HBGary
Software Security Success

VICE – Catch the hookers!

(Plus new rootkit techniques)

Jamie Butler

Greg Hoglund

Agenda

- Introduction to Rootkits
- Where to Hook
- VICE detection
- Direct Kernel Object Manipulation (DKOM)
  - No hooking required!
- Demonstration of FU rootkit

Operating System Design

- User Land
  - Operating system provides common API for developers to use
    - Kernel32.dll
    - Ntdll.dll
- Kernel Mode
  - The low level kernel functions that implement the services needed in user land
  - Protected memory containing objects such as those for processes, tokens, ports, etc.

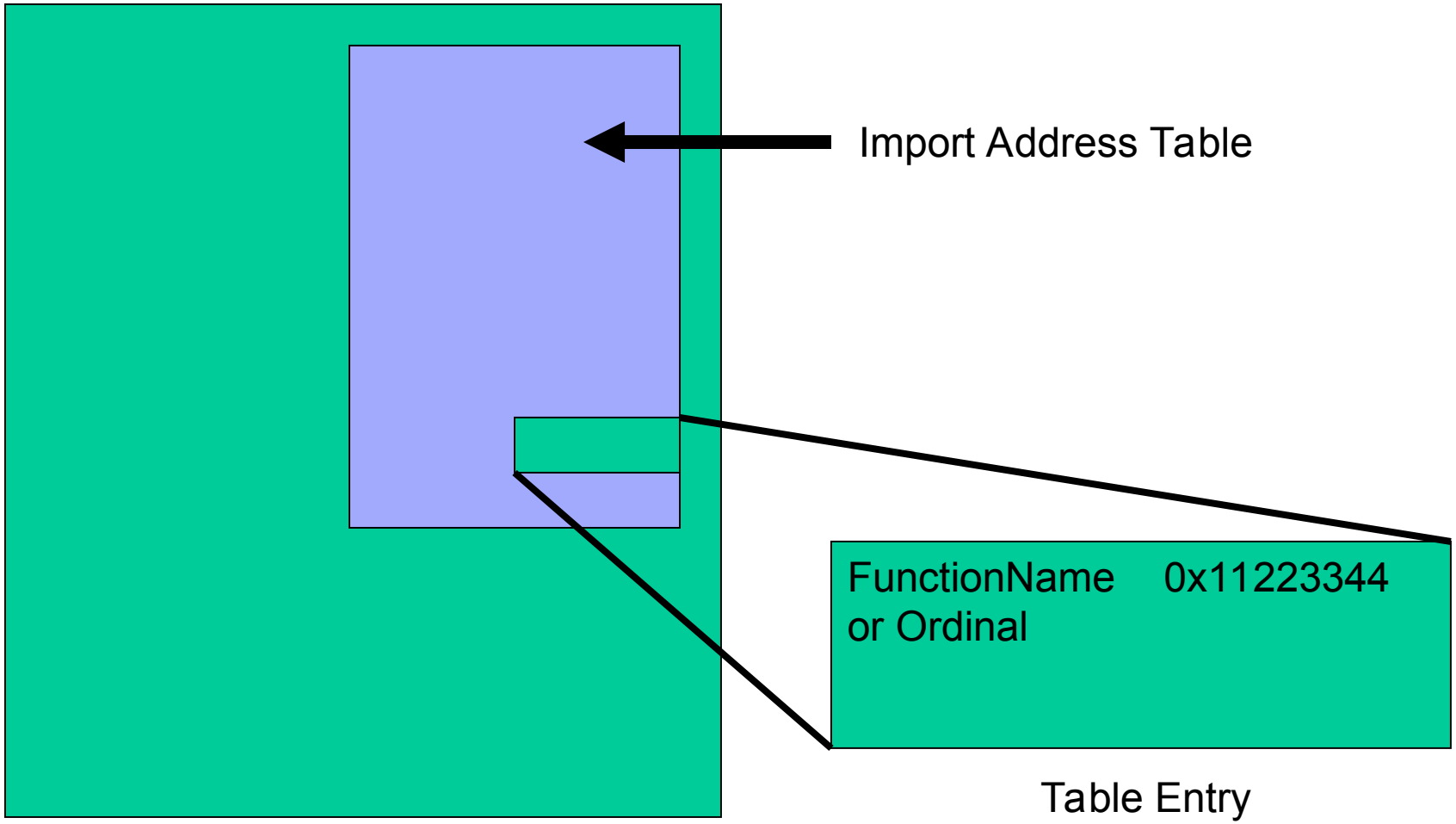
Attack Scenario

- Attacker gains elevated access to computer system
- Attacker installs a Rootkit
- Rootkit's functions
  - Hide processes
  - Hide files
  - Hide network connections
  - Install a backdoor for future access to the system
- Rootkits act as a part of the operating system so they have access to kernel functions.

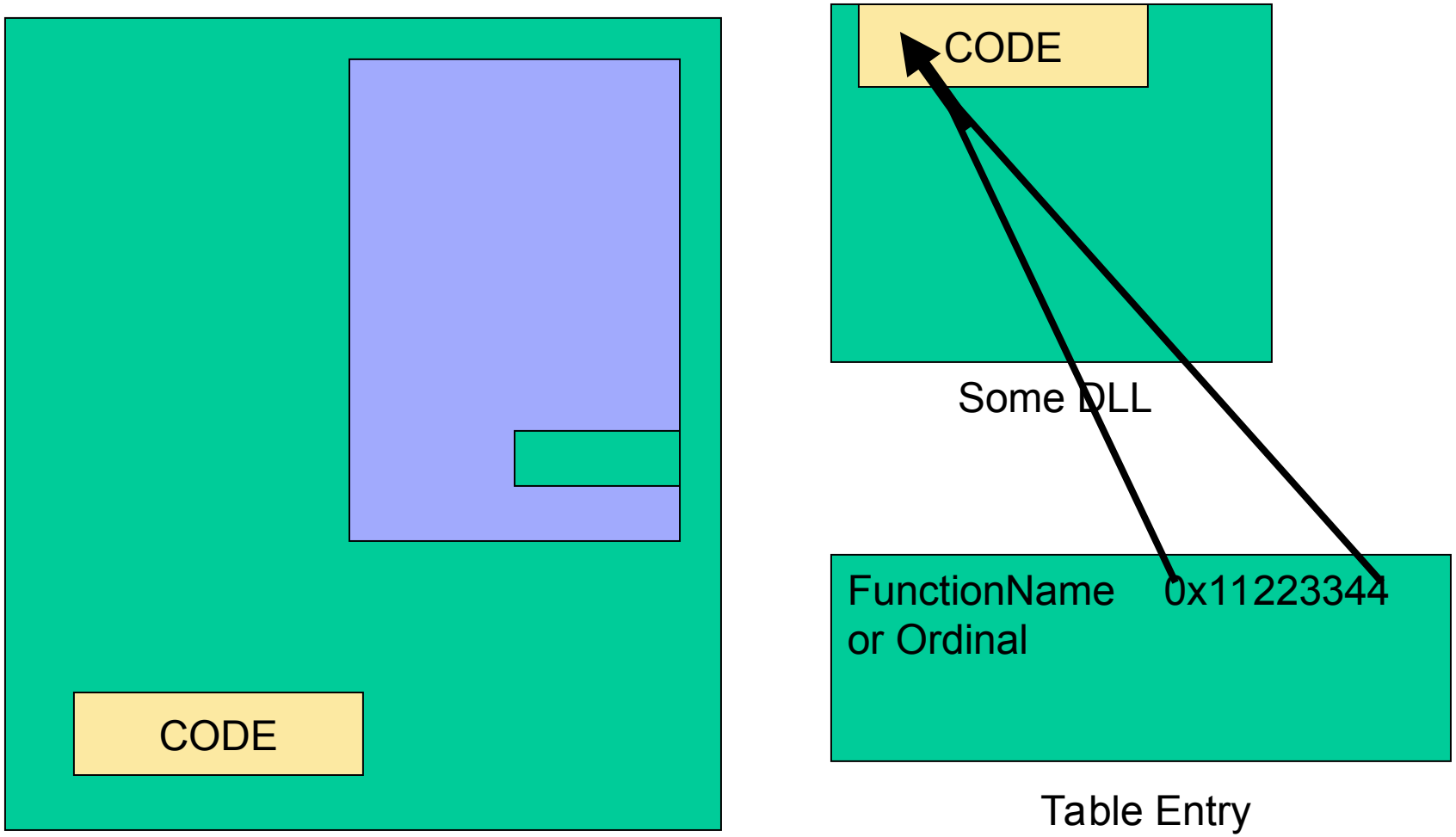
State of Current Rootkits

- Until recently, rootkits were nothing more than Trojan programs such as ps, ls, top, du, and netstat
- Advanced rootkits ***filter*** data
  - Hook the Import Address Table (IAT) in processes
  - Hook the System Call Table of the operating system (the functions exported by the kernel)
  - Hook the Interrupt Descriptor Table (IDT)
    - Interrupts are used to signal to the kernel that it has work to perform.
    - By hooking one interrupt, a clever rootkit can filter all exported kernel functions.

