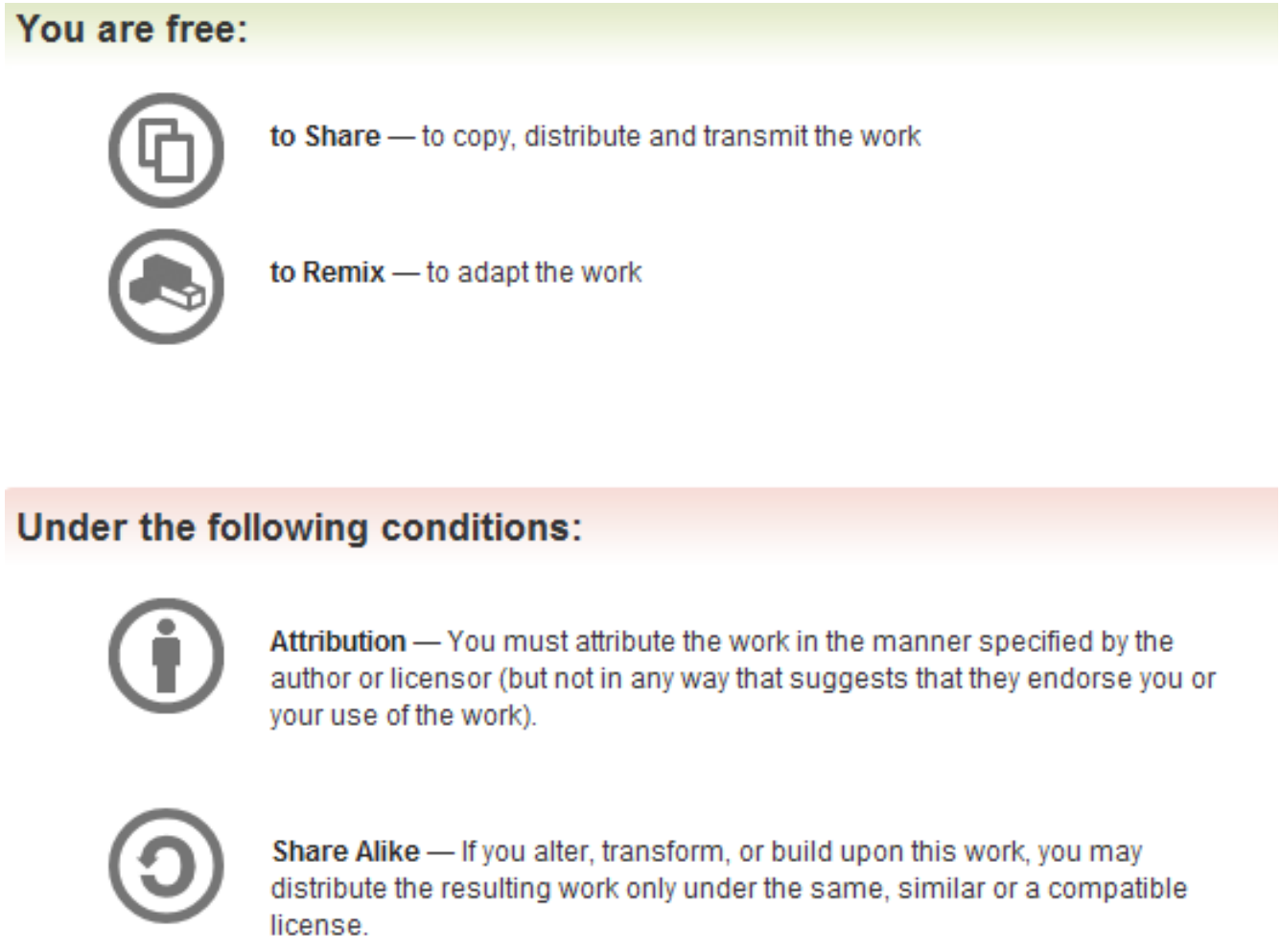
Rootkits:
What they are and how to find them
Part 2

Xeno Kovah – 2010

xkovah at gmail

All materials is licensed under a Creative Commons “Share Alike” license.

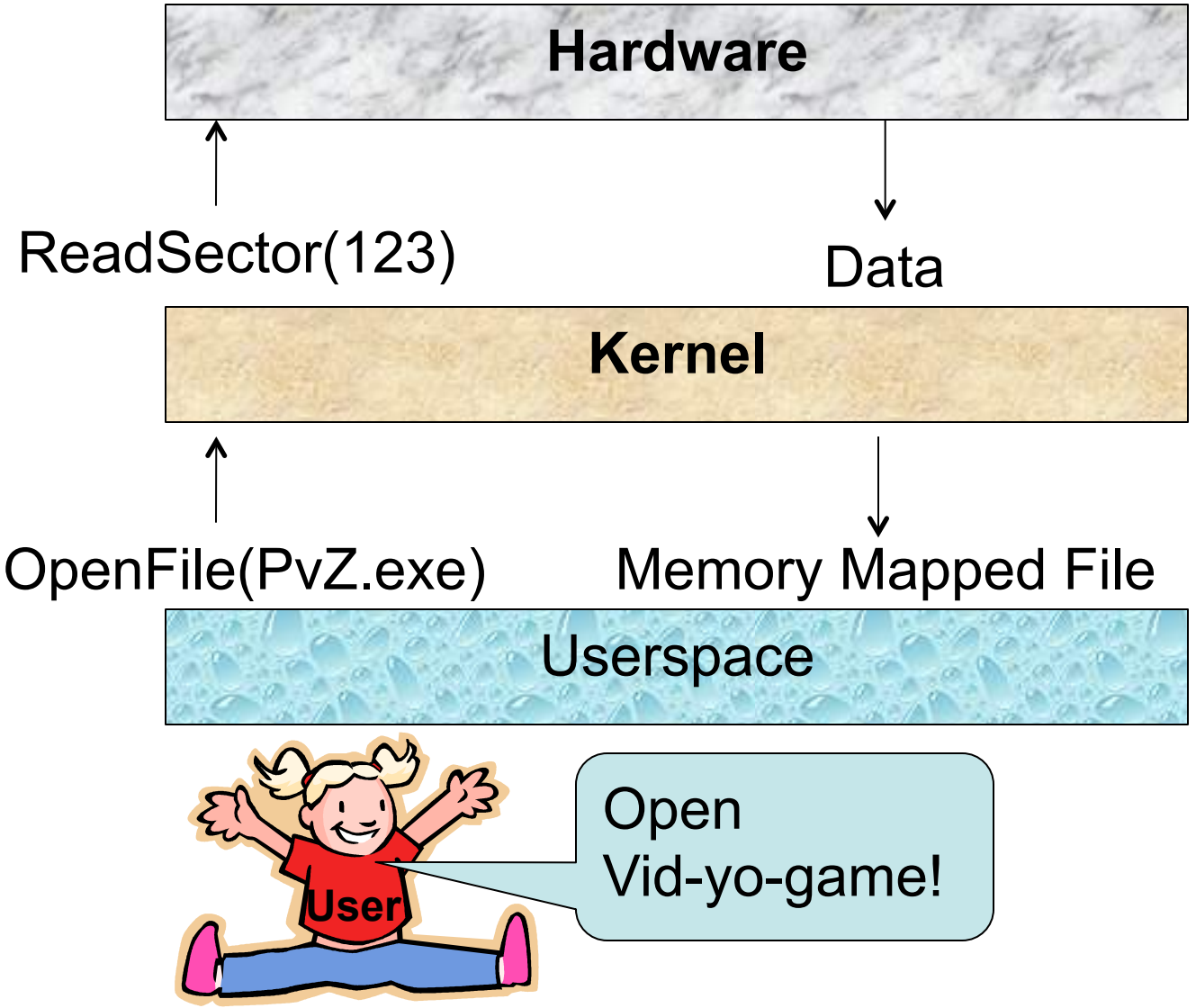
- http://creativecommons.org/licenses/by-sa/3.0/



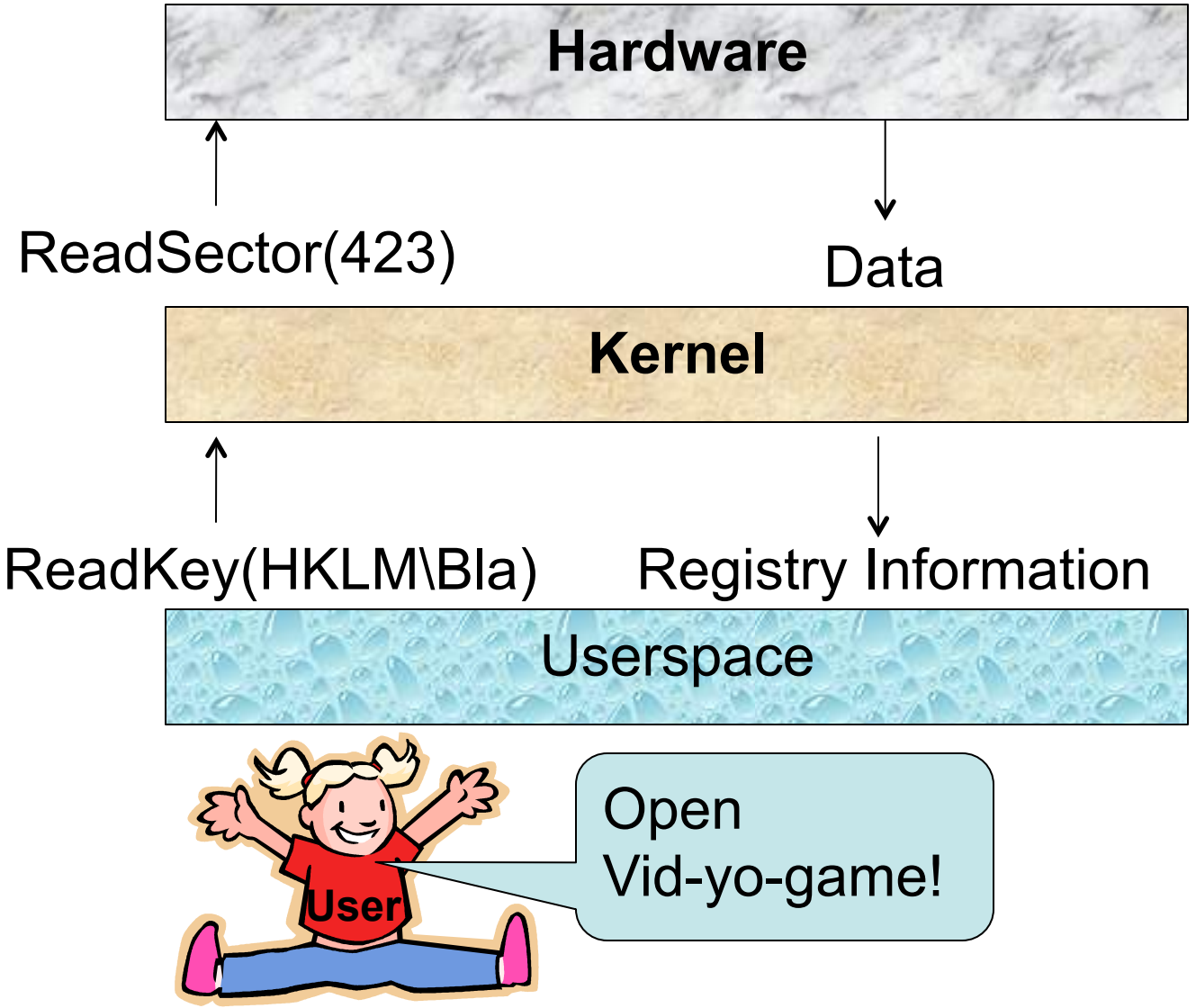
System Calls Revisited

- You need to see the full path, and know that attackers can hook basically everywhere along the path.

