Hidden Process Detection

- Goal for Process Detection:
  - Signature that can not be zeroed out
  - Signature that is unique
  - Way to verify the signature
  - Signature must not have false positives

Hidden Process Detection

- Signature:
  - Locate pointers to “ServiceTable”
    - ServiceTable = nt!KeServiceDescriptorTableShadow
    - ServiceTable = nt!KeServiceDescriptorTable
  - Contained in all ETHREAD
- Diffing:
  - Spawn a process with random name
    - Process sends a list of processes visible to RAIDE
    - RAIDE diffs the two lists finding the hidden processes

Shadow Walker Detection: Illuminating the Shadows

- Shadow Walker relies on IDT hook
  - Check IDT 0x0e for a hook
    - SW could modify itself to hide the IDT hook
- Other detection schemes out there
  - Remapping

Forensics

- Hook Restoration
- Relinking Processes (DKOM method reversed)
- Dumping process

Hook Restoration

- If an SSDT index hook is detected
  - Open ntoskrnl
  - Obtain KeServiceDescriptorTable from file on disk
  - Obtain original address for hooked index
  - Recalculate address
  - “re-hook” SSDT index with original address

Hook Restoration

- If it is an inline hook:
  - Open ntoskrnl on disk
  - Obtain original function address
  - Read first three instructions
  - Restore first three instructions
    - Can restore more

Relinking Processes

- DKOM is common hiding method
  - DKOM relies on unlinking the EPROCESS link pointers
  - Restore link pointers by passing system eproc and hidden eproc to *InsertTailList*
  - Allows user to see process
- NOTE: Closing the process once visible may blue screen system as the process was not expecting to be closed!

Dumping Process

- Dumping Process
  - Allows Security Analysts to reverse the executable or system file and see what it was doing.
  - Does not matter if the file is originally hidden on the HD
    - Dumped file is renamed and put in working directory
    - Dumping lets analysts bypass any packer protection

DEMO

Thanks

- Peter: bugcheck, skape, pedram, greg h, #nologin and research'ers. And my school for cutting me a break on midterms.
- Jamie:

Questions?