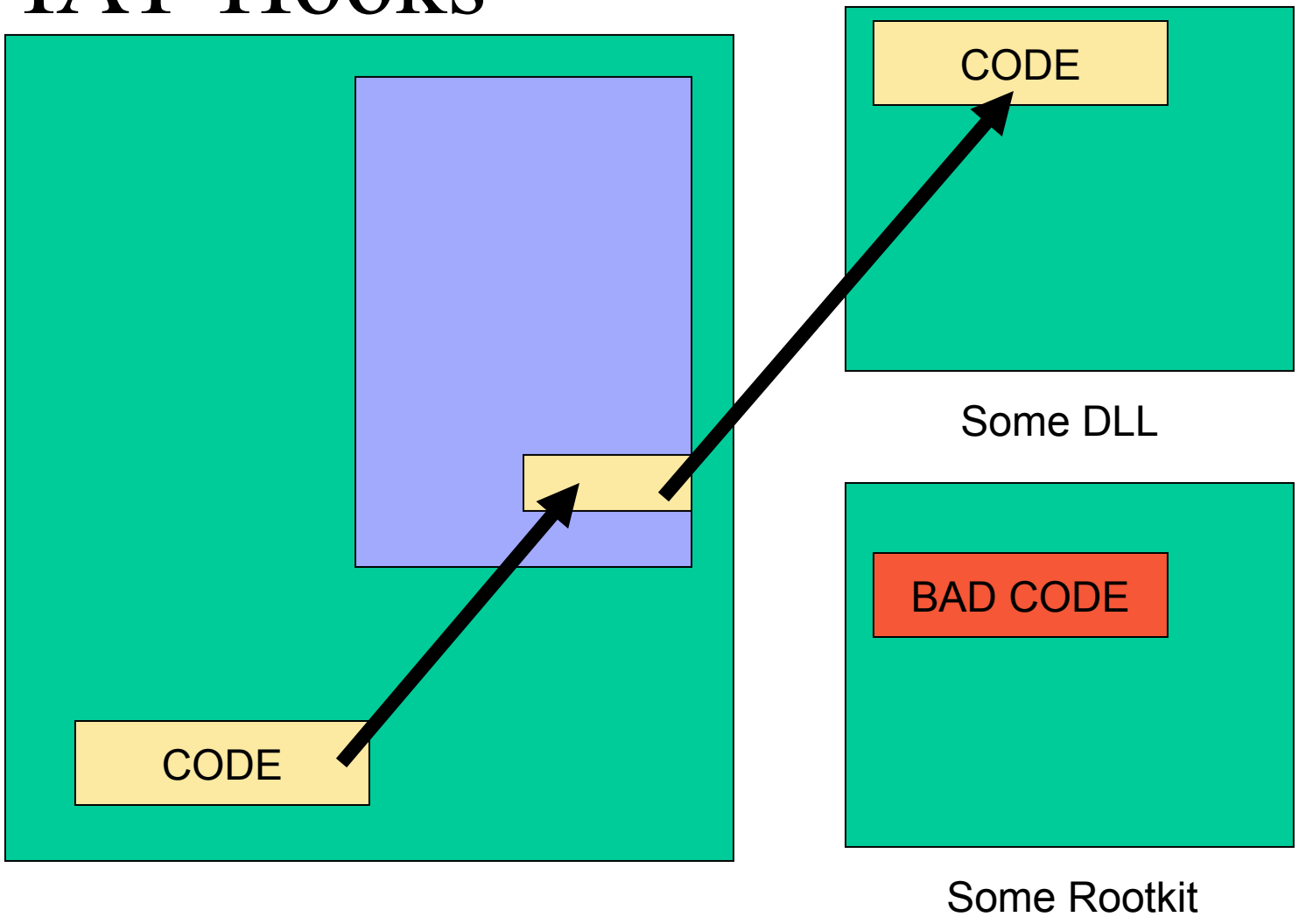
IAT Hooks


Import Address Table
FunctionName 0x11223344
or Ordinal
Table Entry

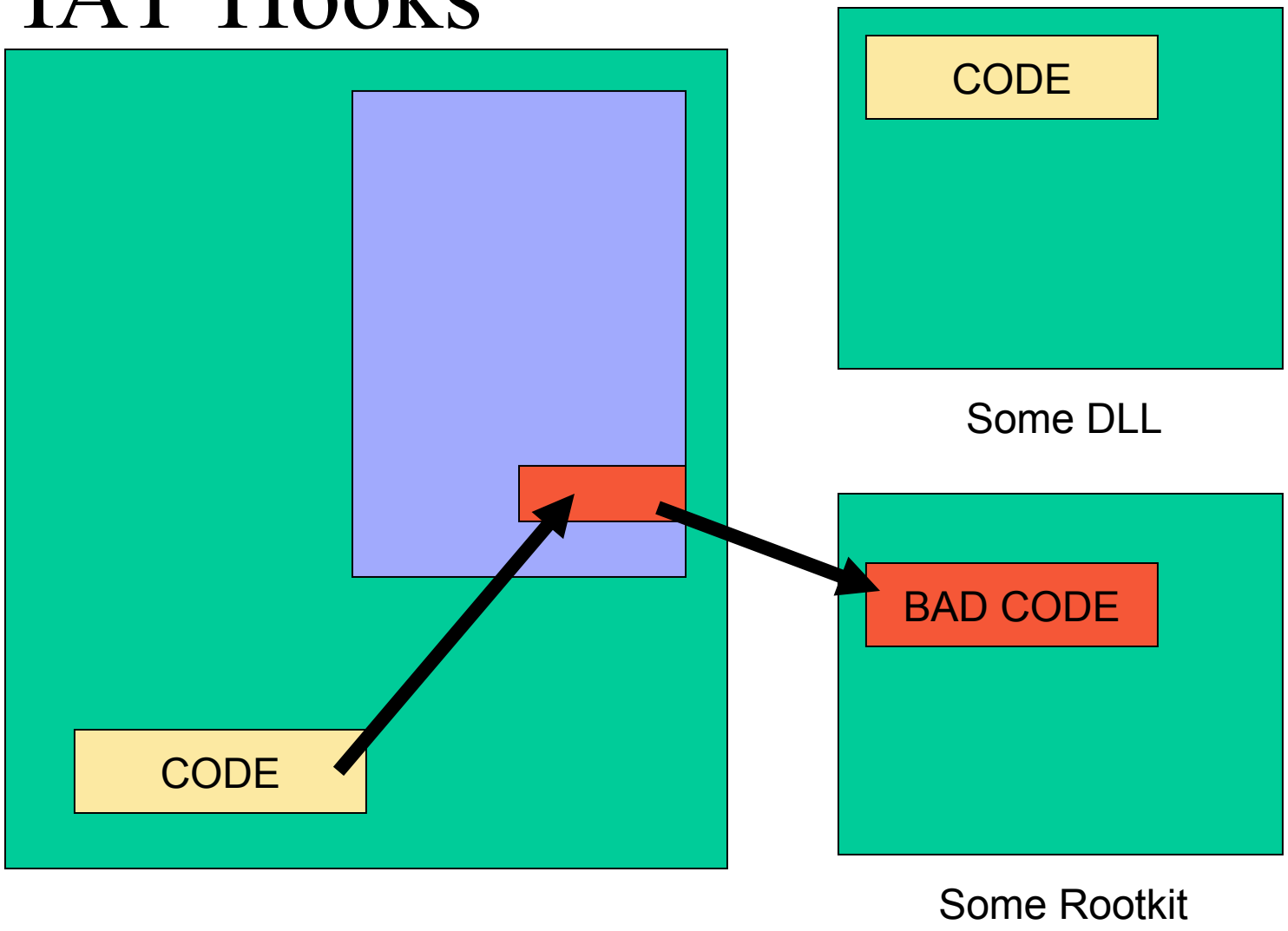
IAT Hooks

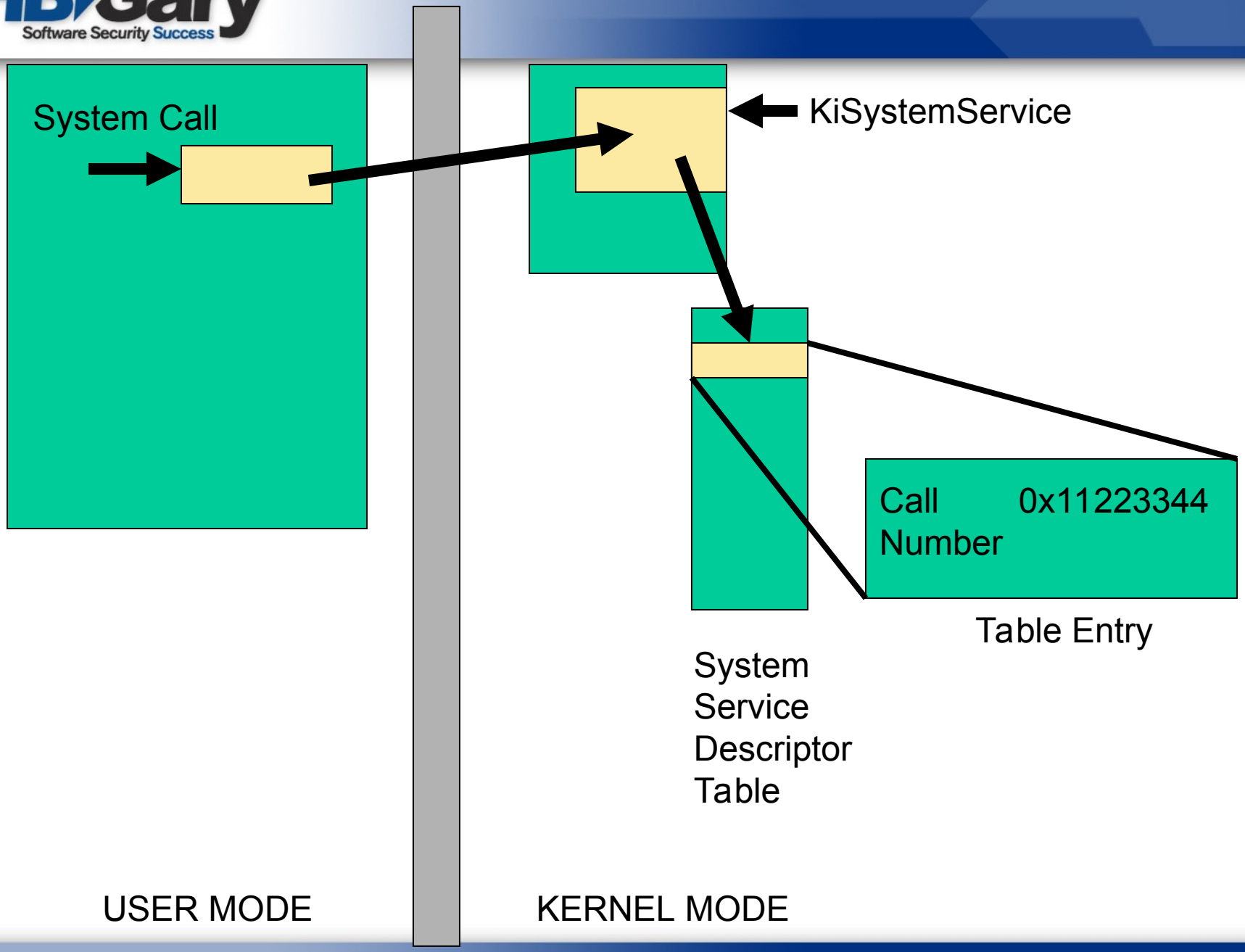

CODE
CODE
Some DLL
FunctionName
or Ordinal
0x11223344
Table Entry