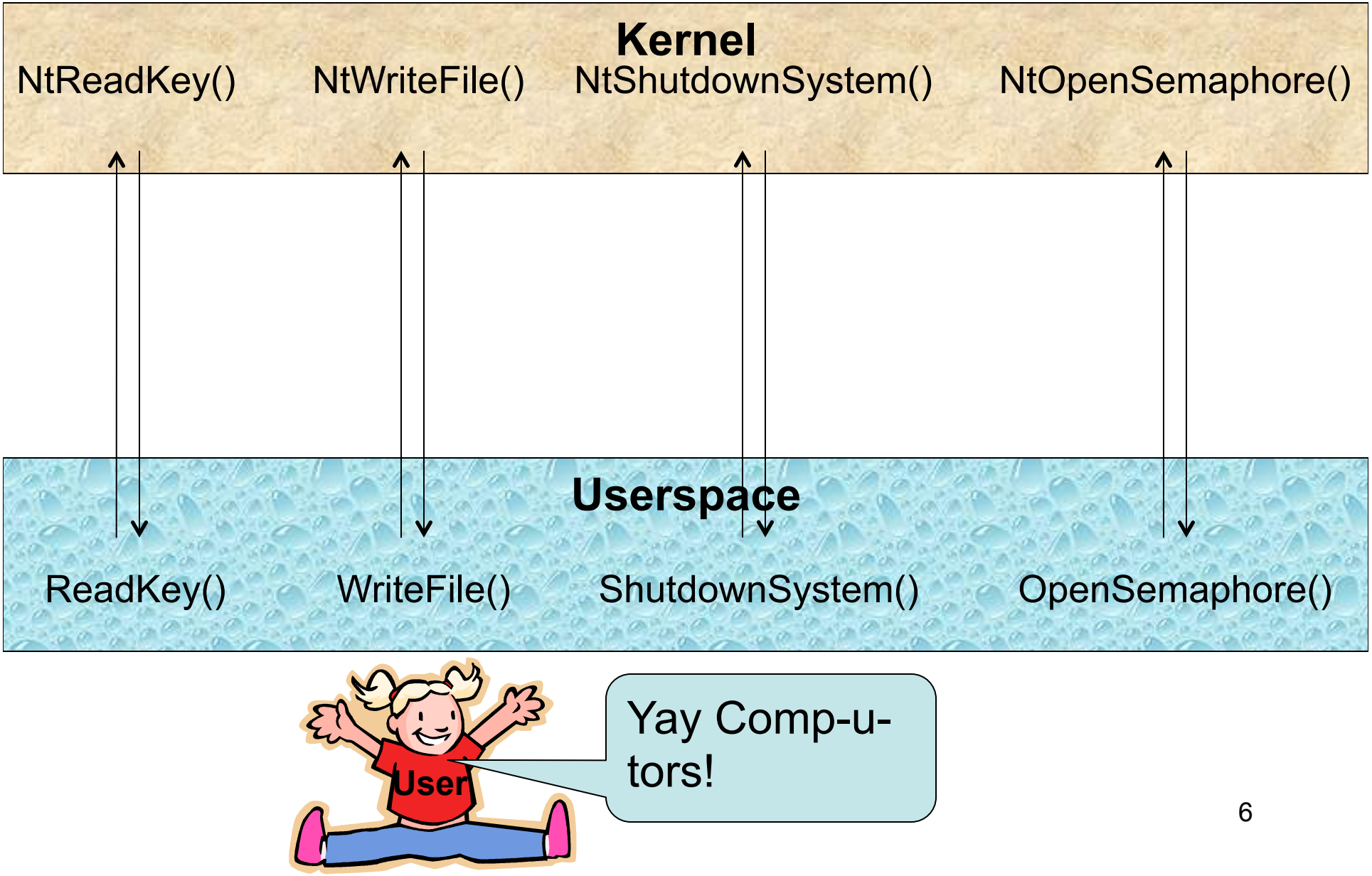
Conceptual Separation of Duties



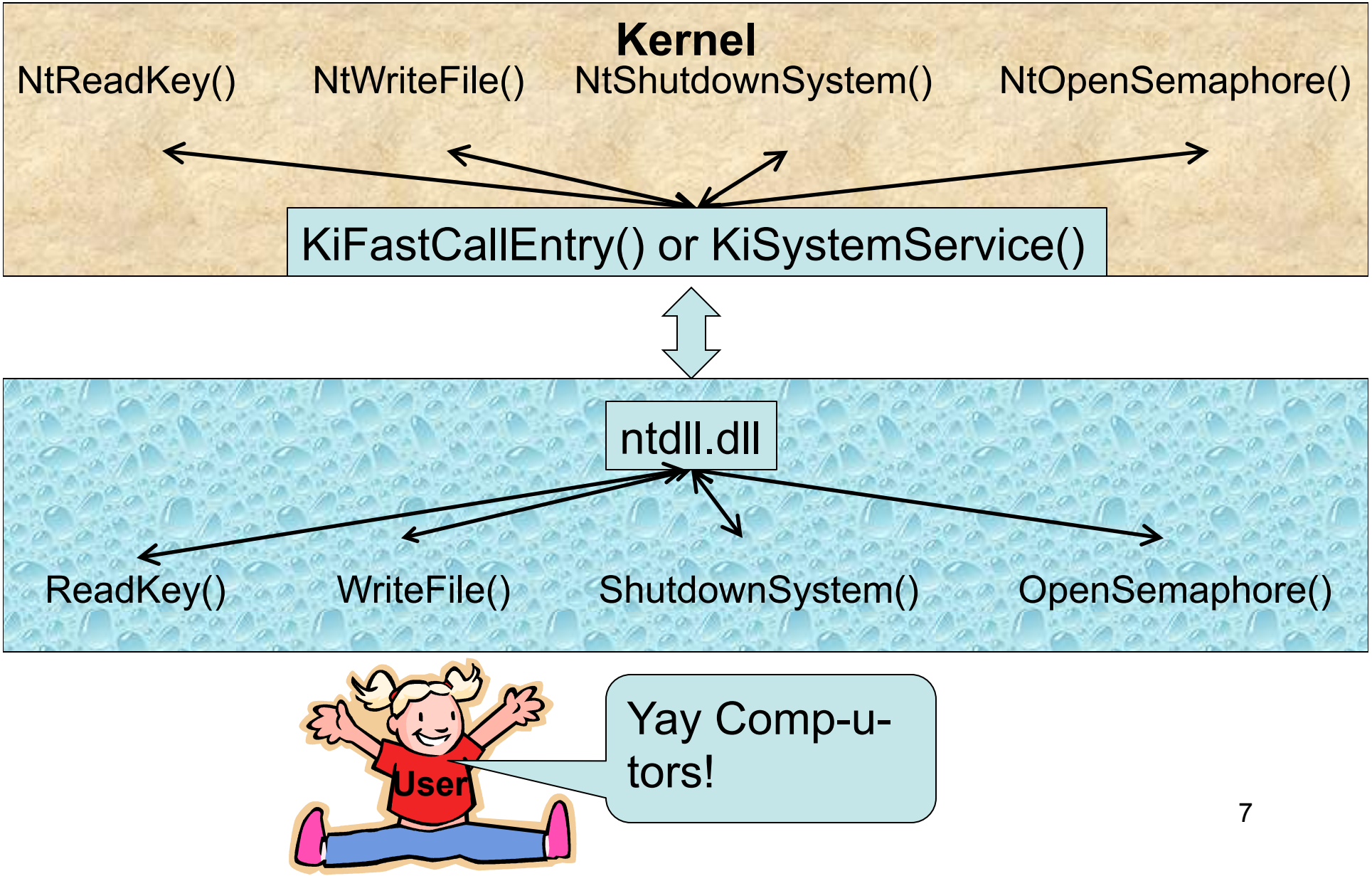
Conceptual Separation of Duties

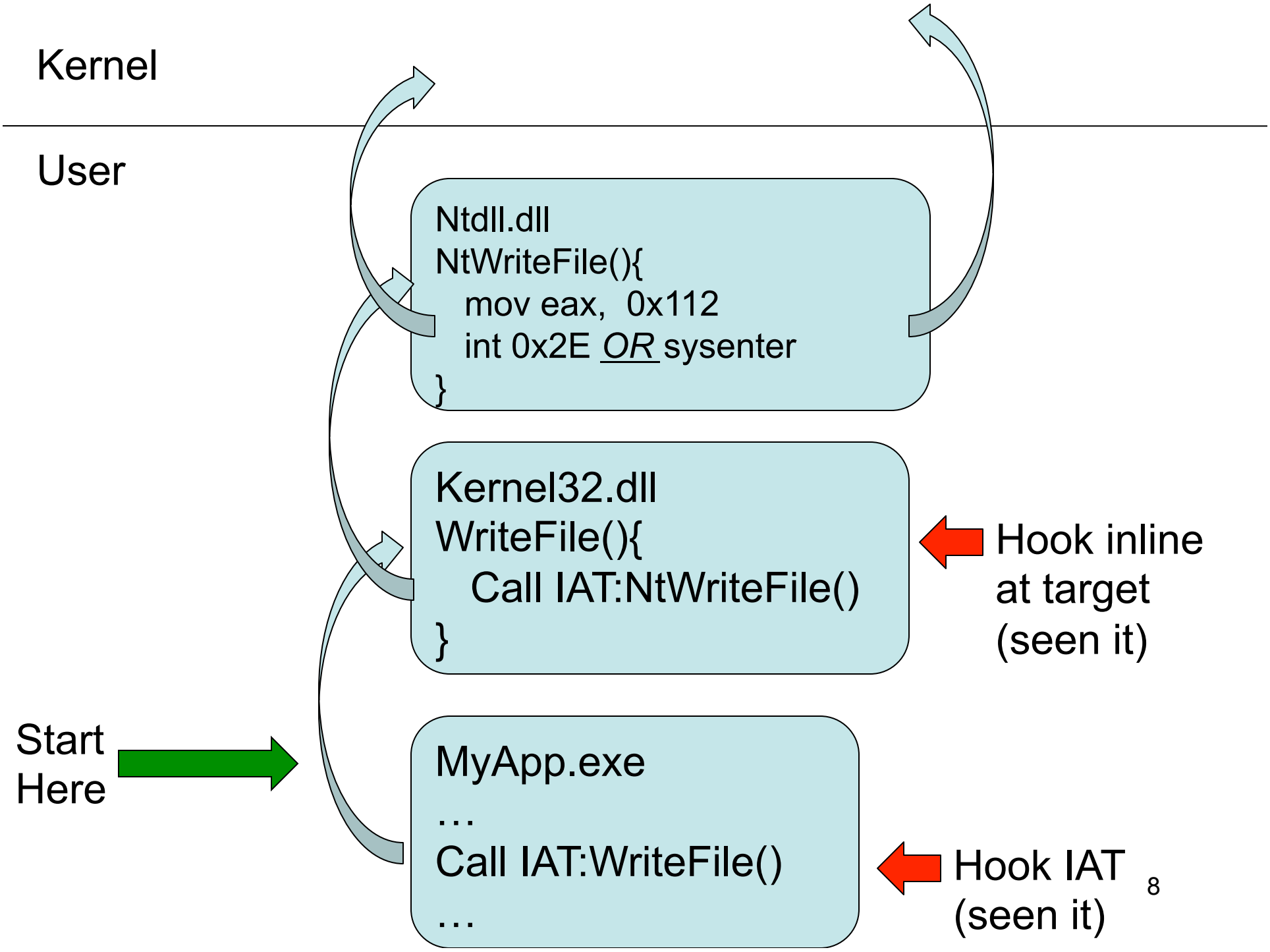


Conceptual System Call Interface

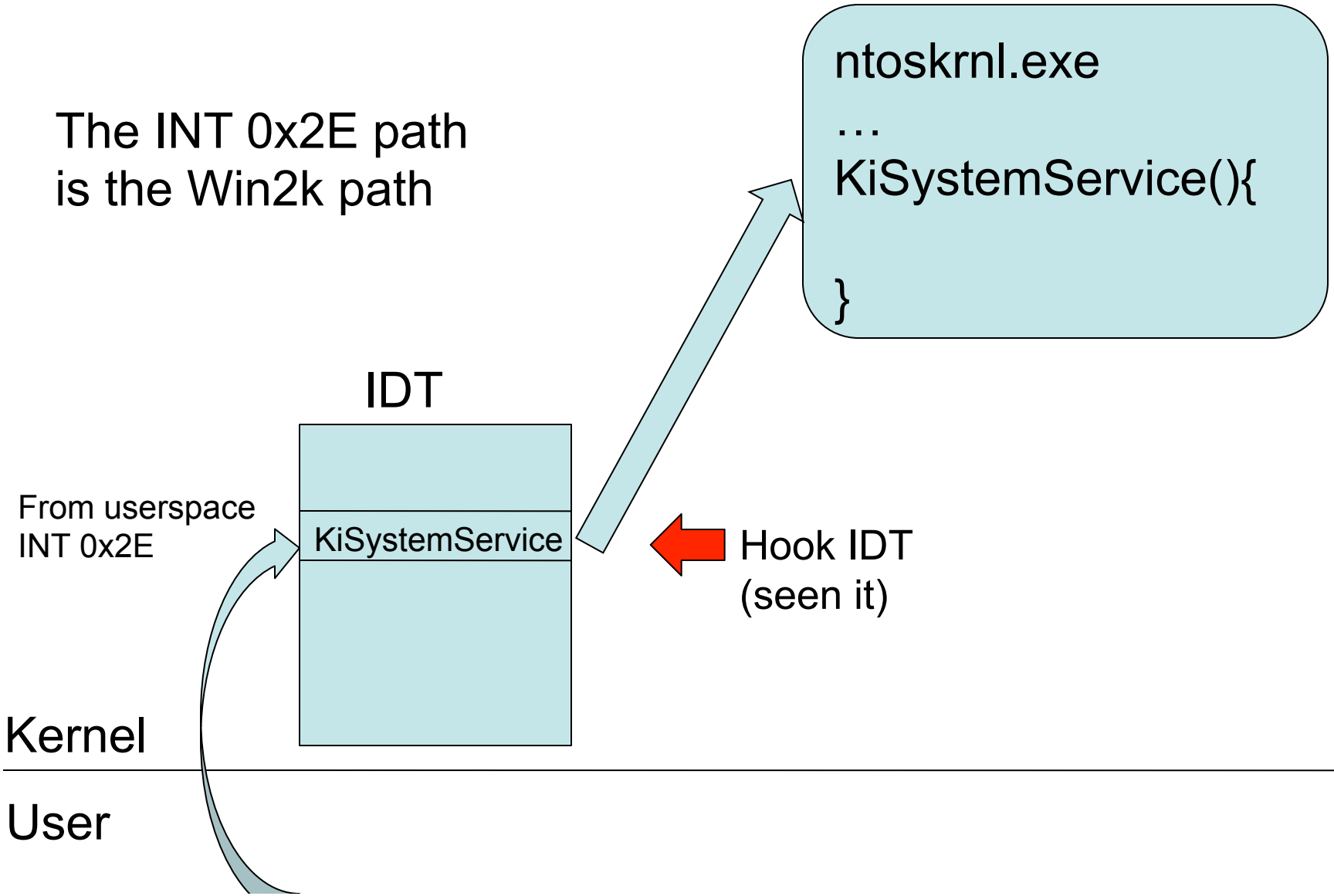


Slightly More Accurate System Call Interface



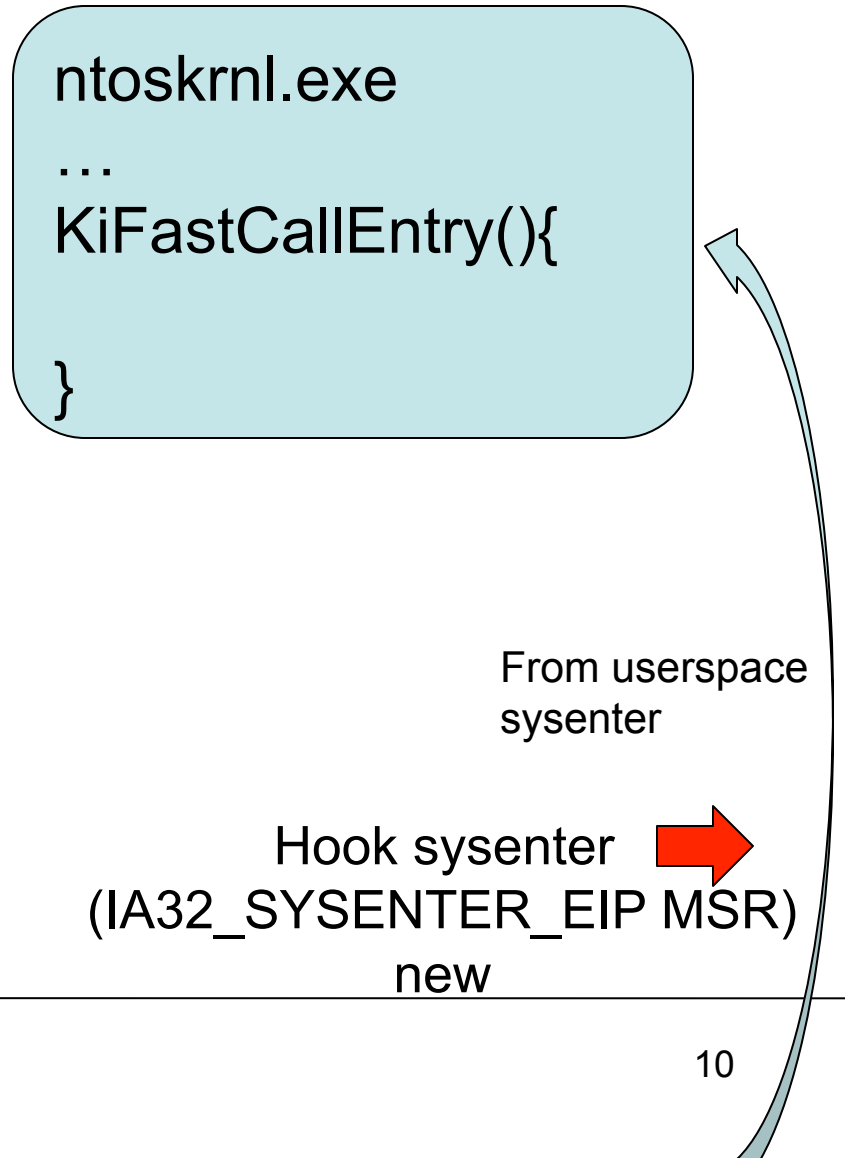

Kernel
User
Ntdll.dll
NtWriteFile(){
mov eax, 0x112
int 0x2E OR sysenter
}
Kernel32.dll
WriteFile(){
Call IAT:NtWriteFile()
}
Hook inline
at target
(seen it)
Start
Here
MyApp.exe
…
Call IAT:WriteFile()
…
Hook IAT
(seen it)

The INT 0x2E path
is the Win2k path



The sysenter path is the > Win2k path

KiFastCallEntry != KiSystemService



unused
IIS spud.sys (if installed and running)
unused
Native API

KeServiceDescriptorTable

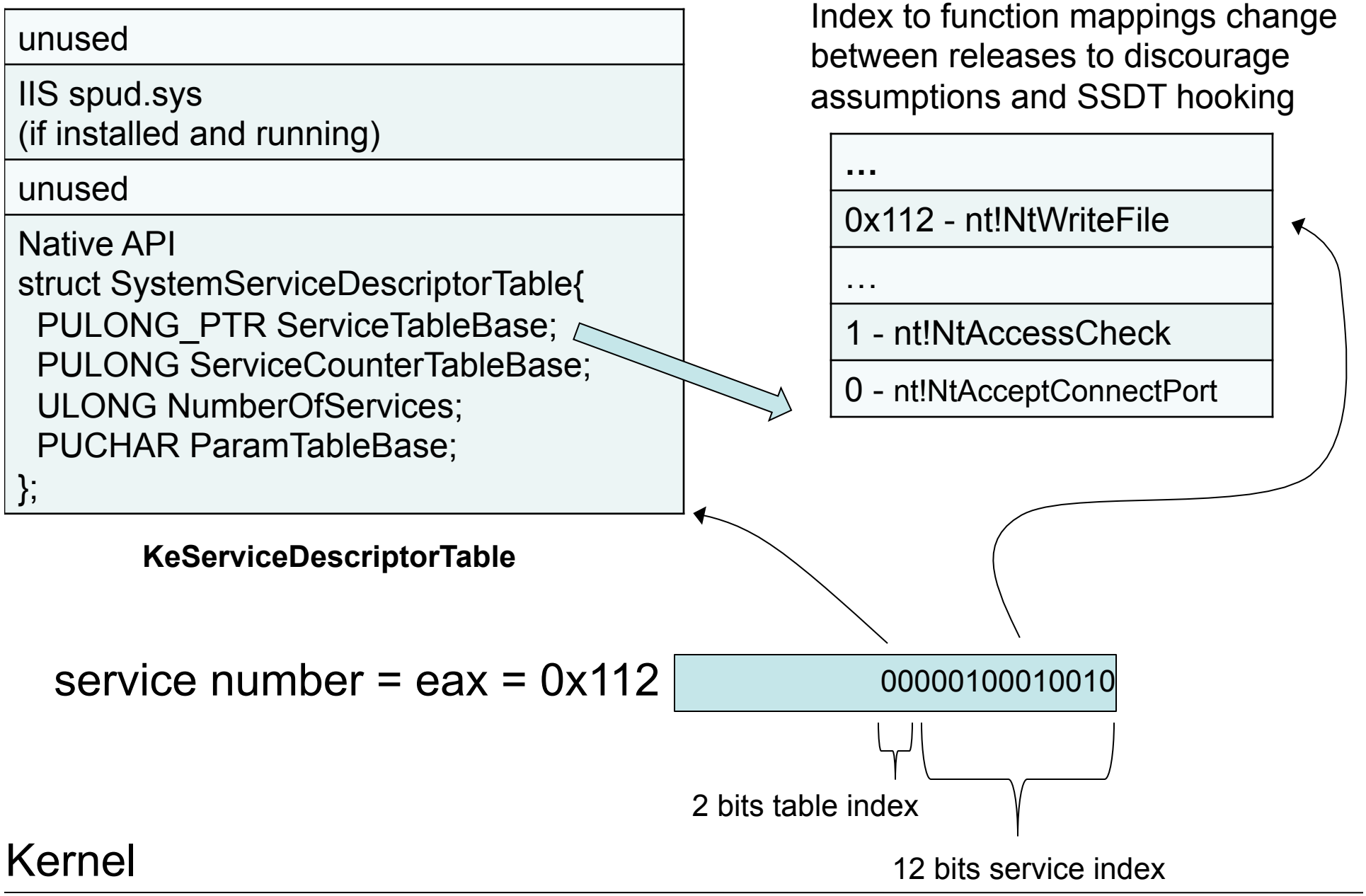
unused
IIS spud.sys (if installed and running)
Win32k.sys API
Native API

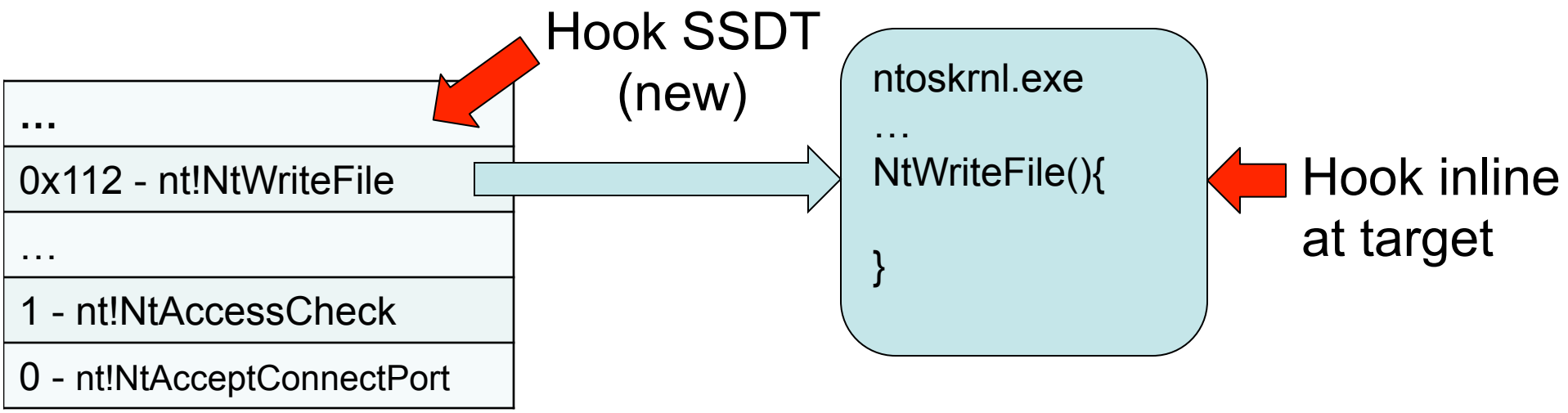
KeServiceDescriptorTableShadow

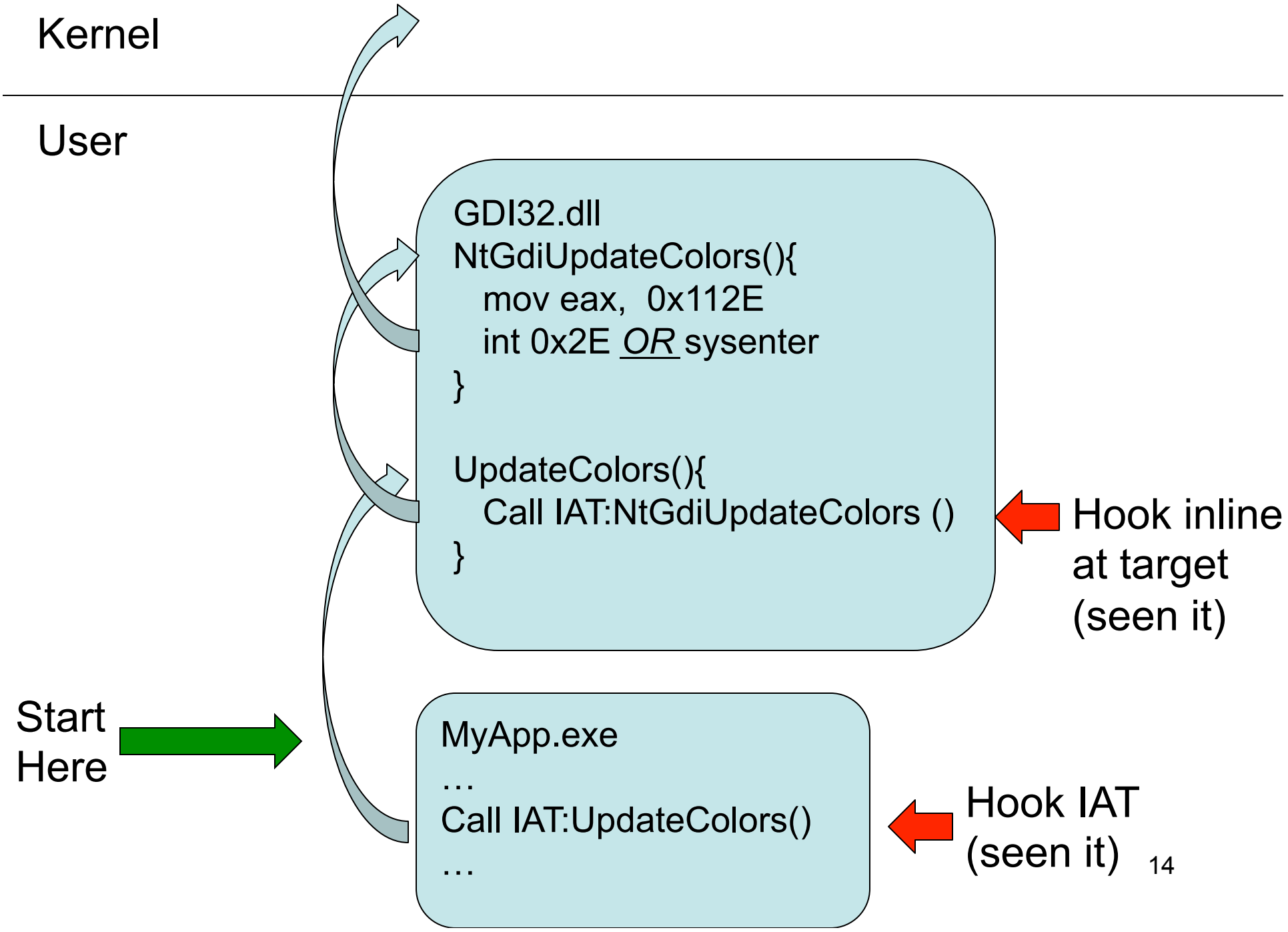
```
ntoskrnl.exe
…
KiSystemService() or KiFastCallEntry(){
  * Consult Thread Info
  * Extract address of System Service Descriptor Table (SSDT)
which is KeServiceDescriptorTable normally or
KeServiceDescriptorTableShadow if the process has used any
graphical (GDI) routines
 * Parse eax for specific table entry
}
```

Kernel

User


unused
IIS spud.sys
(if installed and running)
unused
Native API
struct SystemServiceDescriptorTable{
PULONG_PTR ServiceTableBase;
PULONG ServiceCounterTableBase;
ULONG NumberOfServices;
PUCHAR ParamTableBase;
};
KeServiceDescriptorTable
Index to function mappings change between releases to discourage assumptions and SSDT hooking
...
0x112 - nt!NtWriteFile
…
1 - nt!NtAccessCheck
0 - nt!NtAcceptConnectPort
service number = eax = 0x112
00000100010010
2 bits table index
12 bits service index
Kernel
User


Hook SSDT
(new)
...
0x112 - nt!NtWriteFile
…
1 - nt!NtAccessCheck
0 - nt!NtAcceptConnectPort
ntoskrnl.exe
…
NtWriteFile(){
}
Hook inline
at target


Kernel
User
GDI32.dll
NtGdiUpdateColors(){
mov eax, 0x112E
int 0x2E OR sysenter
}
UpdateColors(){
Call IAT:NtGdiUpdateColors ()
}
Hook inline at target (seen it)
Start Here
MyApp.exe
…
Call IAT:UpdateColors()
…
Hook IAT (seen it)

unused
IIS spud.sys (if installed and running)
unused
Native API

KeServiceDescriptorTable

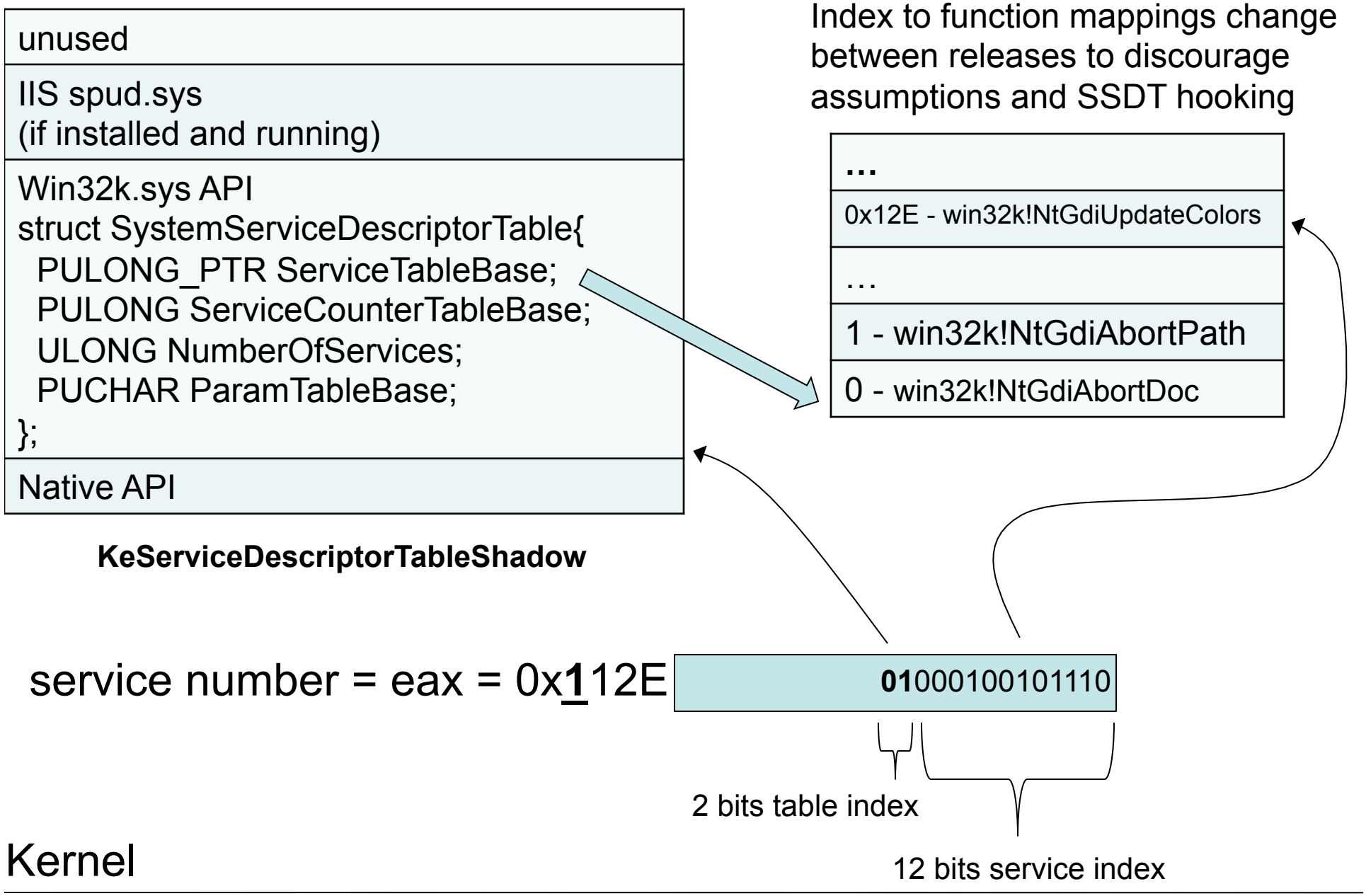
unused
IIS spud.sys (if installed and running)
Win32k.sys API
Native API