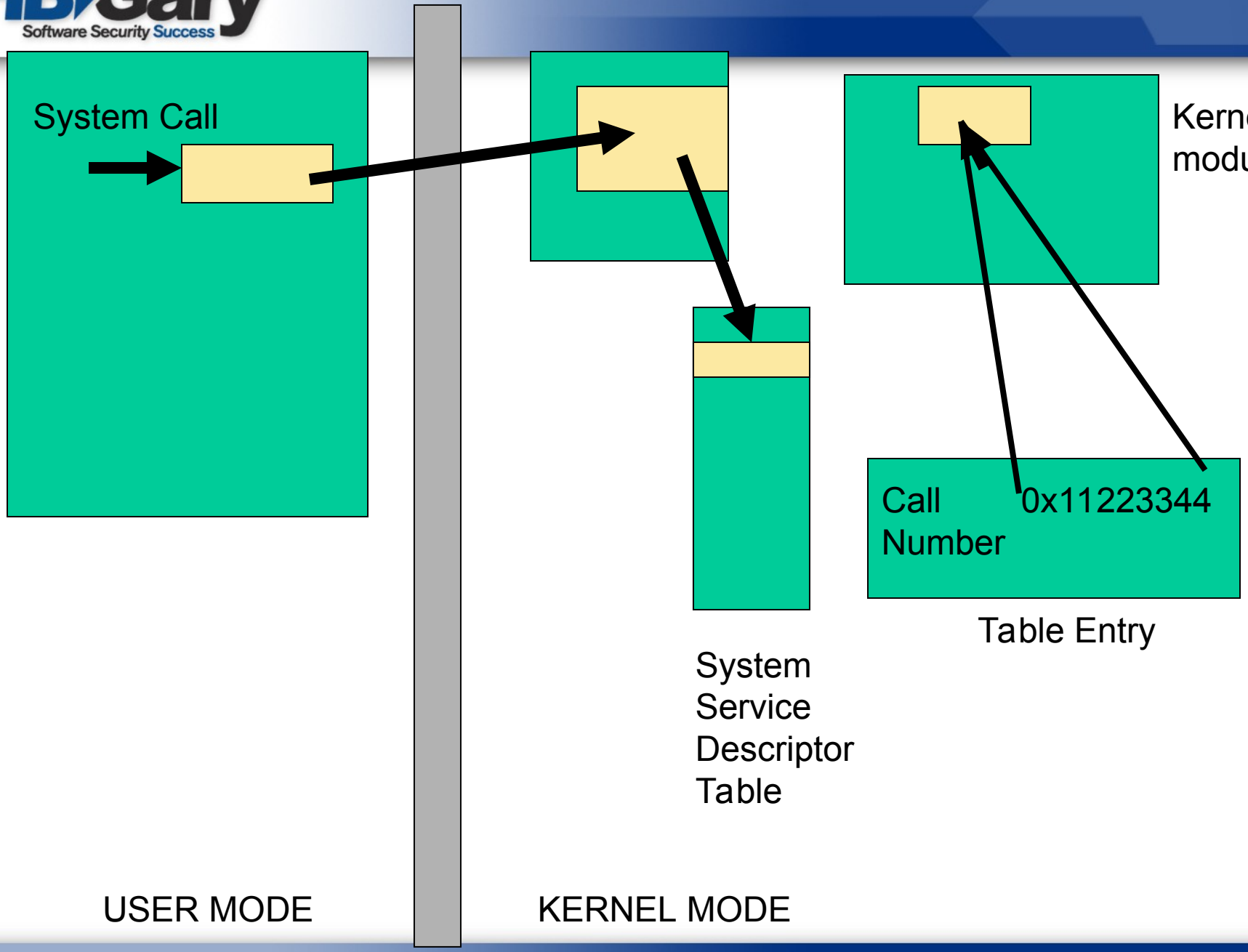
IAT Hooks

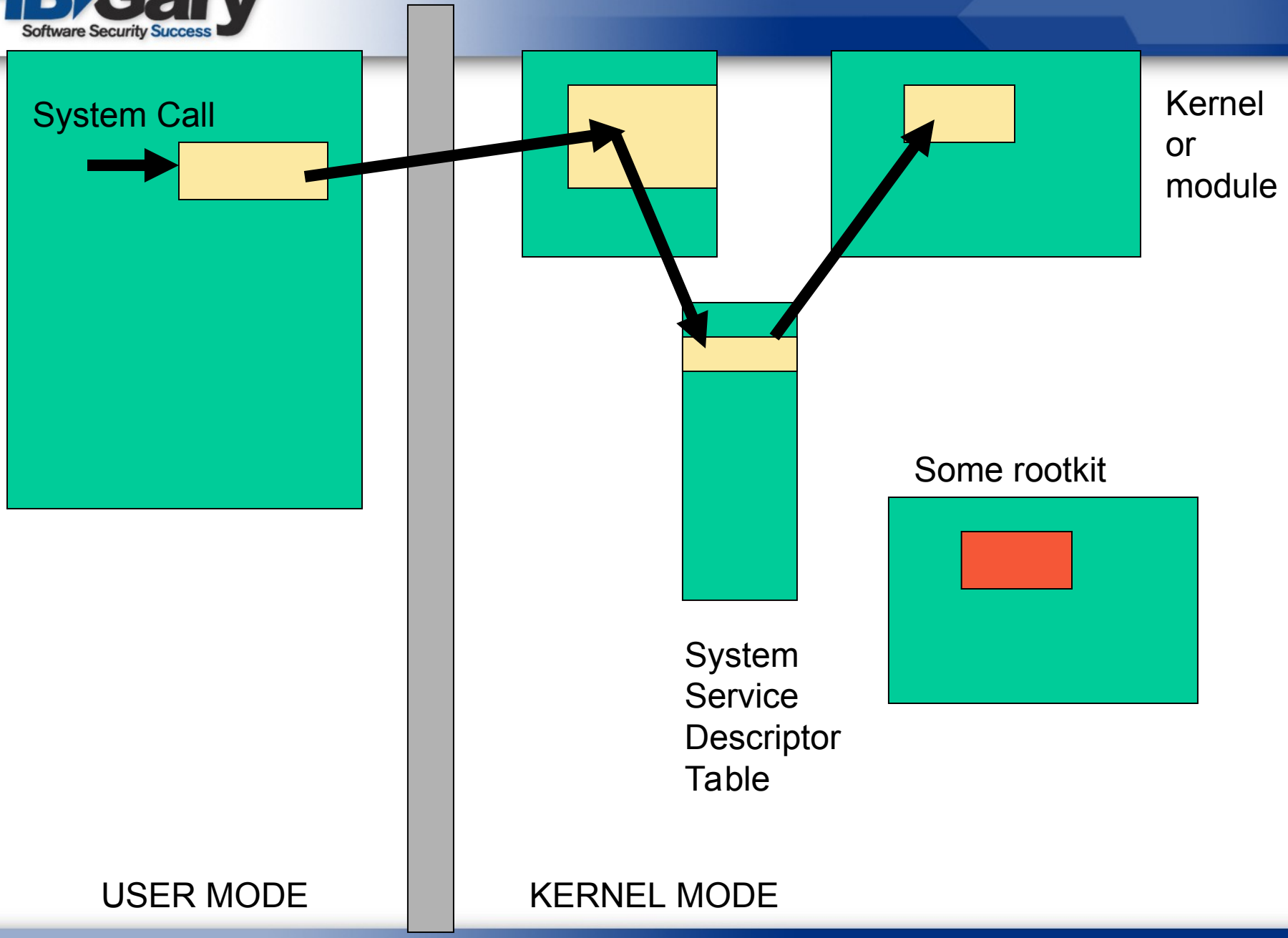


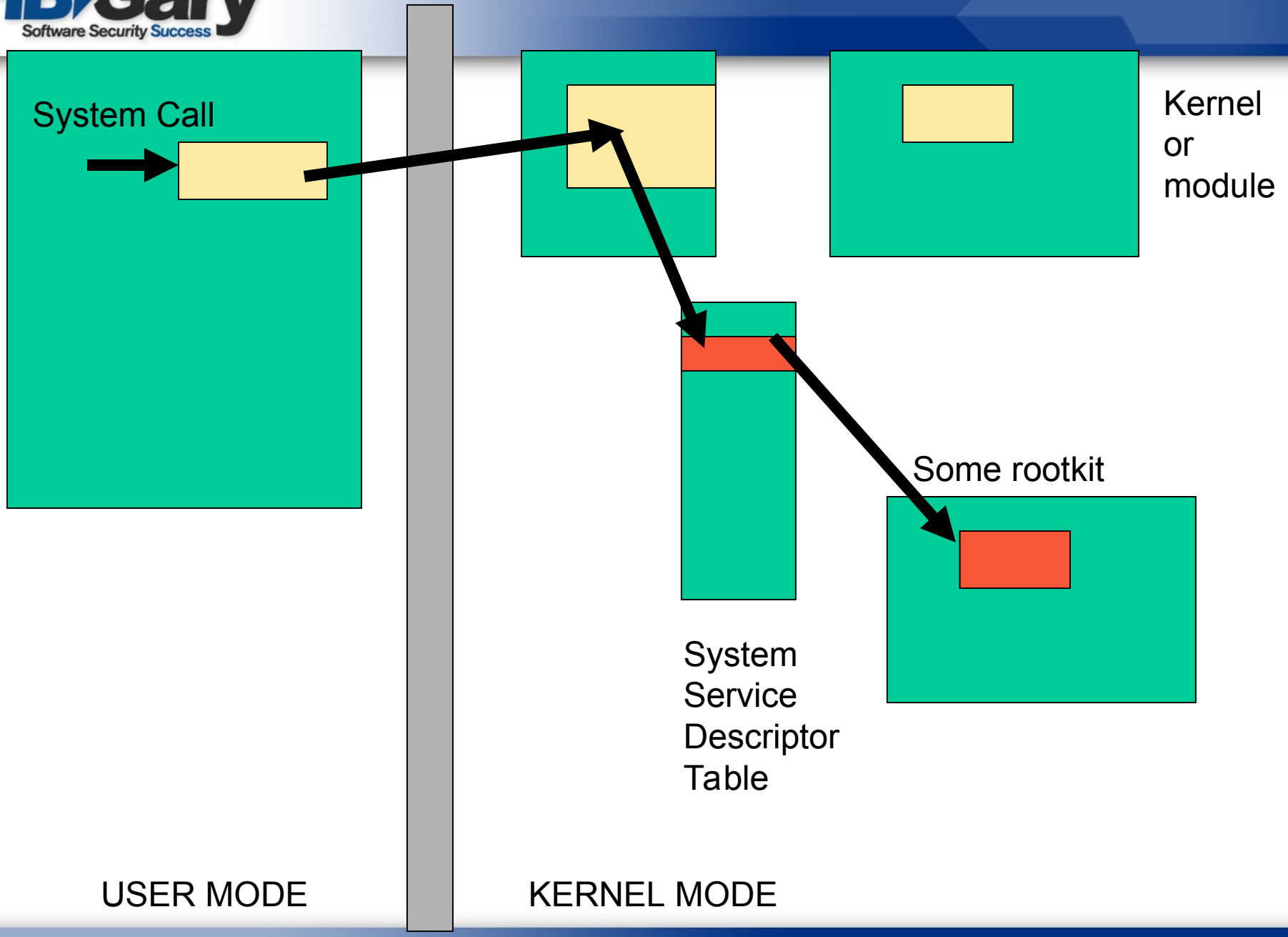
IAT Hooks

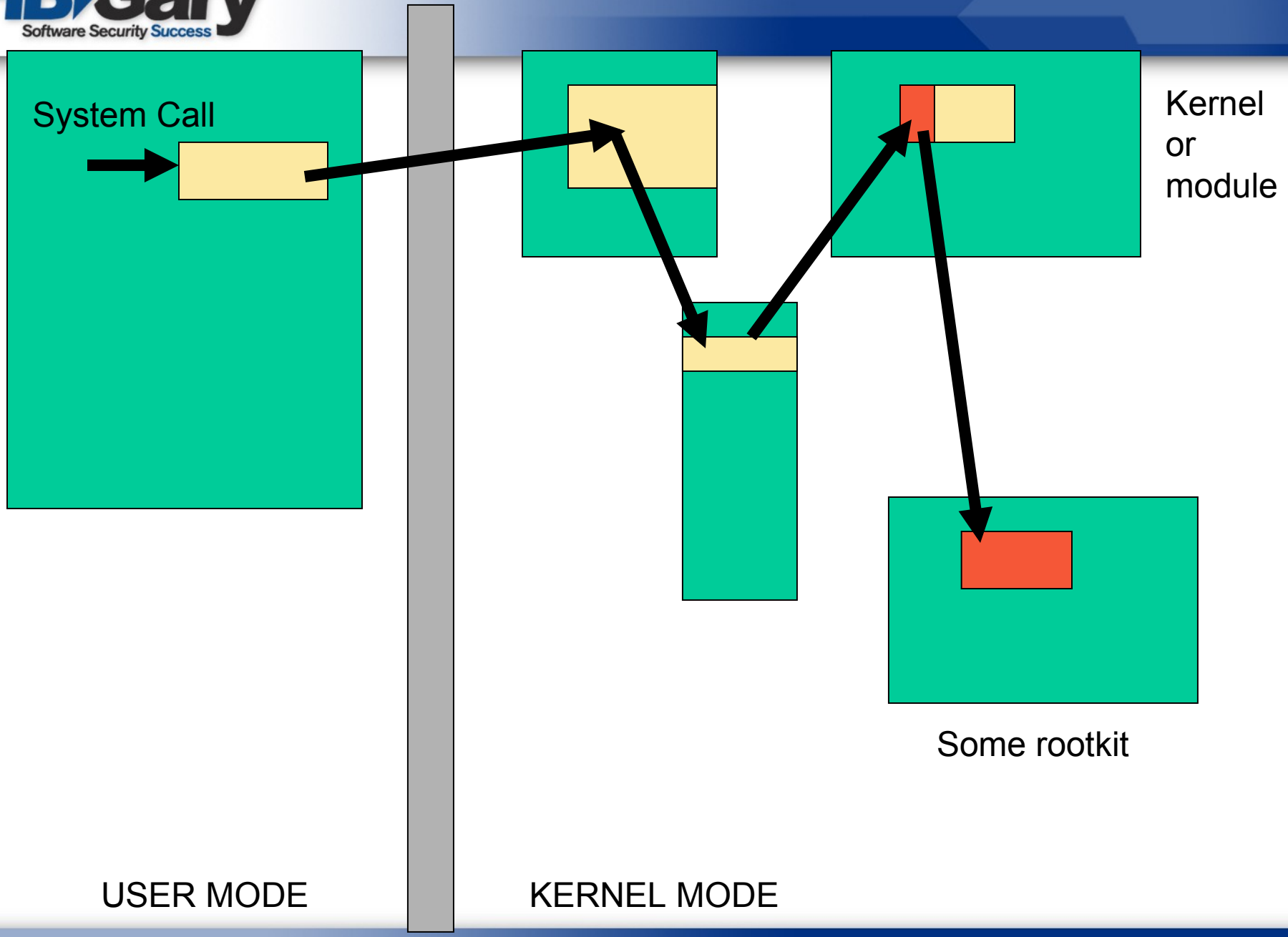



System Call
KiSystemService
Call Number
0x11223344
Table Entry
System Service Descriptor Table