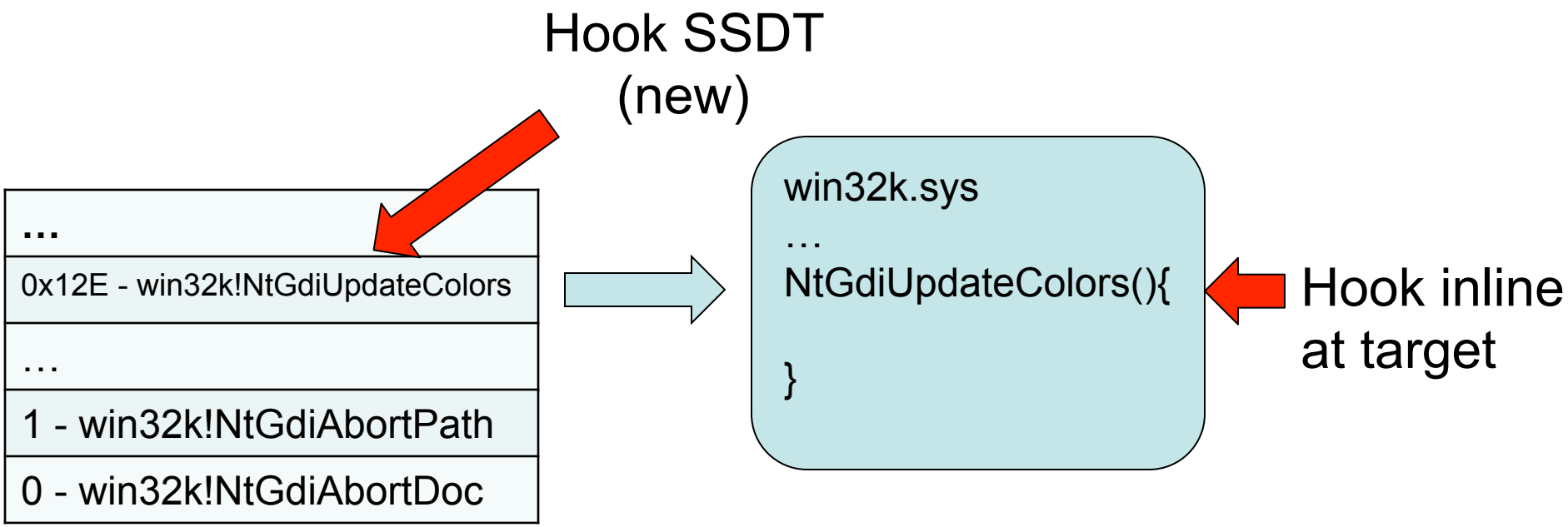
KeServiceDescriptorTableShadow

```
ntoskrnl.exe
…
KiSystemService() or KiFastCallEntry(){
  * Consult Thread Info
  * Extract address of System Service Descriptor Table (SSDT)
which is KeServiceDescriptorTable normally or
KeServiceDescriptorTableShadow if the process has used any
graphical (GDI) routines
 * Parse eax for specific table entry
}
```

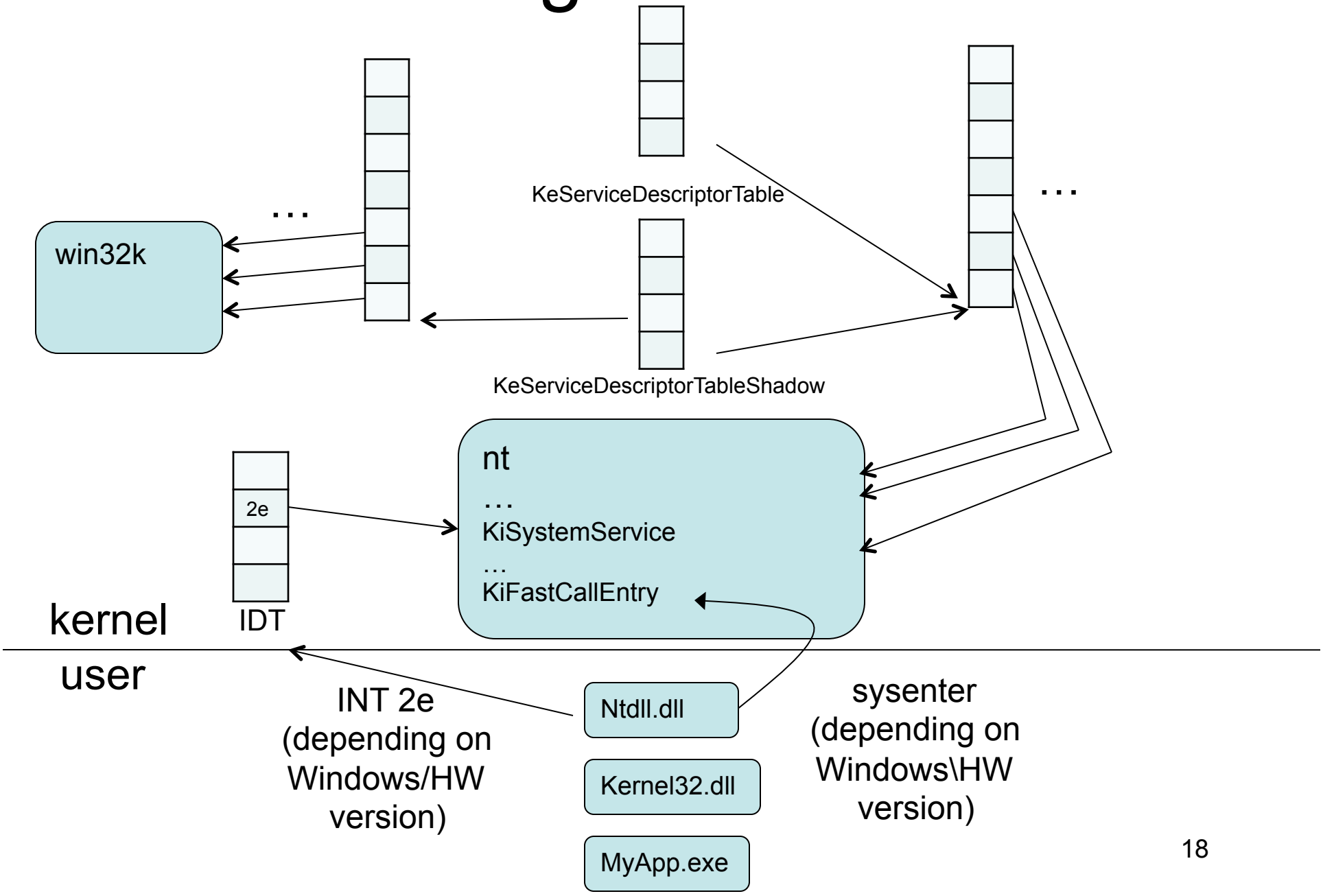
Kernel

User

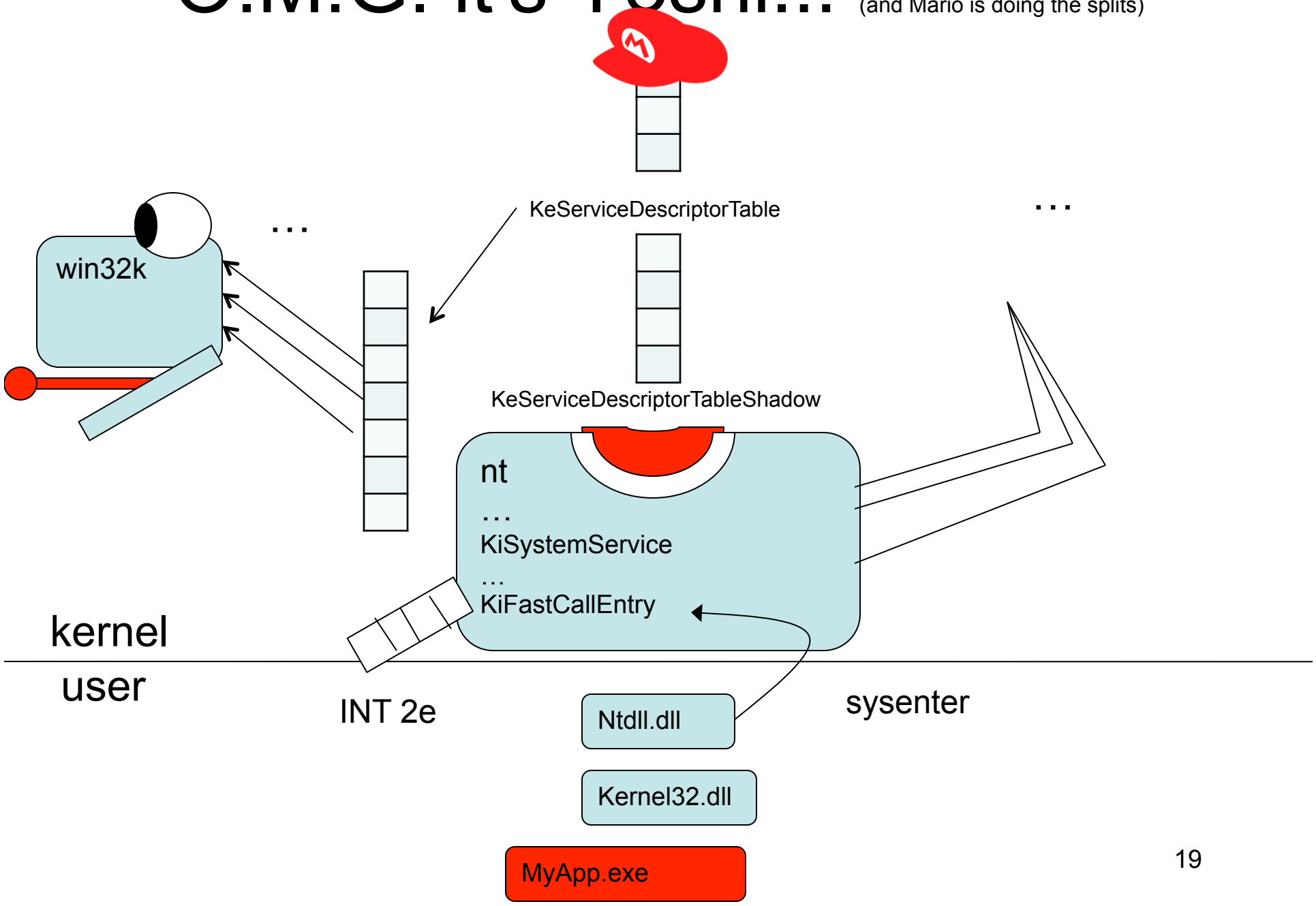

unused
IIS spud.sys
(if installed and running)
Win32k.sys API
struct SystemServiceDescriptorTable{
PULONG_PTR ServiceTableBase;
PULONG ServiceCounterTableBase;
ULONG NumberOfServices;
PUCHAR ParamTableBase;
};
Native API
KeServiceDescriptorTableShadow
Index to function mappings change between releases to discourage assumptions and SSDT hooking
...
0x12E - win32k!NtGdiUpdateColors
…
1 - win32k!NtGdiAbortPath
0 - win32k!NtGdiAbortDoc
service number = eax = 0x1112E
01000100101110
2 bits table index
12 bits service index
Kernel
User


Hook SSDT
(new)
...
0x12E - win32k!NtGdiUpdateColors
…
1 - win32k!NtGdiAbortPath
0 - win32k!NtGdiAbortDoc
win32k.sys
…
NtGdiUpdateColors(){
}
Hook inline
at target

Big Picture


KeServiceDescriptorTable
KeServiceDescriptorTableShadow
win32k
...
...
nt
…
KiSystemService
…
KiFastCallEntry
2e
IDT
kernel
user
INT 2e
(depending on
Windows/HW
version)
Ntdll.dll
Kernel32.dll
MyApp.exe
sysenter
(depending on
Windows\HW
version)

O.M.G. it's Yoshi!!! (and Mario is doing the splits)



Nu2U - sysenter

- We never talked about the sysenter instruction in Intermediate x86, due to lack of time, even though it would work well in that class.
- Sort of as a background notion when we were talking about interrupts that they were 1) a way for hardware to get the CPU's attention, and 2) a way to get some kernel code to execute (BreakOnThruToTheOtherSide lab, discussion of interrupts underlying debugging.)
- So "back in the day" systems would implement the "system call table"(*nix) or "system service descriptor table" (Windows) as a way for userspace code to ask the kernel to do specific actions for it. E.g. open a file, allocate some memory,
- This was achieved by putting a system call number in some register and then calling int 0x80 (linux), or int 0x2e (Windows). The code on the kernel side would then just check the designated register(s) which were input parameters and call the appropriate kernel library function.