USER MODE
KERNEL MODE

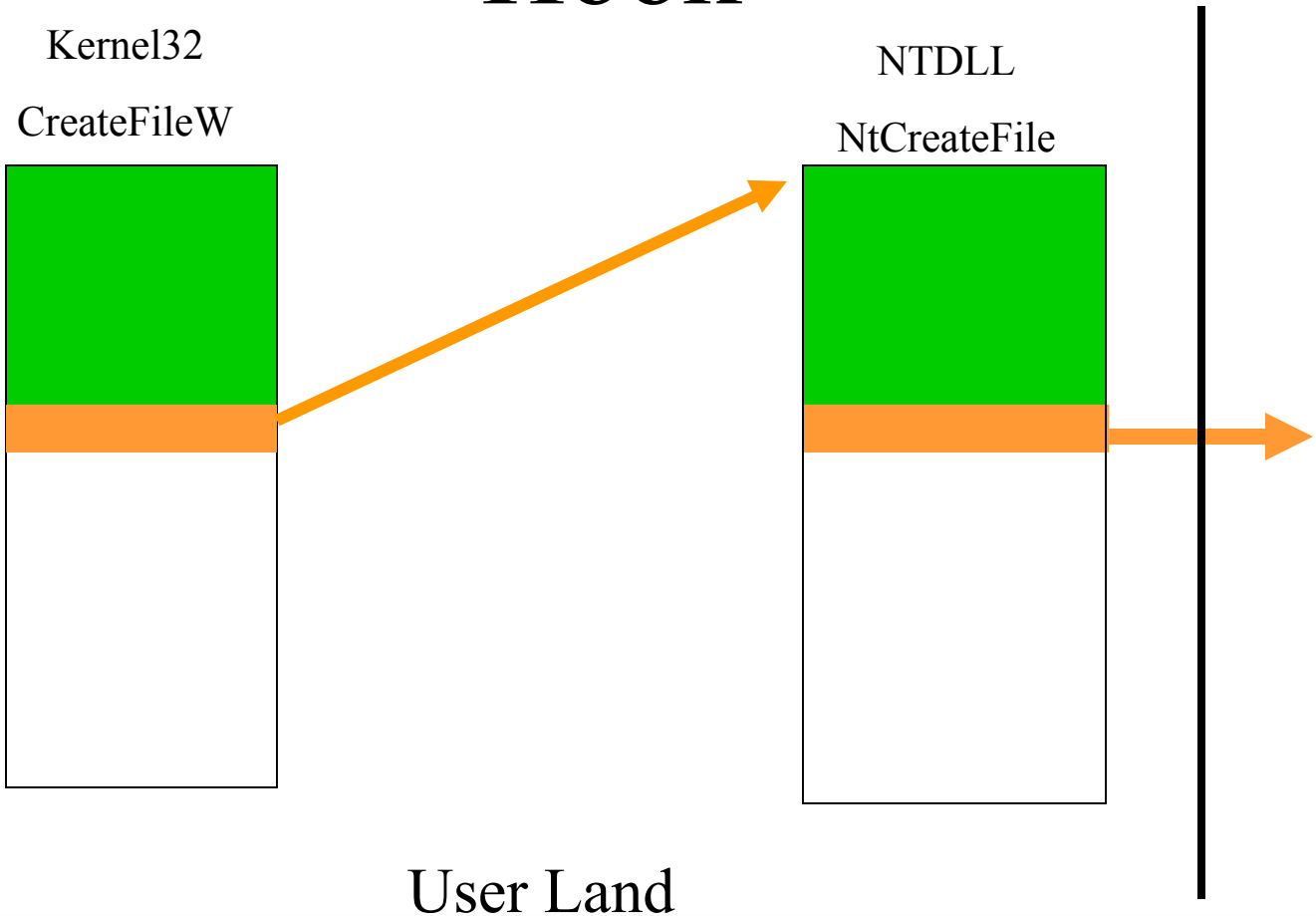

System Call
Kernel or module
Call Number
0x11223344
Table Entry
System Service Descriptor Table
USER MODE
KERNEL MODE


System Call
Kernel
or
module
Some rootkit
System
Service
Descriptor
Table
USER MODE
KERNEL MODE