Book page 279, 321

Out with the old, in with the Nu2U

- Intel and AMD introduced a specific instruction for achieving this same sort of system call table capability for kernels, but doing it more efficiently.
- The instructions for doing this are syscall/sysret on AMD and sysenter/ sysexit on Intel.
- Linux used int 0x80 <= 2.4, and sys* >= 2.5, Windows used int 0x2e <= Win2k, sys* >= XP

MiSeRly MiSeRy MiSanthRopy

- The syscall/sysenter instructions basically just jump to a predefined location in the kernel ala an interrupt. That location is predefined by using a "Model Specific Register" (MSR)
- MSRs are special registers which exist on specific models and have specific purposes (not a "general purpose" register like eax, ebx, etc.)
- You read and write MSRs with "rdmsr" (read msr) and "wrmsr“ (write MSR)
- IA32_SYSENTER_EIP = 0x176

MiSeRly MiSeRy MiSanthRopy

- IA32_SYSENTER_EIP = 0x176
- Reading from the MSR
  - mov ecx, 0x176
  - rdmsr
  - (eax now contains value that was in the MSR)
- Writing a MSR
  - mov eax, 0xdeadbeef
  - mov ecx, 0x176
  - wrmsr
  - (IA32_SYSENTER_EIP now holds the value 0xdeadbeef)

More about system calls

- For more into on int vs sys*, as well as how interrupts work and worked on Windows:
  - How Do Windows NT System Calls REALLY Work? - http://www.codeguru.com/Cpp/W-P/system/devicedriverdevelopment/article.php/c8035/
  - System Call Optimization with the SYSENTER Instruction - http://www.codeguru.com/cpp/w-p/system/devicedriverdevelopment/print.php/c8223
- It's going to make a whole lot more sense thanks to Intermediate x86 :)

NooTooYoo - SSDT

- WinDbg command to print tables:
- !for_each_thread ".echo Thread: @#Thread; dt nt!_kthread ServiceTable @#Thread"
- (from http://www.securabit.com/wp-content/uploads/2010/03/Rootkit-Analysis-Hiding-SSDT-Hooks1.pdf)

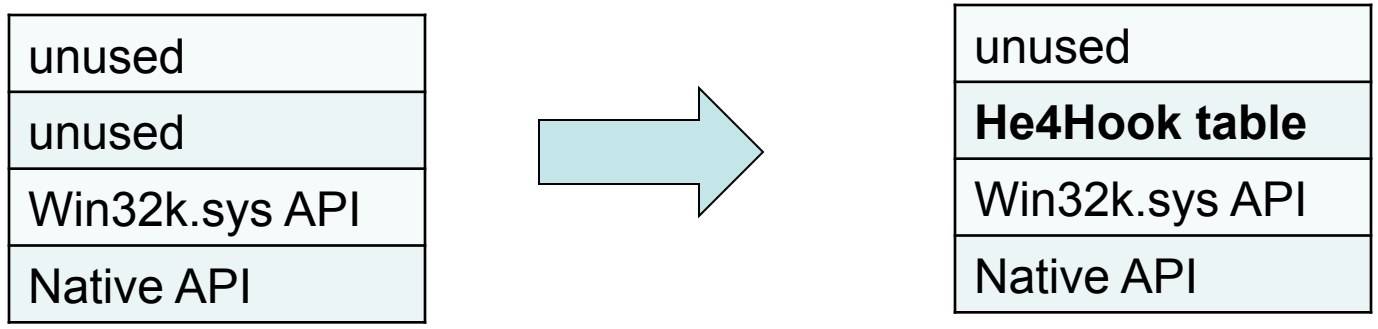
Book page 286, 324

KeAddSystemServiceTable()

- He4Hook uses KeAddSystemServiceTable() (which was first talked about in Hoglund's NT Rootkit phrack article) to talk from its userspace component to kernel
- KeAddSystemServiceTable() adds in one of those SystemServiceDescriptorTable structs onto the table pointed to by KeServiceDescriptorTableShadow

He4Hook KeAddSystemServiceTable

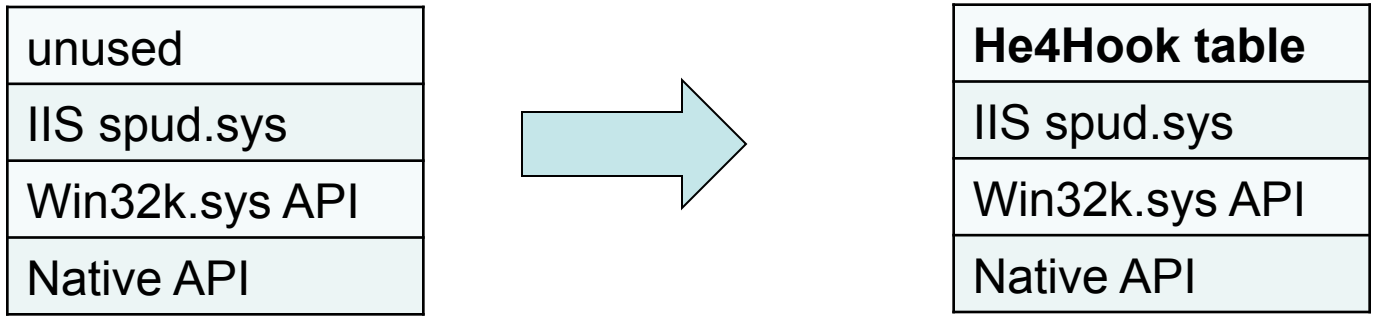
ASSUMING NO IIS INSTALLED



KeServiceDescriptorTableShadow before KeAddSystemServiceTable()

KeServiceDescriptorTableShadow after KeAddSystemServiceTable()

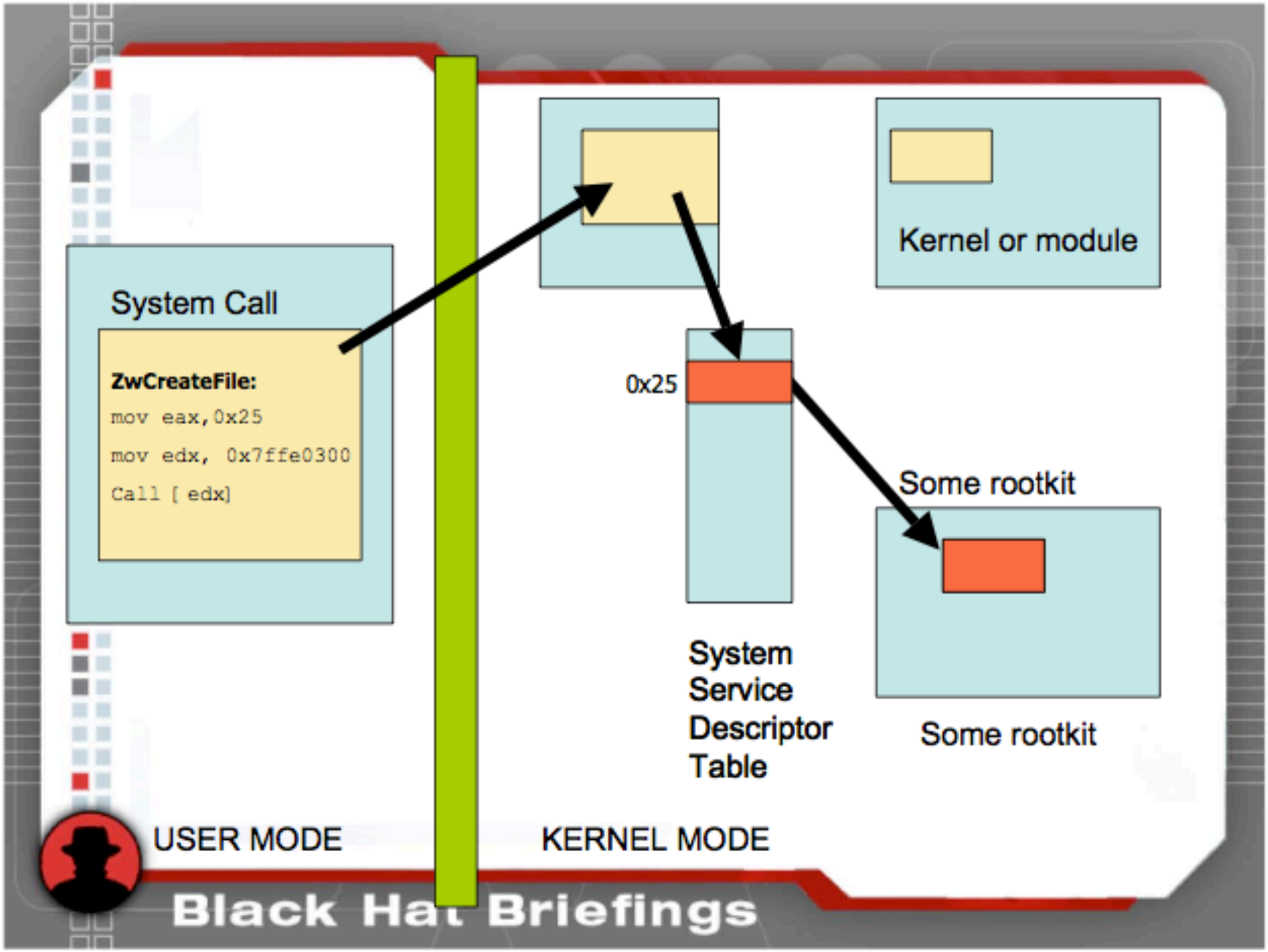
ASSUMING IIS INSTALLED



KeServiceDescriptorTableShadow before KeAddSystemServiceTable()

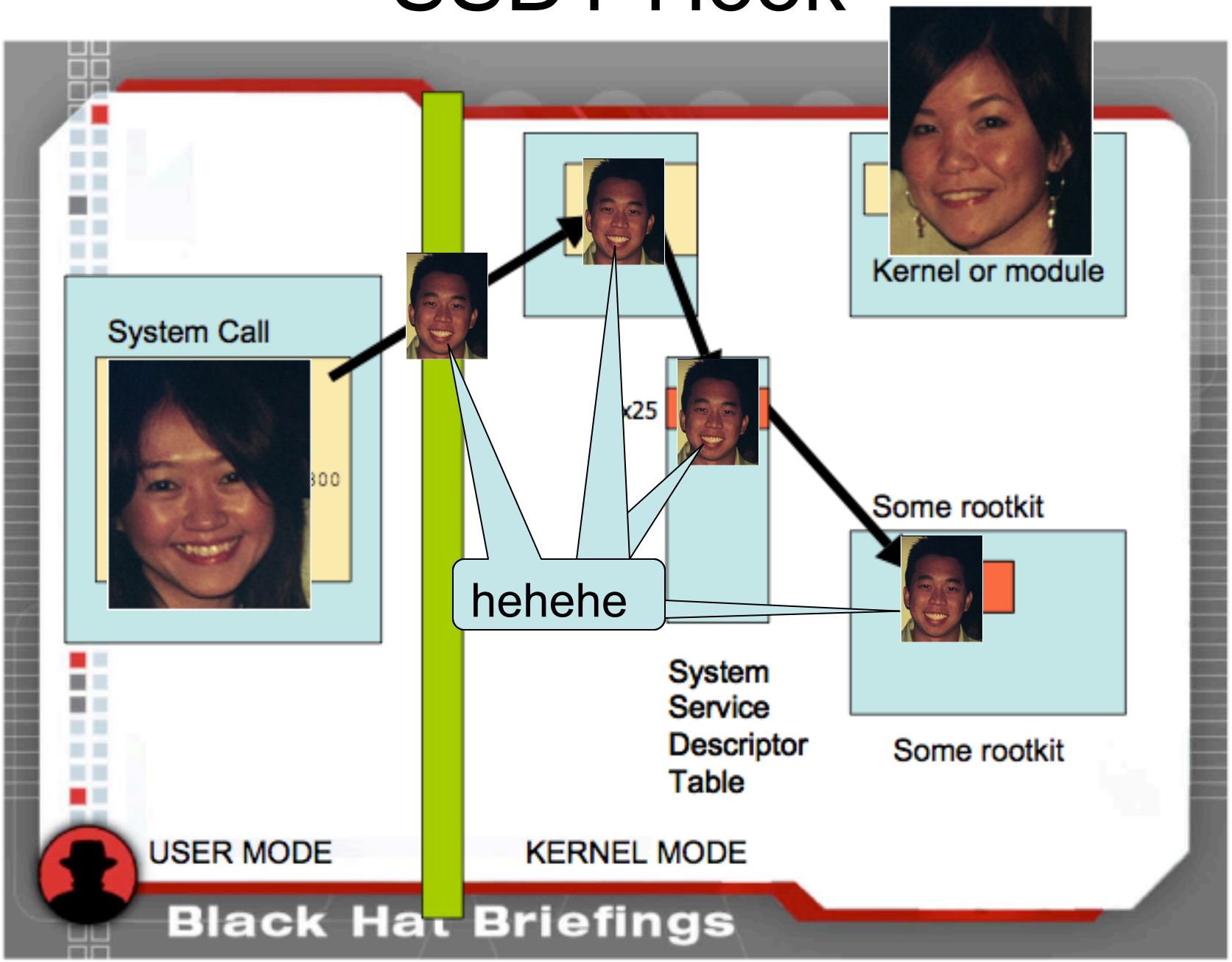
KeServiceDescriptorTableShadow after KeAddSystemServiceTable()

SSDT Hook



From: http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf

SSDT Hook



From: http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf

SSDT False Positives

(go look at the overall SSDT results again at this point)

GMER 1.0.14.14536

Rootkit/Malware | > > >

Type	Name	Value
SSDT	81EB36A0	ZwAlertResumeThread
SSDT	81EB3E50	ZwAlertThread
SSDT	82327FC0	ZwAllocateVirtualMemory
SSDT	82129928	ZwConnectPort
SSDT	820BEE88	ZwCreateMutant
SSDT	822B61F8	ZwCreateThread
SSDT	8211F8F0	ZwFreeVirtualMemory
SSDT	81E9FF40	ZwImpersonateAnonymousToken
SSDT	81EE2008	ZwImpersonateThread
SSDT	822B63F0	ZwMapViewOfSection
SSDT	822D57B0	ZwOpenEvent
SSDT	81EB65F0	ZwOpenProcessToken
SSDT	822A1168	ZwOpenThreadToken
SSDT	\??\C:\WINDOWS\system32\drivers\wpsdrvnt.sys (Symantec C...	ZwProtectVirtualMemory [0xF8E94880]
SSDT	81EB6F30	ZwResumeThread
SSDT	81EB60F0	ZwSetContextThread
SSDT	82104168	ZwSetInformationProcess
SSDT	8206A168	ZwSetInformationThread
SSDT	821FC0B8	ZwSuspendProcess
SSDT	81EB4848	ZwSuspendThread
SSDT	81EB67A8	ZwTerminateProcess
SSDT	81EB5600	ZwTerminateThread
SSDT	81EB62D8	ZwUnmapViewOfSection
SSDT	821EDB48	ZwWriteVirtualMemory

How you could determine these are due to symantec and not a rootkit is given in the tiddlywiki file in the class materials