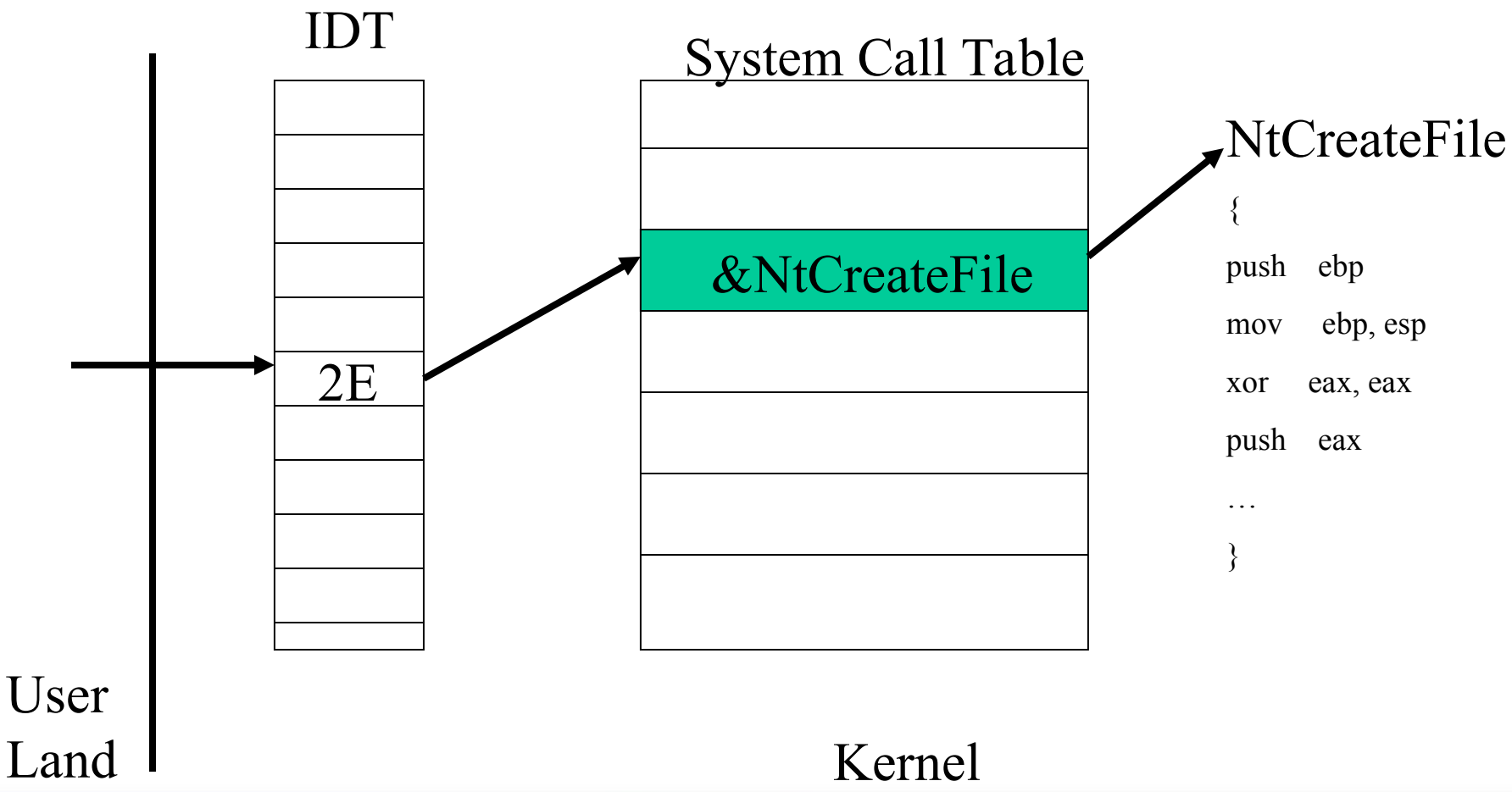

System Call
Kernel
or
module
Some rootkit
System
Service
Descriptor
Table
USER MODE
KERNEL MODE


System Call
Kernel
or
module
Some rootkit
USER MODE
KERNEL MODE

Control Flow … aka Places to Hook



Control Flow … aka Places to Hook



VICE

- VICE is a tool to detect rootkits
  - Designed originally to detect hooks
    - Kernel System Call Hooks
    - Win32 API Hooks
    - In-line function patching
- VICE has an API so other it can be easily incorporated into other tools

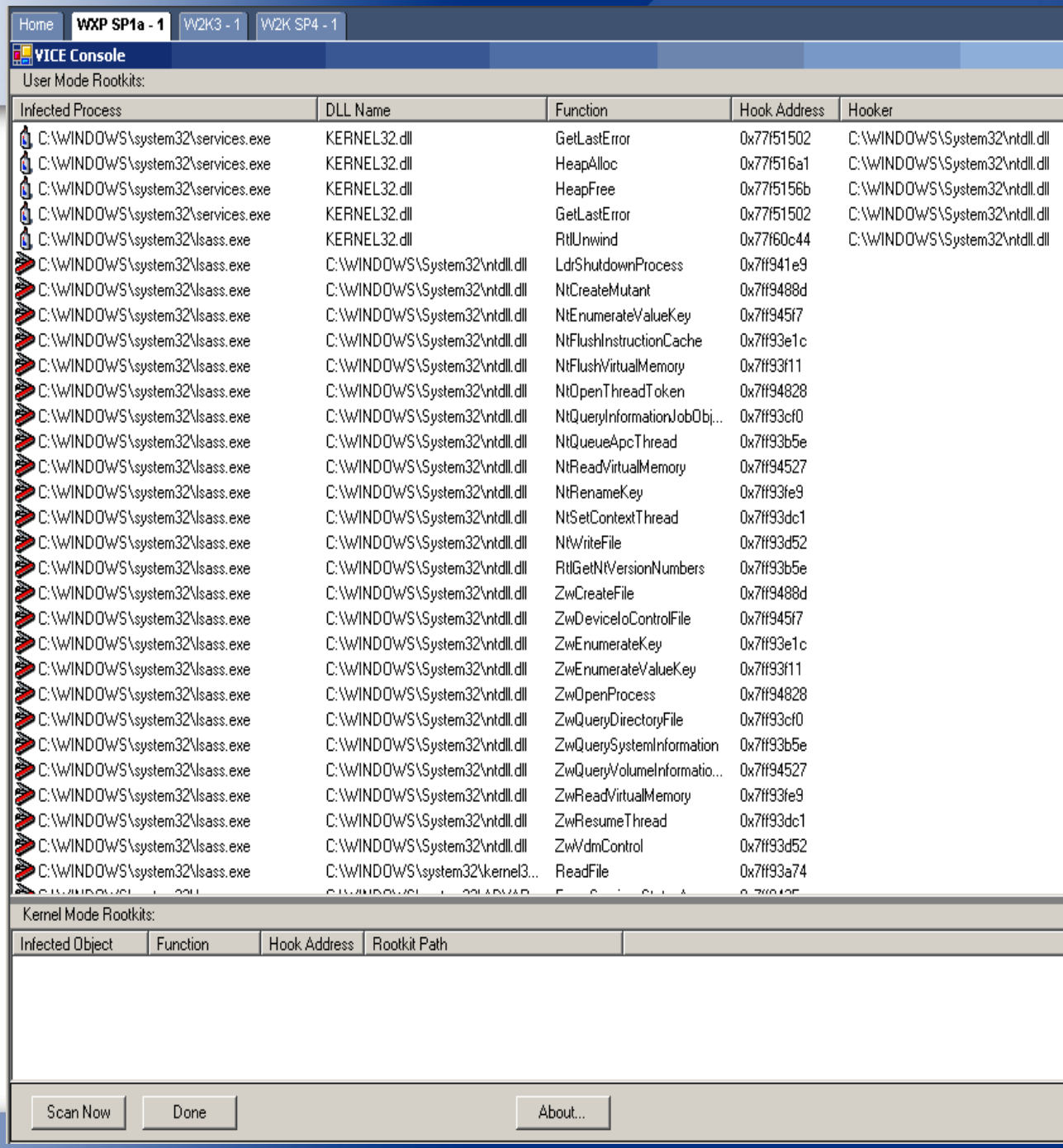
VICE Demonstrations

HBGary
Software Security Success

- VICE

vs

hxdef rootkit

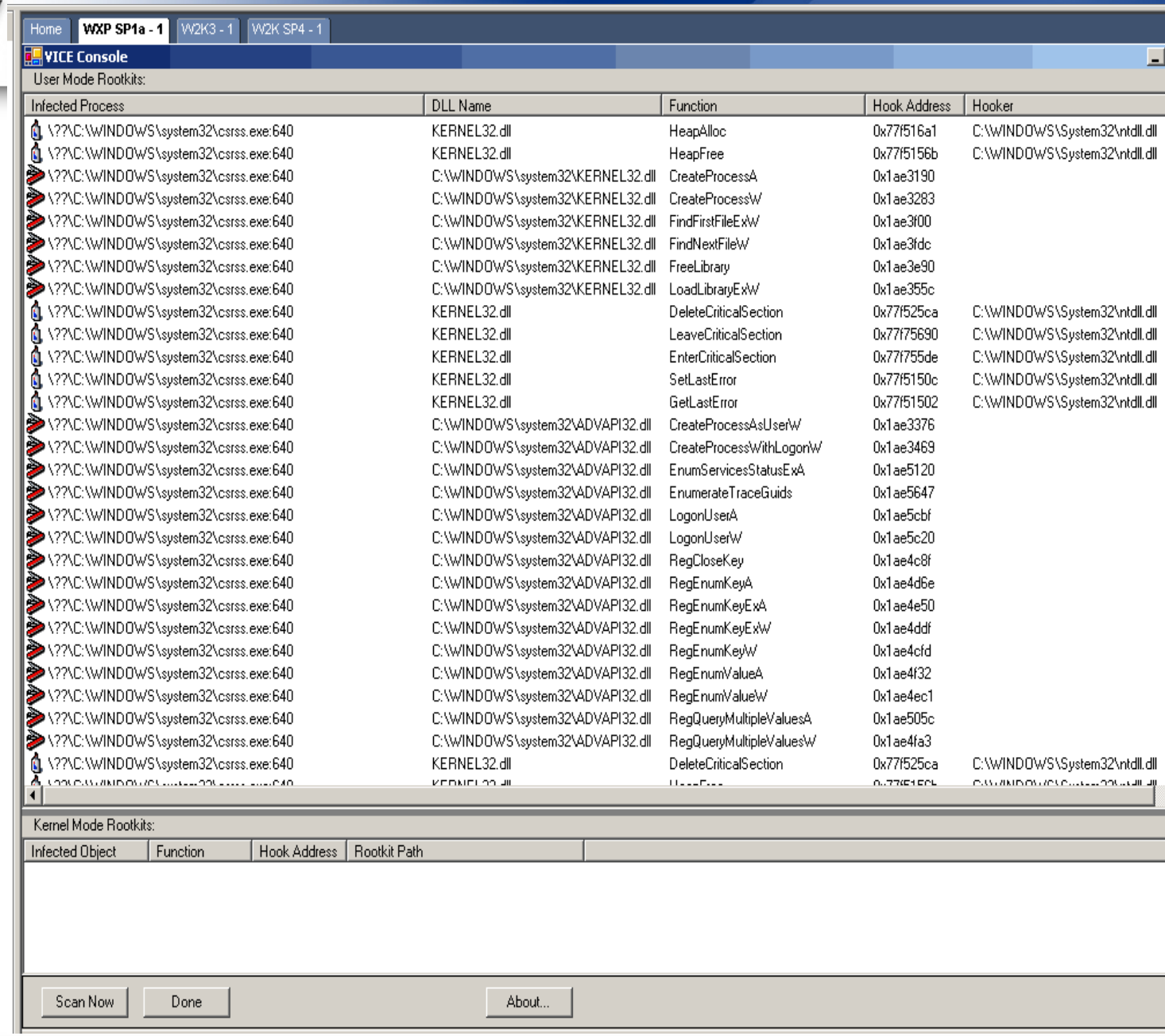


HBGary
Software Security Success

- VICE

vs

Vanquish rootkit



Consumers demand more…

- Corporations and many private consumers see the need for more security
  - Personal firewalls
  - Host based intrusion prevention systems

Current HIDS/HIPS Functions

- To detect or prevent:
  - Processes running
  - Files that are created/deleted/modified
  - Network connections made
  - Privilege escalation
- Trusts the operating system to report these activities.
- If the underlying operating system is compromised, the HIDS/HIPS fails.