NouTouYou - IRP

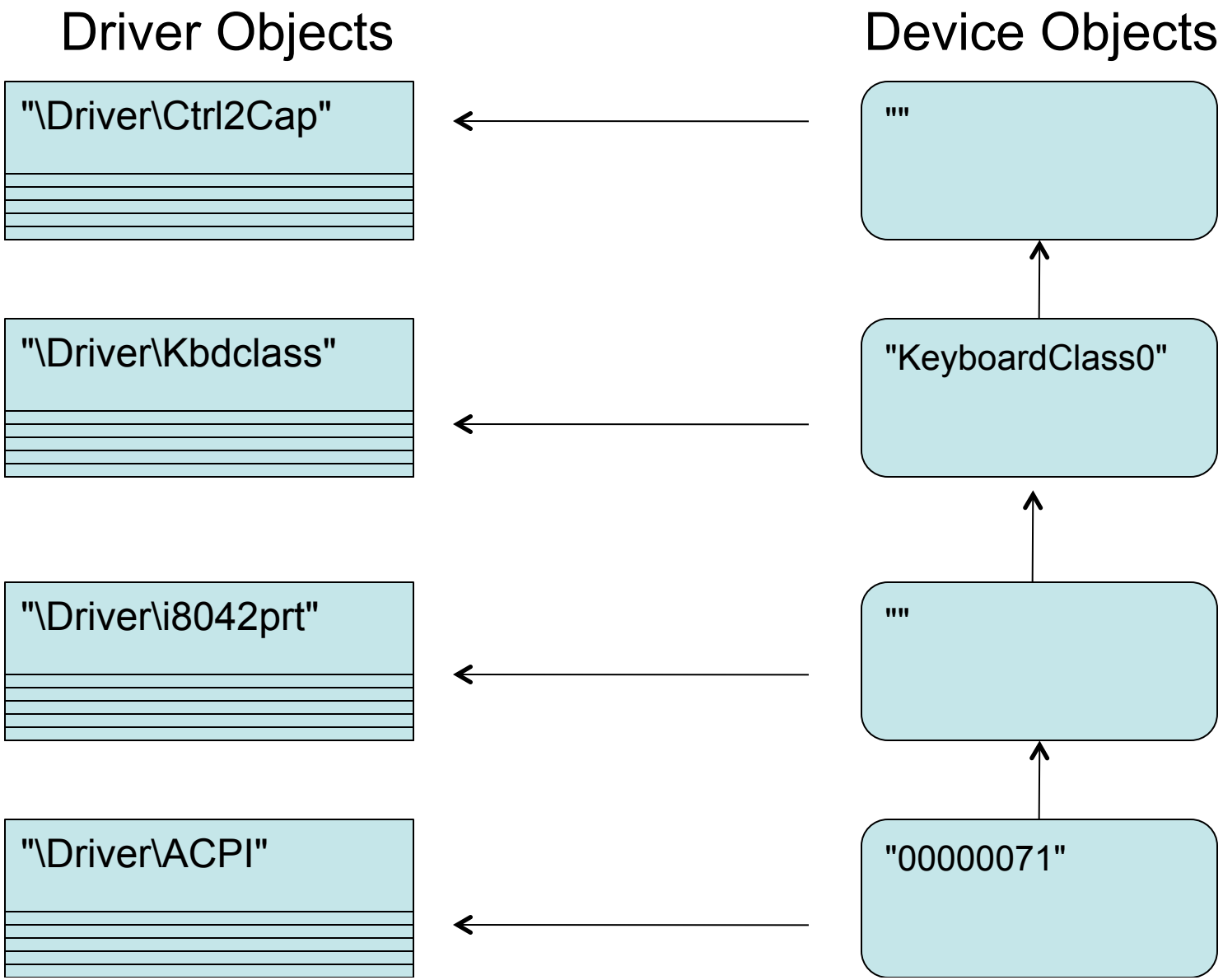
- Windows uses an abstraction called IO Request Packets (IRPs) in order to send events to and from hardware IO devices.
- Drivers can attach to devices with IoAttachDeviceToDeviceStack(), which is how they indicate that they would like to hear about IRPs to/from a specific device.
- They can also just *not* attach to the stack, and instead intercept the calls to someone who's already attached.

Book page 325, 457

A tale of two objects

- Driver Object
  - Every driver gets this object passed to it when it's loaded as the first parameter of the required driver entry point function
- Device Object
  - Used to create a linked list which holds the other devices for other drivers which want to hear about IRP activity

IRP chain


Driver Objects
Device Objects
"\Driver\Ctrl2Cap"
""
"\Driver\Kbdclass"
"KeyboardClass0"
"\Driver\i8042prt"
""
"\Driver\ACPI"
"00000071"

DRIVER_OBJECT struct (on XP)

```
lkd> dt nt!_DRIVER_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x004 DeviceObject     : Ptr32 _DEVICE_OBJECT
   +0x008 Flags            : Uint4B
   +0x00c DriverStart      : Ptr32 Void
   +0x010 DriverSize       : Uint4B
   +0x014 DriverSection    : Ptr32 Void
   +0x018 DriverExtension  : Ptr32 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING
   +0x024 HardwareDatabase : Ptr32 _UNICODE_STRING
   +0x028 FastIoDispatch   : Ptr32 _FAST_IO_DISPATCH
   +0x02c DriverInit       : Ptr32     long
   +0x030 DriverStartIo    : Ptr32     void
   +0x034 DriverUnload     : Ptr32     void
   +0x038 MajorFunction    : [28] Ptr32     long
```

DEVICE_OBJECT struct (on XP)

http://msdn.microsoft.com/en-us/library/ff543147(v=vs.85).aspx

```
typedef struct _DEVICE_OBJECT {
  CSHORT                      Type;
  USHORT                      Size;
  LONG                        ReferenceCount;
  struct _DRIVER_OBJECT *     DriverObject;
  struct _DEVICE_OBJECT *     NextDevice;
  struct _DEVICE_OBJECT *     AttachedDevice;
  struct _IRP *               CurrentIrp;
  PIO_TIMER                   Timer;
  ULONG                       Flags;
  ULONG                       Characteristics;
  __volatile PVPB             Vpb;
  PVOID                       DeviceExtension;
  DEVICE_TYPE                 DeviceType;
  CCHAR                       StackSize;
  union {
    LIST_ENTRY         ListEntry;
    WAIT_CONTEXT_BLOCK Wcb;
  } Queue;
  ULONG                       AlignmentRequirement;
  KDEVICE_QUEUE               DeviceQueue;
  KDPC                        Dpc;
  ULONG                       ActiveThreadCount;
  PSECURITY_DESCRIPTOR        SecurityDescriptor;
  KEVENT                      DeviceLock;
  USHORT                      SectorSize;
  USHORT                      Spare1;
  struct _DEVOBJ_EXTENSION  * DeviceObjectExtension;
  PVOID                       Reserved;
} DEVICE_OBJECT, *PDEVICE_OBJECT;
```

"**NextDevice:** A pointer to the next device object, if any, that was created by the same driver. The I/O manager updates this list at each successful call to **IoCreateDevice or IoCreateDeviceSecure."**

"The device object that is pointed to by the **AttachedDevice** member typically is the device object of a filter driver, which intercepts I/O requests originally targeted to the device represent by the device object. "

IRP struct

(see wdm.h for comments on fields)

```
kd> dt _IRP
ntdll!_IRP
   +0x000 Type             : Int2B
   +0x002 Size             : Uint2B
   +0x004 MdlAddress       : Ptr32 _MDL
   +0x008 Flags            : Uint4B
   +0x00c AssociatedIrp    : __unnamed
   +0x010 ThreadListEntry  : _LIST_ENTRY
   +0x018 IoStatus         : _IO_STATUS_BLOCK
   +0x020 RequestorMode    : Char
   +0x021 PendingReturned  : UChar
   +0x022 StackCount       : Char
   +0x023 CurrentLocation  : Char
   +0x024 Cancel           : UChar
   +0x025 CancelIrql       : UChar
   +0x026 ApcEnvironment   : Char
   +0x027 AllocationFlags  : UChar
   +0x028 UserIosb         : Ptr32 _IO_STATUS_BLOCK
   +0x02c UserEvent        : Ptr32 _KEVENT
   +0x030 Overlay          : __unnamed
   +0x038 CancelRoutine    : Ptr32     void
   +0x03c UserBuffer       : Ptr32 Void
   +0x040 Tail             : __unnamed
```

I am the very model of a modern major function bla

(in parody, it's important to maintain the correct number of syllables…wes :P)

- MajorFunction[] is an array of callback functions which will be called when IRPs are traversing the chain.
- This table is the target for function pointer hooking, in both legitimate and illegitimate software. So just like with the SSDT, you have to be aware of what 3rd party software might be hooking it.

major functions

(from wdm.h)

http://msdn.microsoft.com/en-us/library/ff550710.aspx

```
#define IRP_MJ_CREATE                   0x00
#define IRP_MJ_CREATE_NAMED_PIPE        0x01
#define IRP_MJ_CLOSE                    0x02
#define IRP_MJ_READ                     0x03
#define IRP_MJ_WRITE                    0x04
#define IRP_MJ_QUERY_INFORMATION        0x05
#define IRP_MJ_SET_INFORMATION          0x06
#define IRP_MJ_QUERY_EA                 0x07
#define IRP_MJ_SET_EA                   0x08
#define IRP_MJ_FLUSH_BUFFERS            0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION   0x0b
#define IRP_MJ_DIRECTORY_CONTROL        0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL      0x0d
#define IRP_MJ_DEVICE_CONTROL           0x0e
```

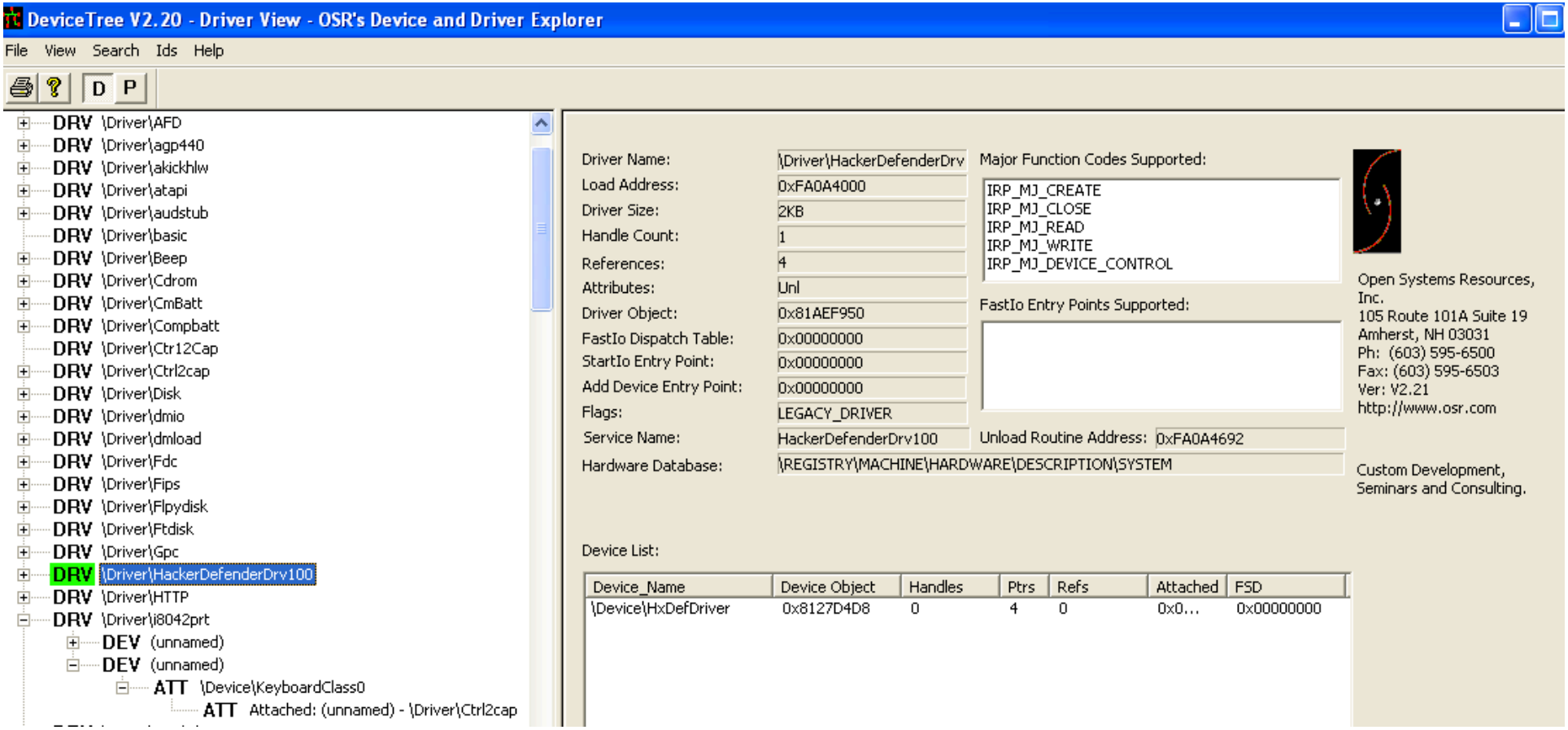
major functions 2

(from wdm.h)

```
#define IRP_MJ_INTERNAL_DEVICE_CONTROL  0x0f
#define IRP_MJ_SHUTDOWN                 0x10
#define IRP_MJ_LOCK_CONTROL             0x11
#define IRP_MJ_CLEANUP                  0x12
#define IRP_MJ_CREATE_MAILSLOT          0x13
#define IRP_MJ_QUERY_SECURITY           0x14
#define IRP_MJ_SET_SECURITY             0x15
#define IRP_MJ_POWER                    0x16
#define IRP_MJ_SYSTEM_CONTROL           0x17
#define IRP_MJ_DEVICE_CHANGE            0x18
#define IRP_MJ_QUERY_QUOTA              0x19
#define IRP_MJ_SET_QUOTA                0x1a
#define IRP_MJ_PNP                      0x1b
#define IRP_MJ_PNP_POWER                IRP_MJ_PNP      //
  Obsolete....
#define IRP_MJ_MAXIMUM_FUNCTION         0x1b
```

DeviceTree

- http://www.osronline.com/article.cfm?article=97



WinDbg (display device driver stack)

```
kd> !object \device\keyboardclass0
Object: 814e7d28  Type: (819b8ca0) Device
    ObjectHeader: 814e7d10 (old version)
    HandleCount: 0  PointerCount: 3
    Directory Object: e1006948  Name: KeyboardClass0
kd> !devstack 814e7d28
  !DevObj   !DrvObj            !DevExt   ObjectName
> 814e7d28  \Driver\Kbdclass   814e7de0  KeyboardClass0
  814e7020  \Driver\i8042prt   814e70d8
  8167c030  \Driver\ACPI       819a32e8  00000070
!DevNode 818f7348 :
  DeviceInst is "ACPI\PNP0303\4&5289e18&0"
  ServiceName is "i8042prt"
```

WinDbg 2 (display driver object)

```
kd> !devobj 814e7d28
Device object (814e7d28) is for:
 KeyboardClass0 \Driver\Kbdclass DriverObject 814ea0b8
Current Irp 00000000 RefCount 0 Type 0000000b Flags 00002044
Dacl e13cf7cc DevExt 814e7de0 DevObjExt 814e7ec0
ExtensionFlags (0000000000)
AttachedTo (Lower) 814e7020 \Driver\i8042prt
Device queue is not busy.
kd> dt nt!_DRIVER_OBJECT 814ea0b8
   +0x000 Type             : 4
   +0x002 Size             : 168
   +0x004 DeviceObject     : 0x81872030 _DEVICE_OBJECT
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xf9c4c000
   +0x010 DriverSize       : 0x6000
   +0x014 DriverSection    : 0x819b7aa8
   +0x018 DriverExtension  : 0x814ea160 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\Kbdclass"
   +0x024 HardwareDatabase : 0x80670de0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE
      \DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xf9c50610     long  kbdclass!GsDriverEntry+0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : (null)
   +0x038 MajorFunction    : [28] 0xf9c4cdd0     long  kbdclass!KeyboardClassCreate+0
```

WinDbg 3 (display next driver object)

```
kd> !devobj 814e7020
Device object (814e7020) is for:
  \Driver\i8042prt DriverObject 814ea410
Current Irp 00000000 RefCount 0 Type 00000027 Flags 00002004
DevExt 814e70d8 DevObjExt 814e7368
ExtensionFlags (0000000000)
AttachedDevice (Upper) 814e7d28 \Driver\Kbdclass
AttachedTo (Lower) 8167c030 \Driver\ACPI
Device queue is not busy.
kd> dt nt!_DRIVER_OBJECT 814ea410
   +0x000 Type           : 4
   +0x002 Size           : 168
   +0x004 DeviceObject     : 0x817dda40 _DEVICE_OBJECT
   +0x008 Flags          : 0x12
   +0x00c DriverStart      : 0xf9a2c000
   +0x010 DriverSize       : 0xcd00
   +0x014 DriverSection    : 0x81973070
   +0x018 DriverExtension  : 0x814ea4b8 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\i8042prt"
   +0x024 HardwareDatabase : 0x80670de0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE
     \DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xf9a35285     long  i8042prt!GsDriverEntry+0
   +0x030 DriverStartIo    : 0xf9a2c910     void  i8042prt!I8xStartIo+0
   +0x034 DriverUnload     : 0xf9a32eb6     void  i8042prt!I8xUnload+0
   +0x038 MajorFunction    : [28] 0xf9a2faa6     long  i8042prt!I8xCreate+0
```

WinDbg 4 (print IRP table)

```
kd> dps 814ea410+0x38 L1C
814ea448  f9a2faa6 i8042prt!I8xCreate
814ea44c  804f355a nt!IopInvalidDeviceRequest
814ea450  f9a32e18 i8042prt!I8xClose
814ea454  804f355a nt!IopInvalidDeviceRequest
814ea458  804f355a nt!IopInvalidDeviceRequest
814ea45c  804f355a nt!IopInvalidDeviceRequest
814ea460  804f355a nt!IopInvalidDeviceRequest
814ea464  804f355a nt!IopInvalidDeviceRequest
814ea468  804f355a nt!IopInvalidDeviceRequest
814ea46c  f9a2e1f9 i8042prt!I8xFlush
814ea470  804f355a nt!IopInvalidDeviceRequest
814ea474  804f355a nt!IopInvalidDeviceRequest
814ea478  804f355a nt!IopInvalidDeviceRequest
814ea47c  804f355a nt!IopInvalidDeviceRequest
814ea480  f9a32e4b i8042prt!I8xDeviceControl
814ea484  f9a2c836 i8042prt!I8xInternalDeviceControl
814ea488  804f355a nt!IopInvalidDeviceRequest
814ea48c  804f355a nt!IopInvalidDeviceRequest
814ea490  804f355a nt!IopInvalidDeviceRequest
814ea494  804f355a nt!IopInvalidDeviceRequest
814ea498  804f355a nt!IopInvalidDeviceRequest
814ea49c  804f355a nt!IopInvalidDeviceRequest
814ea4a0  f9a337ea i8042prt!I8xPower
814ea4a4  f9a2fa59 i8042prt!I8xSystemControl
814ea4a8  804f355a nt!IopInvalidDeviceRequest
814ea4ac  804f355a nt!IopInvalidDeviceRequest
814ea4b0  804f355a nt!IopInvalidDeviceRequest
814ea4b4  f9a2f990 i8042prt!I8xPnP
```

dps = **d**isplay **p**rocessor-sized pointer (meaning it decides whether it should be 16-64 bits), as a pointer to a **s**ymbol

dds = **d**isplay **d**word as a pointer to a **s**ymbol

Stuxnet IRP filtering

http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

"The driver scans the following filesystem driver objects:

\FileSystem\ntfs
\FileSystem\fastfat
\FileSystem\cdfs

A new device object is created by Stuxnet and attached to the device chain for each device object managed by these driver objects. The MrxNet.sys driver will manage this driver object. By inserting such objects, Stuxnet is able to intercept IRP requests (example: writes, reads, to devices NTFS, FAT or CD-ROM devices)."

Stuxnet IRP filtering 2

- "The driver monitors 'directory control' IRPs, in particular 'directory query' notifications. Such IRPs are sent to the device when a user program is browsing a directory, and requests the list of files it contains for instance."

He4Hook code

(from kirpfilter.cpp)

```
NTSTATUS KIrpFilter::IrpHandler(…){
…
if (
     dwMajorFn == IRP_MJ_DIRECTORY_CONTROL
     &&
     dwMinorFn == IRP_MN_QUERY_DIRECTORY
    )
    {
    NtStatus = OnQueryDirectory(pIrp, pIrpStack, pDrvInfo);
    }
…
}
```

Direct Kernel Object Manipulation (DKOM)

(It's a joke see? Since Distributed Component Object Model - DCOM - is a MS technology?)

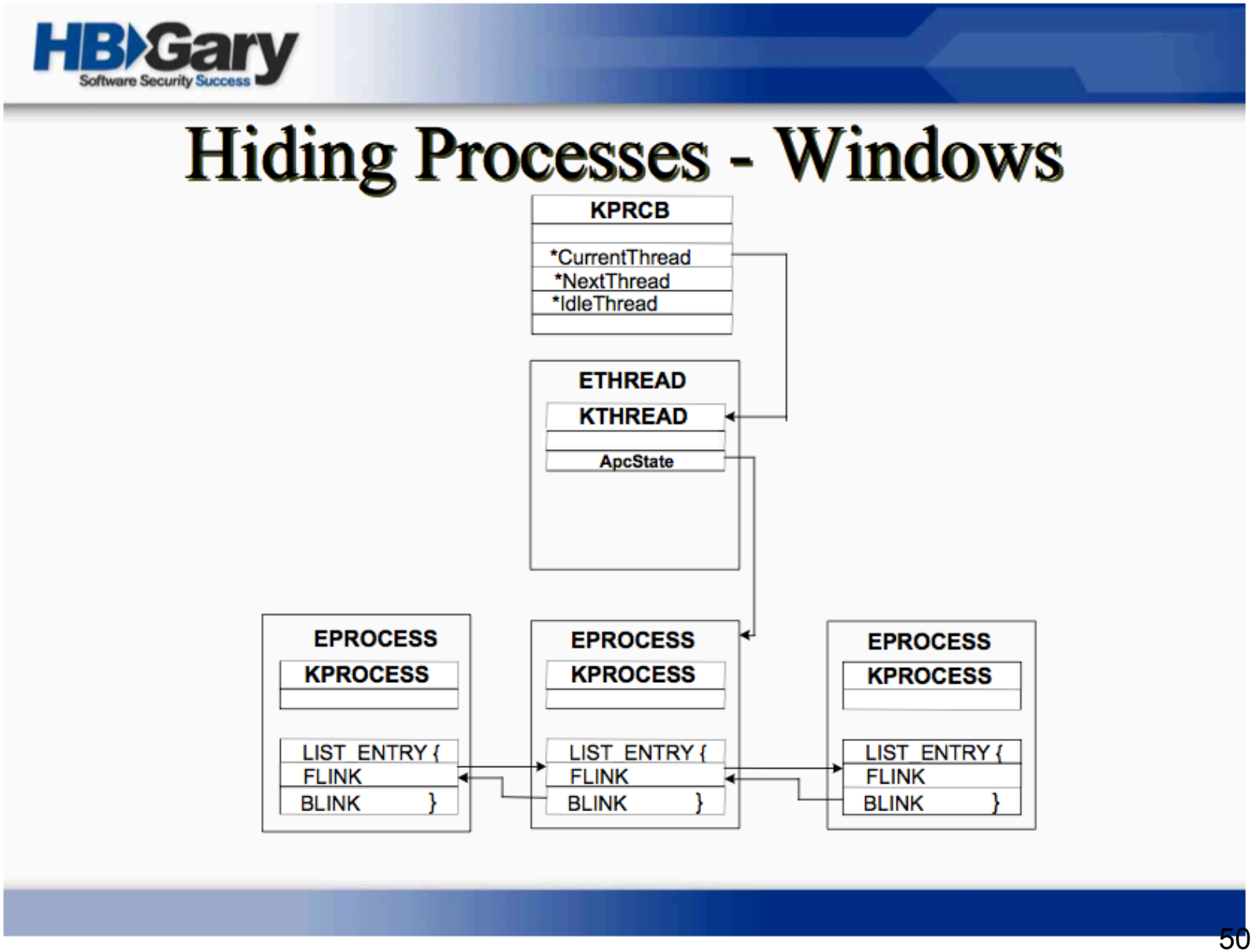
- Introduced by Jamie Butler in the FU rootkit. http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf
- Recognized the prevailing technique of hooking was easily detected, so wrote a detector ("VICE - Catch the hookers!" ;))
- DKOM perpetuates the arms race and shows the importance of information asymmetry for rootkits. The attacker reverse engineers a component he finds to be relevant to his goal. Then, having more understanding of the system than the defender, will likely succeed in having the manipulation go undetected.

Book page 428, 436

Canonical DKOM

- Exploit the fact that the lists which taskmgr.exe or top (on *nix) consult to find running processes, are not the lists that the scheduler uses. Therefore things can be removed from those lists but they will still get to run.
- Also manipulates security tokens to elevate process or user privileges. Is DKOM, but isn't about hiding. But just like hooking does not automatically imply rootkit, DKOM can be used for non-rootkit ends

Process Linked List Before DKOM

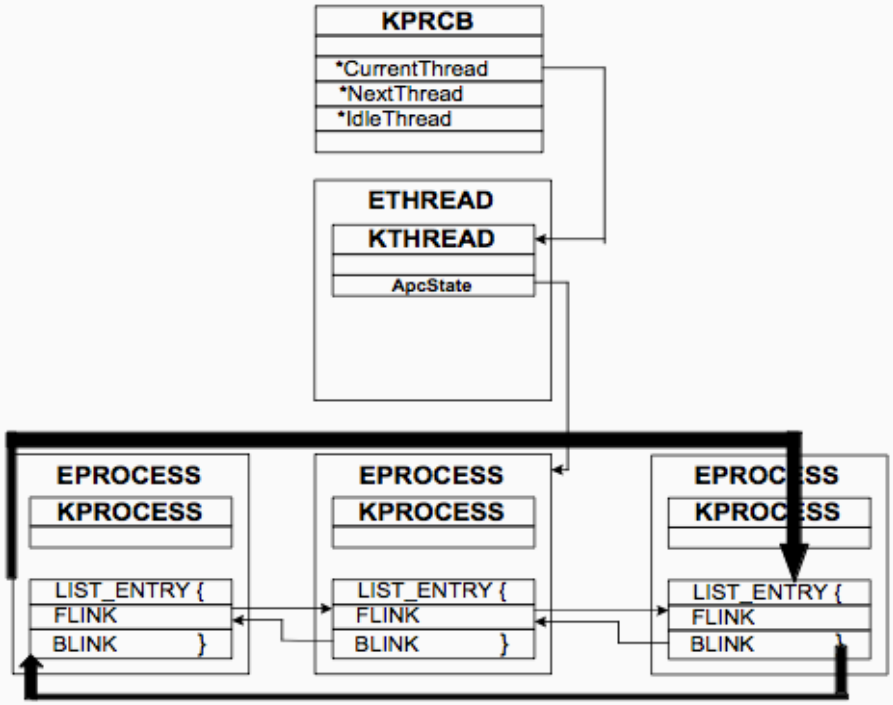


From: http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

Process Linked List After DKOM

HBGary
Software Security Success

Hiding Processes - Windows



From: http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

Detecting DKOM

- Different tools used different means to detect FU's process hiding.
- F-Secure BlackLight used a bruteforce where it calls OpenProcess() on all possible PIDs (which behind the scenes is just consulting PspCidTable, which has a handle for every open process. These handles are not hidden as part of DKOM.) It then calls CreateToolhelp32Snapshot() as another more traditional way to get a list of processes. Any discrepancy in the lists is deemed a hidden process.
- So Peter Silberman introduced FUTo (http://uninformed.org/index.cgi?v=3&a=7&t=sumry) which bypassed BlackLight by manipulating the PspCidTable.
- Then Butler and Silberman put out RAIDE to detect the FUTo hiding too (using memory signature searching I believe)
- Klister by Rutkowska walked the list that the scheduler uses
- This is an example of "cross view detection"

FWIW: turns out…

This change:

For the record I emailed the GMER author and he said:

"The 32-bit hex number that is after PID[0] is the EPROCESS structure pointer. In this case GMER cannot identify the name of process."

If you look at my omega.bat in the tiddlywiki install proceedure, you will see that I'm using fu.exe to hide pid 4 ("system"). Sometimes system is called pid 0, sometimes pid 4 (on XP)

OS-Provided Callbacks

(The ones we want to highlight for the moment anyway. Go to the links and go up one level on the side bar to find more.)

- On registry actions:
  - CmRegisterCallback{Ex}
  - http://msdn.microsoft.com/en-us/library/ff541918(v=vs.85).aspx
- On process creation/deletion:
  - PsSetCreateProcessNotifyRoutine{Ex}
  - http://msdn.microsoft.com/en-us/library/ff559951(v=VS.85).aspx
- On thread creation/deletion:
  - PsSetCreateThreadNotifyRoutine{Ex}
  - http://msdn.microsoft.com/en-us/library/ff559954(v=vs.85).aspx
- On image load (e.g. DLL, EXE, SYS mapped into memory, imports resolved, but entry point not yet called):
  - PsSetLoadImageNotifyRoutine
  - http://msdn.microsoft.com/en-us/library/ff559957(v=VS.85).aspx
- Filesystem becoming active (to attach to with a filesystem filter driver):
  - IoRegisterFsRegistrationChange
  - http://msdn.microsoft.com/en-us/library/ff551037(v=vs.85).aspx
- System Shutdown:
  - IoRegisterShutdownNotification
  - http://msdn.microsoft.com/en-us/library/ff549541.aspx

Some example rootkit use of callbacks

- He4Hook - PoC
  - Sets a callback with PsSetCreateThreadNotifyRoutine() (see the source)
- FUTo - PoC
  - PsSetCreateProcessNotifyRoutine() (see source)
- Black Energy 2, Rustock
  - http://code.google.com/p/volatility/wiki/CommandReference#notifyroutines
  - Black Energy Thread notify, Rustock Process notify
- HybridHook - PoC
  - Sets a callback with PsSetLoadImageNotifyRoutine() and does IAT hooking at load time (see the source)
- TDSS/TDL3
  - http://www.prevx.com/blog/139/Tdss-rootkit-silently-owns-the-net.html
  - PsSetLoadImageNotifyRoutine() to inject DLLs
- Stuxnet
  - http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf
  - "The driver also registers to a filesystem registration callback routine in order to hook newly created filesystem objects on the fly."

Listing registered callbacks in WinDbg

- ~~http://analyze-v.com/?p=746 - process/memory image load (**PsSetCreateProcessNotifyRoutine[Ex]/PsSetImageLoadNotifyRoutine)**~~
- ~~http://analyze-v.com/?p=756 - registry callbacks(**CmRegisterCallback[Ex])**~~
- **Here comes a new challenger! Hadoken!**
- http://www.moonsols.com/2011/02/17/global-windows-callbacks-and-windbg/

```
kd> $$>a<c:\pscallbacks.wbs
**********************************************
* This command brought to you by Analyze-v.com *
**********************************************


************************************
* Printing image load callbacks... *
************************************
********************************************
* Printing process notification callbacks... *
********************************************
814ec008 ff2508605c81    jmp     dword ptr ds:[815C6008h]
```

Listing registered callbacks

- Newest Virus Blok Ada anti-rootkit has fairly comprehensive coverage.