What Makes HIDS/HIPS Possible?

- Querying kernel reporting functions
- Hooking user land API functions
  - Kernel32.dll
  - Ntdll.dll
- Hooking the System Call Table
- Registering OS provided call-back functions

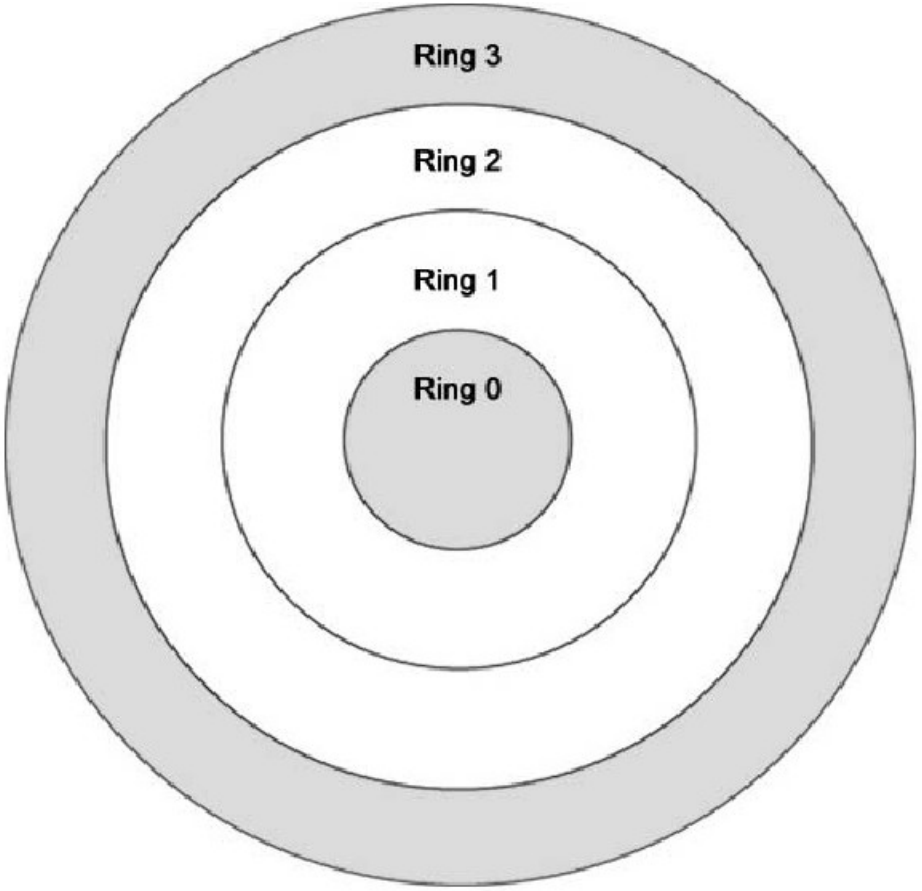
Problems with HIPS Design

- Required to be on the execution path of the attacker to allow/deny actions
- Looks just like the hooks rootkits use (dual technology)

- … **who said an attacker has to use an API**

Operating System Design

- Intel has four privilege levels or rings
- Microsoft and many other OS vendors use only two rings



Operating System Design

- By only using two privilege levels, there is no separation between the kernel itself and third party drivers or loadable kernel modules (LKM's)
- Drivers can modify the memory associated with kernel objects such as those that represent a process's token

Next Generation Rootkit Techniques

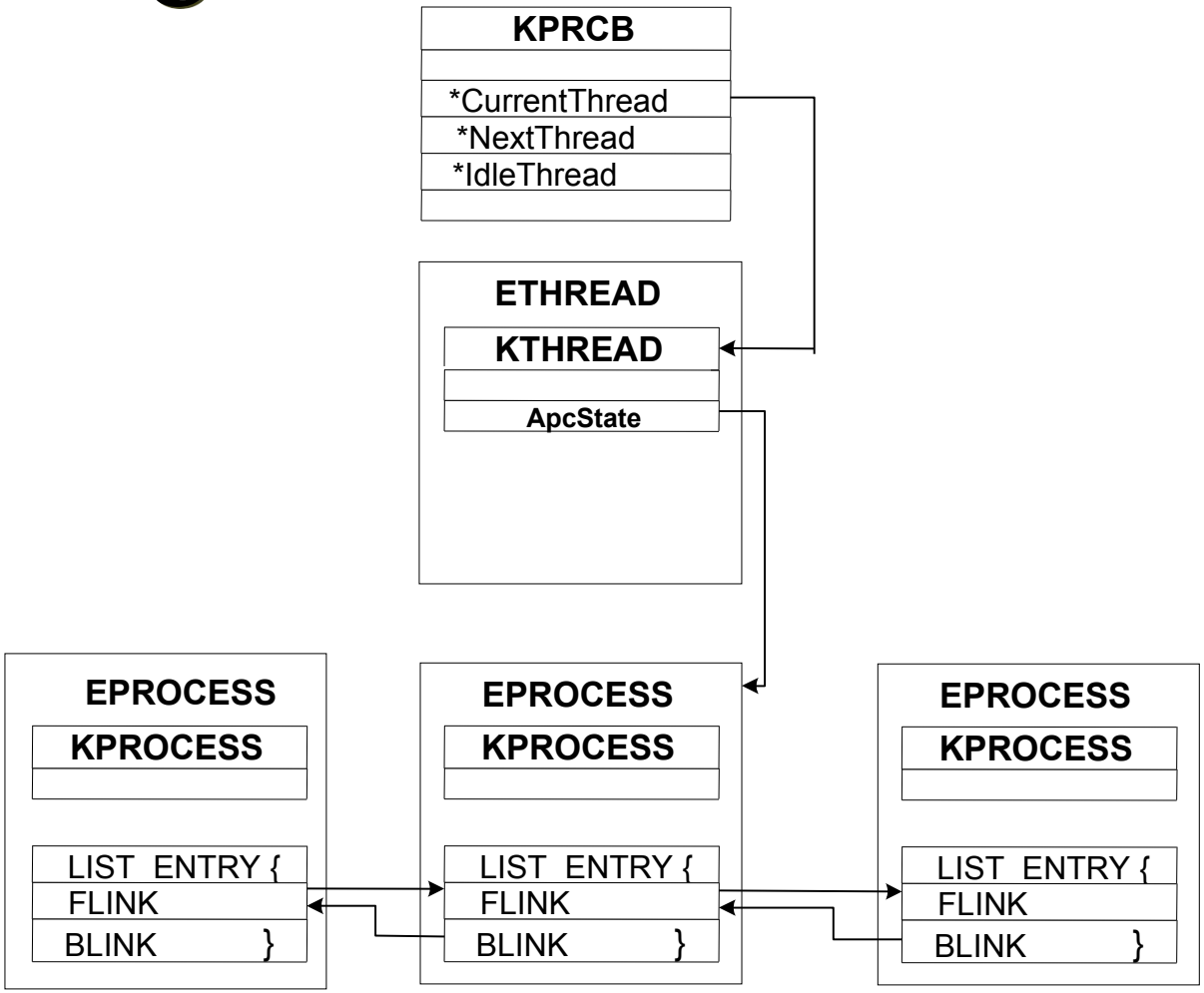
- **Direct Kernel Object Manipulation (DKOM) in memory**
  - A device driver or loadable kernel module has access to kernel memory
  - A sophisticated rootkit can modify the objects directly in memory in a relatively reliable fashion to hide.
  - Recall the goal of rootkits is to hide things: processes, files, and network connections.

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

The Implication of Hidden Processes

- The intruder has full control of the system.
- Defeats a Host Based IDS/IPS that depends upon the underlying operating system.
- Will skew the results of forensic examinations.

Hiding Processes - Windows



Hiding Processes - Windows

- Locate the Processor Control Block (KPRCB)
  - Located at 0xffdff120
  - fs register in kernel mode points to 0xffdff000
- Within the KPRCB is a pointer to the Current Thread block (ETHREAD)
  - Located at fs:[124] or 0xffdff124
  - An ETHREAD contains a KTHREAD structure

Hiding Processes - Windows

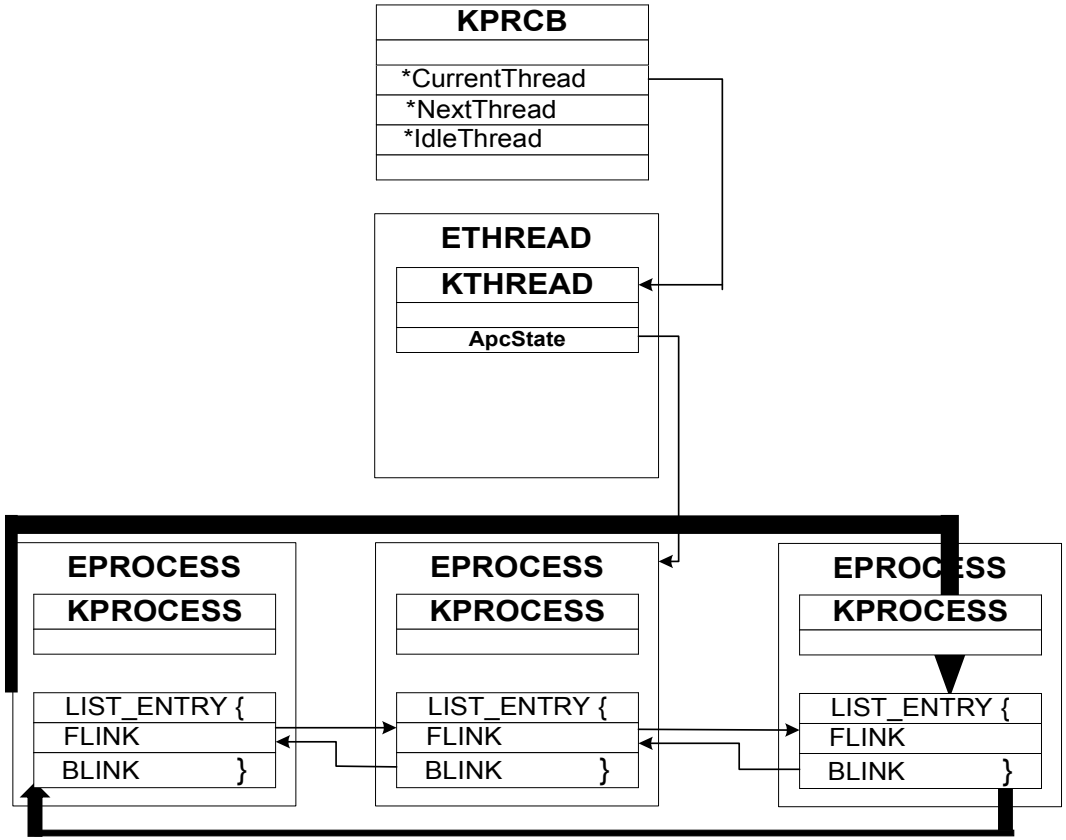
- The KTHREAD structure contains a pointer to the EPROCESS block of the current process
- The EPROCESS block contains a LIST structure, which has a forward and backward pointer to active processes
  - This creates the doubly linked list of active processes in Windows

Hiding Processes - Windows

- To hide a process
  - Locate the EPROCESS block of the process to hide
  - Change the process behind it to point to the process after the process you are hiding
  - Change the process after it to point to the process before the one you are trying to hide

Essentially, the list of active now processes points "around" the hidden process

Hiding Processes - Windows



Hiding Processes - Windows

- Why does the process continue to run?
  - Scheduling in the Windows kernel is thread based and not process based.
- Although scheduling code to run is based upon threads, when the kernel reports what is running on the system, it reports based upon EPROCESS blocks which can be modified with no adverse affect. This is what current tools (IDS/IPS's) rely upon to discover what is running on the system.

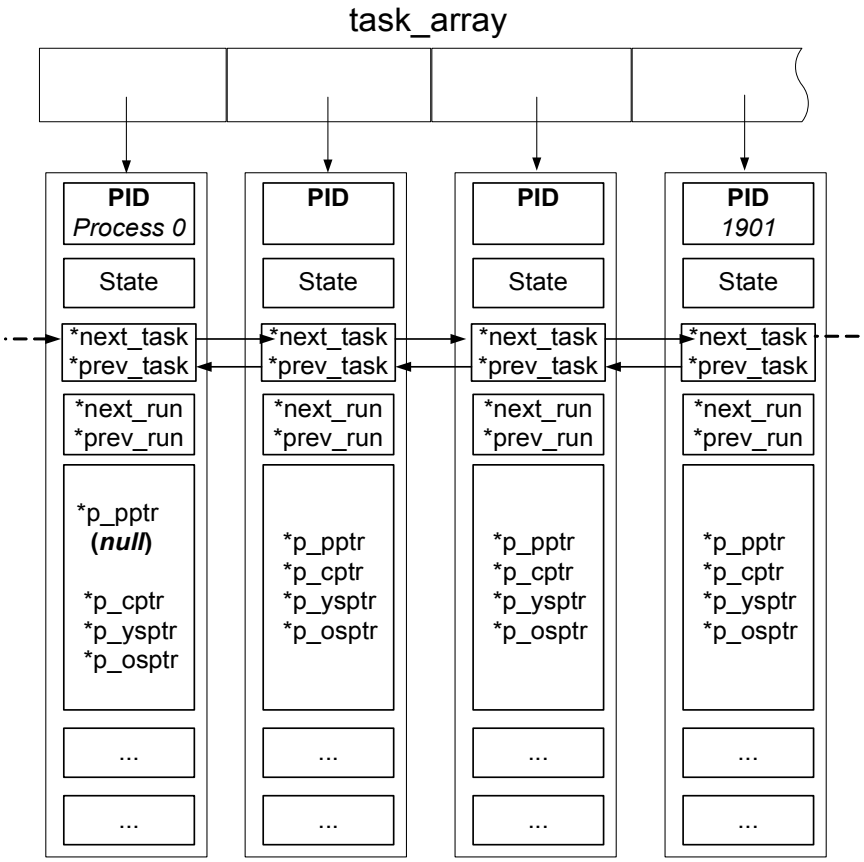
Hiding Processes – LINUX

- The LINUX kernel contains an array of task_struct's.
- A task_struct is similar to an EPROCESS block in Windows
- task_struct contains pointers to the prev_task and next_task
- task_struct also contains pointers to the prev_run and next_run for the running processes

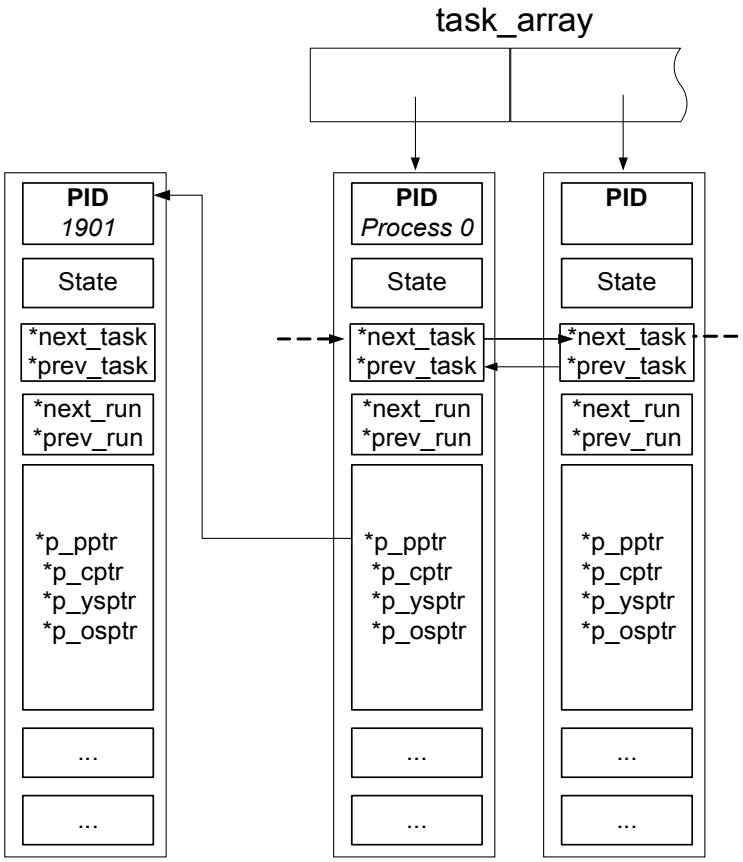
Hiding Processes – LINUX

- To hide a process, remove the process from the list of prev_task and next_task
- Leave next_run and prev_run alone

Hiding Processes - LINUX



Hiding Processes – LINUX



Hiding Processes - LINUX

- To prevent the process from freezing
  - The LINUX scheduler walks the list of task_struct's to calculate the goodness value of the process to decide rather to schedule it or not.
  - The LINUX scheduler must be modified to allocate time quantums to the parent of process of PID 0

Synchronization Issues

- Modifying shared objects such as the active process list is not completely safe.
  - Rootkit could be swapped out
  - Multiprocessor issues
- In Windows, the list of active processes is protected by PspActiveProcessMutex.
- PsLoadedModuleResource guards the list of device drivers.

Synchronization Issues

- Problem: These symbols are not exported by the operating system
- Need a way to find these and other symbols
  - Hardcoding addresses – very unreliable
  - Search for patterns in memory
    - Functions within the kernel use PspActiveProcessMutex
    - Find the mutex's use within functions with a relatively consistent pattern.

Synchronization Research and Code done by Sherri Sparks from the University of Central Florida.

Token Manipulation

- Add Privileges to Token
- Add Groups to Token
- Make the Owner of the Token Any User
- Make Any Actions Taken by the Process Appear to be Someone else such as System
  - Makes forensics difficult
  - Totally fakes out the Windows Event Viewer

Tokens

- Static Part
  - TOKEN SOURCE
  - TokenId
  - AuthenticationId
  - ParentTokenId
  - ExpirationTime
  - TokenLock
  - ModifiedId
  - SessionId
  - UserAndGroupCount
  - RestrictedSidCount
  - PrivilegeCount
  - VariableLength
  - Etc…

Tokens

- Variable Part
  - Privileges
    - LUID
    - Attribute
  - User and Groups
    - Pointer to SID
    - Attribute
  - Restricted SID's
    - Pointer to SID
    - Attribute

Manipulating Tokens

- Difficult to just grow the token because you are not sure what is after the variable part in memory
- Although static portion has pointers to the privileges and groups, just changing these to point to newly allocated memory does not work due to crazy math in a SepDuplicateToken() function