NewTewYew - Master Boot Record (MBR)-infecting rootkits aka "Bootkits"

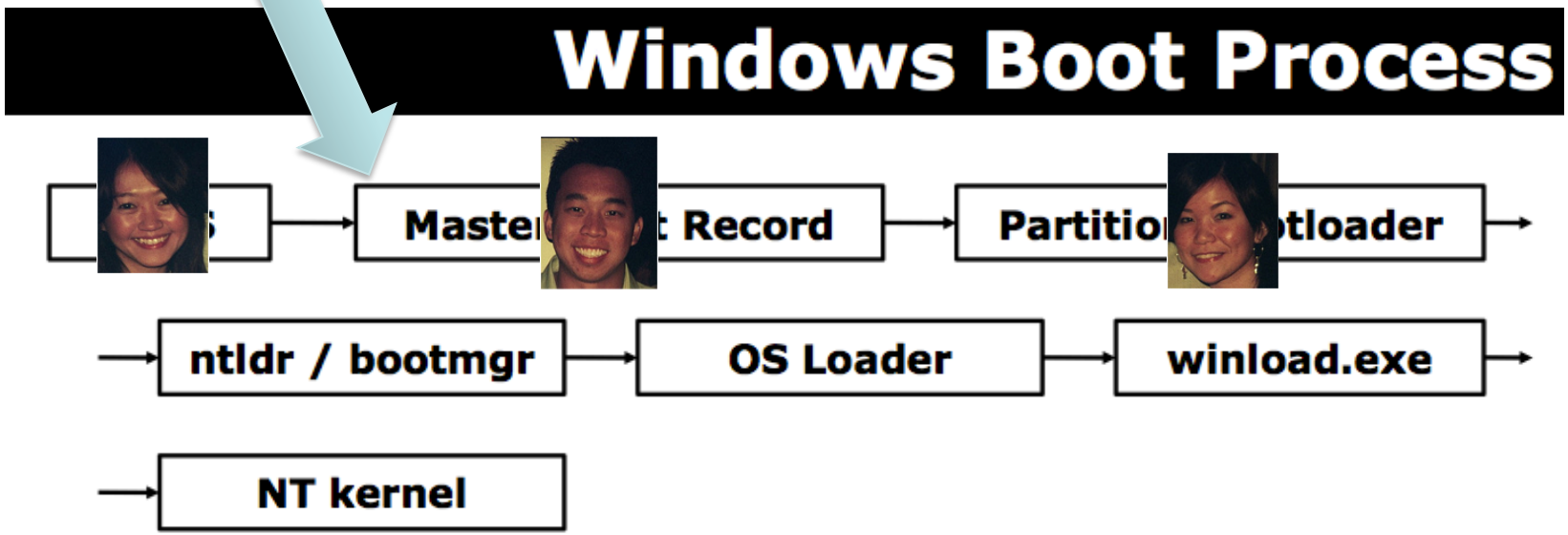
- eEye Bootroot - 2005
  - Derek Soeder and Ryan Permeh
  - From the readme.txt "September 20, 2005: Scott Tenaglia provided a NASM port of the source, included as "ebrknasm.asm". Many folks were not too keen on the MASM requirement, so this is a big boon to BootRootKit users at large. Thanks Scott!"
- VBootkit - 2007, Vbootkit 2 - 2009
  - Nitin Kumar and Vipin Kumar
- Stoned Bootkit - 2009
  - Peter Kleissner

Book page 380

What does an MBR actually look like?

- http://thestarman.narod.ru/asm/mbr/Win2kmbr.htm
- http://thestarman.narod.ru/asm/mbr/VistaMBR.htm

Bootkit Lives here (from disk), but in order to do anything of consequence it has to keep hooking each subsequent thing to keep control.



`Ntldr` = 16-bit stub + `OS Loader` (just binary appended)
Windows Vista splits up `ntldr` into `bootmgr`, `winload.exe` and `winresume.exe`

Windows XP	Windows Vista	Processor Environment
ntldr	bootmgr	Real Mode
OS Loader	OS Loader	Protected Mode
-	winload.exe	Protected Mode
NT kernel	NT kernel	Protected Mode + Paging

8

From http://www.stoned-vienna.com/downloads/Presentation.pdf

eEye Boot Root

- http://www.eeye.com/Resources/Security-Center/Research/Tools/BootRoot
- The first PoC, didn't actually change the MBR on the HD, instead booted from a disk which redirected to the normal boot process, hooking as it went.

VBootKit

- Coined the term "bootkit" for master boot record infecting rootkits
- Vbootkit
  - http://www.blackhat.com/presentations/bh-europe-07/Kumar/Presentation/bh-eu-07-kumar-apr19.pdf
  - First thing supporting Vista

- Vbootkit 2
  - http://conference.hitb.org/hitbsecconf2009dubai/materials/D2T2%20-%20Vipin%20and%20Nitin%20Kumar%20-%20vbootkit%202.0.pdf
  - First thing to support boot subversion on Windows 7 x64
  - Payload includes disabling code signing & kernel patch protection (KPP aka PatchGuard)

Stoned Bootkit

- http://www.stoned-vienna.com
- Basically a weaponized bootkit…so, surprise, surprise, it got used in real malware, and the German cops came knocking (illegal to distribute "hacking tools" in Germany). So now he doesn't distribute the full thing.
- Highly module to support many possible payloads
- Has a customization to work in the presence of TrueCrypt

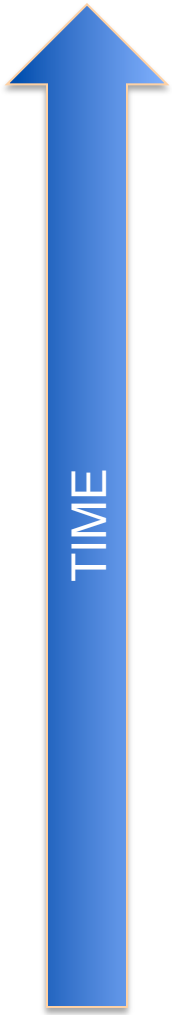
bootkits in the wild

- http://www2.gmer.net/mbr/
  - Good dissection (w comparison to eEye boot root)
  - Also shows adding one nop to asm bypassed MS, Kaspersky, F-Secure, and Sophos detection circa 2008
- Mebroot w/ Torpig/Sinowal
  - http://www.symantec.com/security_response/writeup.jsp?docid=2008-010718-3448-99
  - http://www.f-secure.com/weblog/archives/00001393.html
    - IRP hooking. How droll. How easy peasy ☺
- TDSS/TDL3
  - http://www.prevx.com/blog/139/Tdss-rootkit-silently-owns-the-net.html
    - IRP hooking. How droll. How easy peasy ☺

Detecting bootkits

- http://www2.gmer.net/mbr/mbr.exe (though I think GMER has that mostly built in now)
- TrendMicro RootkitBuster claims to have removal
- Remove with "fixmbr" command from a windows install disk recovery mode
- Turn on your damn TPM! This, and bios subversion, are the whole reason we have trusted boot!
  - Get insPeCtoR from Corey Kallenberg
  - or

An "Integrity Measured" Boot Process



OS

- Measure and store (PCR-4) OS components, other TCB-relevant stuff
- System runs normally!

OS Loader

- Measure OS Components
- Append PCR-3 with measurement
- Pass control to OS

BIOS

- Measure Option ROMs
- Append PCR-1 with measurement--pass control, get it back
- Measure OS Loader, append PCR-2, pass control

CRTM

- Core Root of Trust for Measurement, (e.g. BIOS block)
- Measure itself and BIOS
- Append PCR-0 with hash of measurement

Hardware

- Clear PCRs—not really configurable

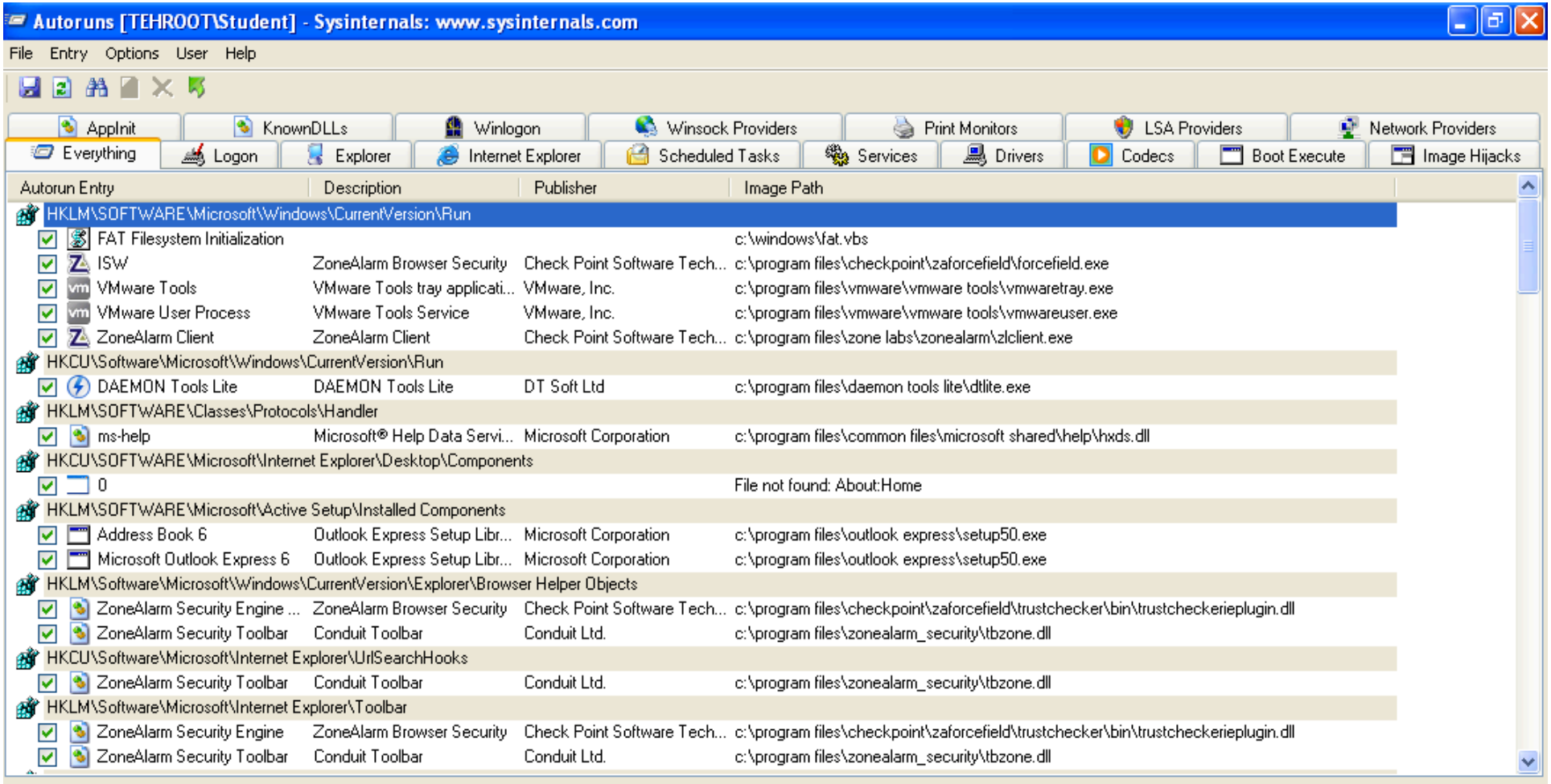
Loading code into kernel

- Service Control Manager (SCM)
  - Leaves registry footprint
- ZwSetSystemInformation()
- - http://seclists.org/bugtraq/2000/Aug/408
- - http://www.nvlabs.in/archives/6-Loading-drivers-and-Native-applications-from-kernel-mode,-without-touching-registry.html
- ZwLoadDriver()
- http://www.codeproject.com/KB/system/DLoad.aspx
  - Uses SCM, ZwSetSystemInformation, ZwLoadDriver
- Windows < Vista used to be able to access \Device \PhysicalMemory
  - http://www.phrack.com/issues.html?issue=59&id=16

Book page 198

Autoruns

- Sysinternals tool to show the various places on the system that are set to automatically load extra code (either on boot, or when something else is loaded)
- http://technet.microsoft.com/en-us/sysinternals/bb963902
- Recent article "Analyzing a Stuxnet Infection with the Sysinternals Tools, Part 1"
  - http://blogs.technet.com/b/markrussinovich/archive/2011/03/30/3416253.aspx


Autoruns [TEHROOT\Student] - Sysinternals: www.sysinternals.com
File Entry Options User Help
AppInit KnownDLLs Winlogon Winsock Providers Print Monitors LSA Providers Network Providers
Everything Logon Explorer Internet Explorer Scheduled Tasks Services Drivers Codecs Boot Execute Image Hijacks
Autorun Entry Description Publisher Image Path
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
FAT Filesystem Initialization c:\windows\fat.vbs
ISW ZoneAlarm Browser Security Check Point Software Tech... c:\program files\checkpoint\zaforcefield\forcefield.exe
VMware Tools VMware Tools tray applicati... VMware, Inc. c:\program files\vmware\vmware tools\vmwaretray.exe
VMware User Process VMware Tools Service VMware, Inc. c:\program files\vmware\vmware tools\vmwareuser.exe
ZoneAlarm Client ZoneAlarm Client Check Point Software Tech... c:\program files\zone labs\zonealarm\zlclient.exe
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
DAEMON Tools Lite DAEMON Tools Lite DT Soft Ltd c:\program files\daemon tools lite\dtlite.exe
HKLM\SOFTWARE\Classes\Protocols\Handler
ms-help Microsoft® Help Data Servi... Microsoft Corporation c:\program files\common files\microsoft shared\help\hxds.dll
HKCU\SOFTWARE\Microsoft\Internet Explorer\Desktop\Components
0 File not found: About:Home
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components
Address Book 6 Outlook Express Setup Libr... Microsoft Corporation c:\program files\outlook express\setup50.exe
Microsoft Outlook Express 6 Outlook Express Setup Libr... Microsoft Corporation c:\program files\outlook express\setup50.exe
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
ZoneAlarm Security Engine ... ZoneAlarm Browser Security Check Point Software Tech... c:\program files\checkpoint\zaforcefield\trustchecker\bin\trustcheckerieplugin.dll
ZoneAlarm Security Toolbar Conduit Toolbar Conduit Ltd. c:\program files\zonealarm_security\tbzone.dll
HKCU\Software\Microsoft\Internet Explorer\UrlSearchHooks
ZoneAlarm Security Toolbar Conduit Toolbar Conduit Ltd. c:\program files\zonealarm_security\tbzone.dll
HKLM\Software\Microsoft\Internet Explorer\Toolbar
ZoneAlarm Security Engine ZoneAlarm Browser Security Check Point Software Tech... c:\program files\checkpoint\zaforcefield\trustchecker\bin\trustcheckerieplugin.dll
ZoneAlarm Security Toolbar Conduit Toolbar Conduit Ltd. c:\program files\zonealarm_security\tbzone.dll