Manipulating Tokens

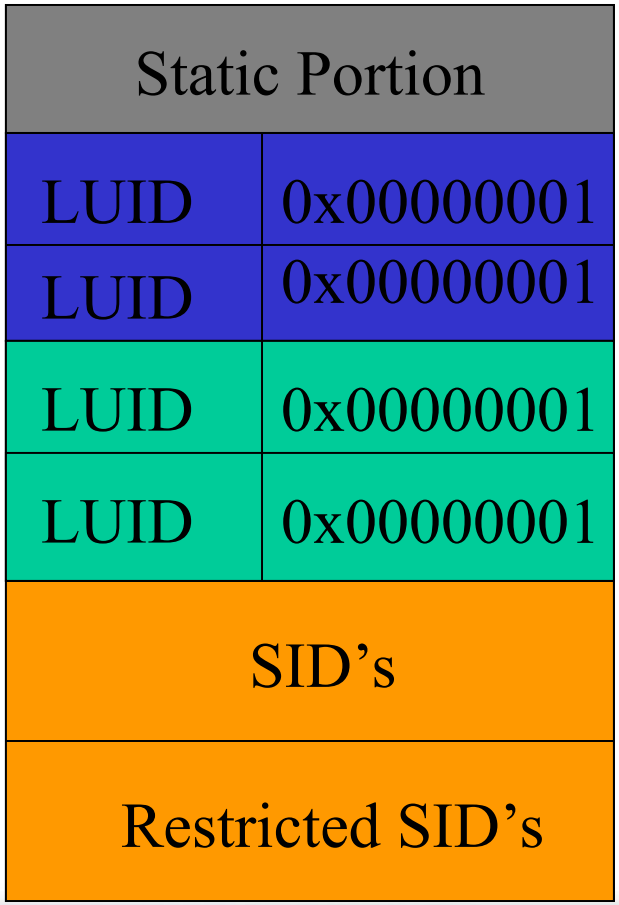
- There are a lot of Privileges in a token that are disabled
- We can discard these since they are disabled anyway and free up space for new privileges and groups
  - The “in-line” method

Adding Privileges to Tokens with DKOM

- Typedef struct _LUID_AND_ATTRIBUTES{

```
        DWORD Luid;
        DWORD Attributes;
}
```

Adding Privileges to Tokens with DKOM

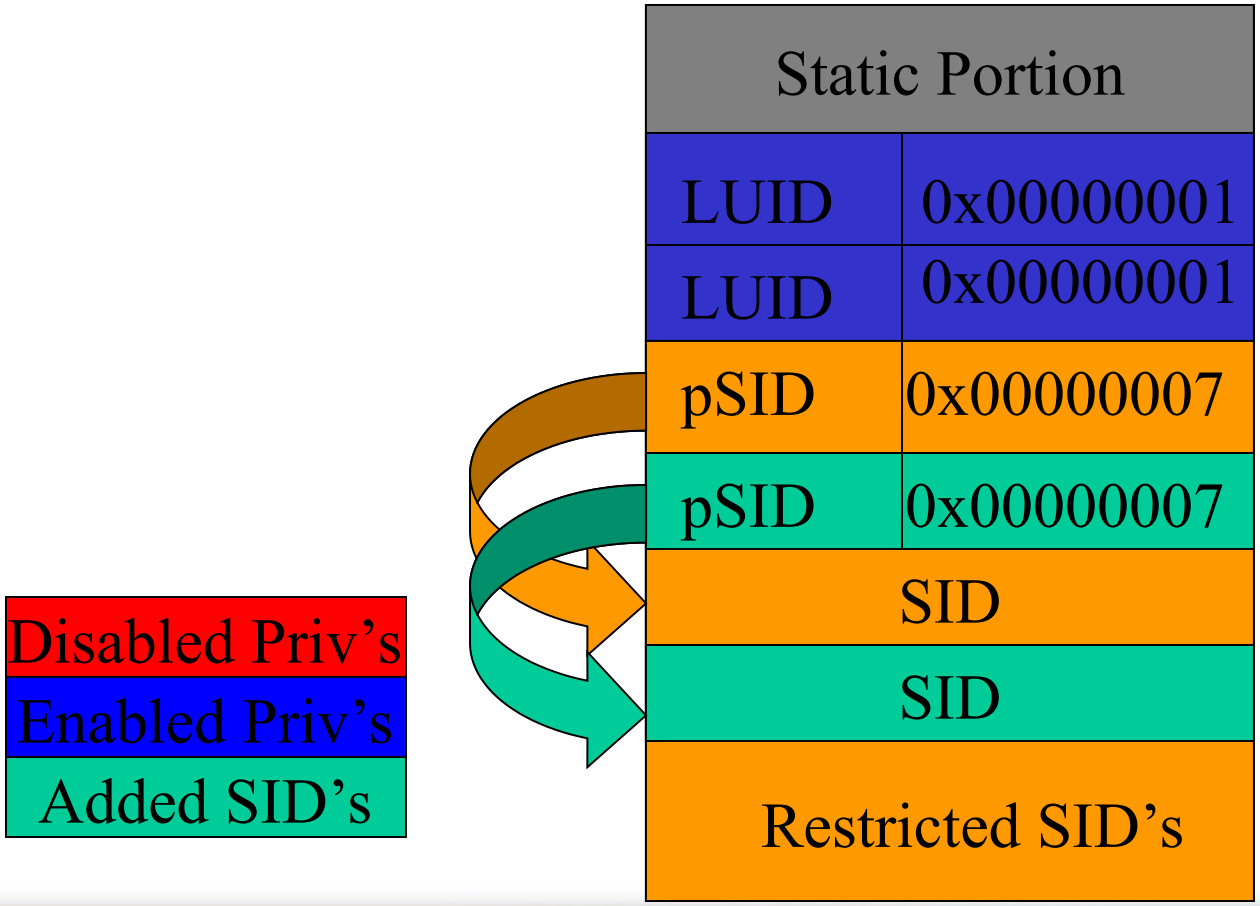


Disabled Priv's
Enabled Priv's
Added Priv's

Adding Groups to Tokens with DKOM

- Typedef struct _SID_AND_ATTRIBUTES
  {
    DWORD pSID;
    DWORD Attributes;
}

Adding Groups to Tokens with DKOM



Faking Out the Windows Event Viewer using DKOM

- Change one DWORD in Static Portion of Token
  - SYSTEM_LUID = 0x000003E7
- Make FIRST SID in Token the System SID
- All logging of the Process now appears as System
- Useful if Detailed Process Tracking is Enabled

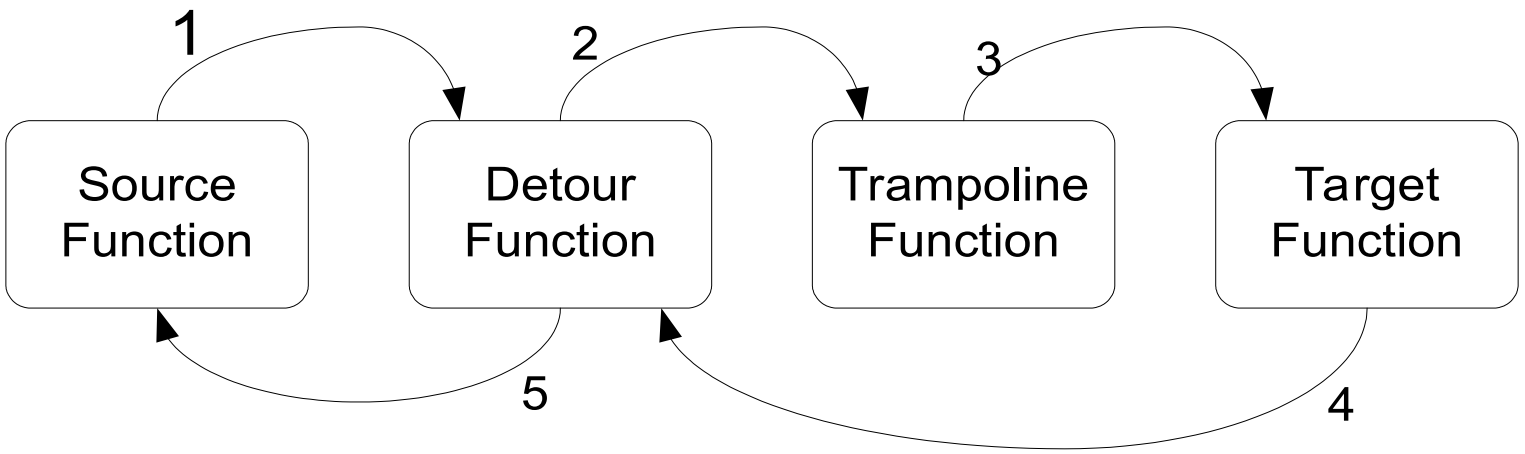
Detecting Hidden Processes in Windows

- Methodology: Examine each thread to ensure its corresponding process descriptor (EPROCESS) is appropriately linked.
- This requires patching the kernel in memory, in particular the SwapContext function.
- Hunt and Brubacher introduced Detours for intercepting Win32 binary functions.

Detours

- Overwrite beginning of target function (SwapContext) with an unconditional jump to a Detour function
- Detour function eventually calls a Trampoline function
- The Trampoline function contains the overwritten bytes of the target (SwapContext) function and calls the target (SwapContext) function
- The Target function returns to the Detour function
- The Detour function returns to the source caller (kernel dispatcher)

Detours


1
2
3
Source Function
Detour Function
Trampoline Function
Target Function
5
4

Patching the Windows kernel

- SwapContext function does context switching between threads in Windows
- Overwrite the first seven bytes of SwapContext with a jump to our Detour function
- The EDI register points to the KTHREAD of the thread to be scheduled to run
- Our Detour function follows the KTHREAD to the EPROCESS block and determines if it is still appropriately linked in the list of active processes.

Other Ways to Detect Hidden Processes

- Klister by Joanna Rutkowska
  - Presented at Black Hat Las Vegas 2003
  - Looks at Thread Queues since threads must be in one of four queues to be scheduled
  - Problem: Queue addresses are not exported so the addresses must be hard coded for each version of the OS

Detecting Hidden Processes in LINUX

- Injectso is a library similar to Detours except for LINUX
- When process state is Task_Running and it is placed in the LINUX run queue by setting the prev_run and next_run pointers appropriately, make sure it is properly linked by testing the next_task and prev_task of its neighbors.

Tool Demonstration: Process Hiding

Tool Demonstration: Gaining System Privilege

Conclusion

- We have shown the evolution of rootkit technology and detection
  - No longer trojanized programs
  - No longer just hooking, which VICE detects
  - Now act as a part of the Trusted Computing Base (TCB)
  - DKOM … what will it be used for next?

Questions?

Thank you.

Email: james.butler@hbgary.com

Attend the Black Hat Training
"Aspects of Offensive Root-kit Technology"