Malware Analysis Part 3

Kernel Level Malware and Analysis

CS-577-A
Jun Xu

Credits of many slides belong to
[1] https://samsclass.info/126/ppt/ch10_2019.pdf
[2] https://www.blackhat.com/docs/asia-16/materials/asia-16-Guo-NumChecker-A-System-Approach-For-Kernel-Rootkit-Detection-And-Identification.pdf
[3] https://www.terena.org/activities/tf-csirt/meeting27/oesterberg-rootkits.pdf

Focus Today

- Learn about malware running in the kernel mode
- Learn about basic ideas of analyzing kernel malware

Rootkit Basics

- Rootkits modify the internal functionality of the OS to conceal themselves
  - Hide processes, network connections, and other resources from running programs
  - Difficult for antivirus, administrators, and security analysts to discover their malicious activity

Different types of rootkits

- Ring 3 (User-mode)
- Ring 0 (Kernel-mode)
- Hardware/Firmware based
- Virtualization based

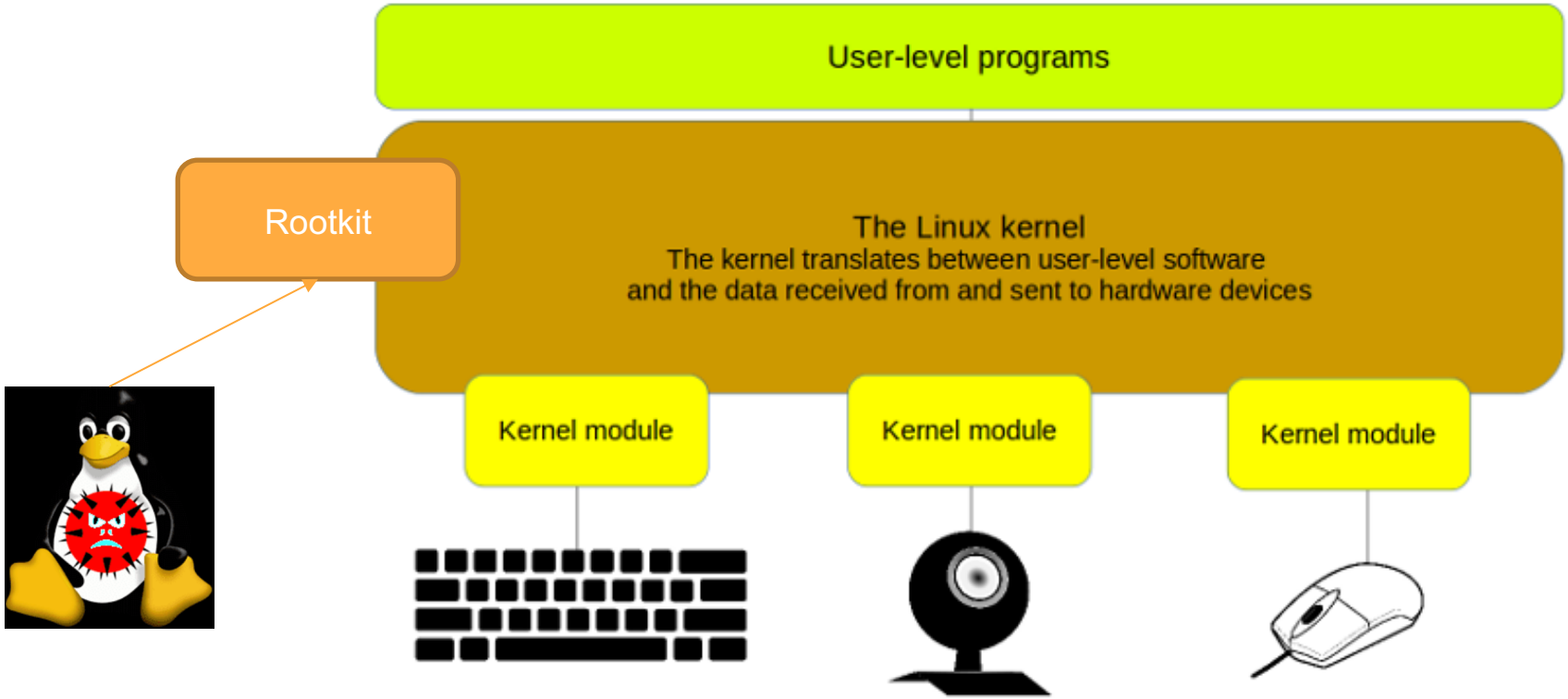
Different types of rootkit techniques

Basic types of rootkit techniques used:

- Hooking
- Injecting
- Unlinking

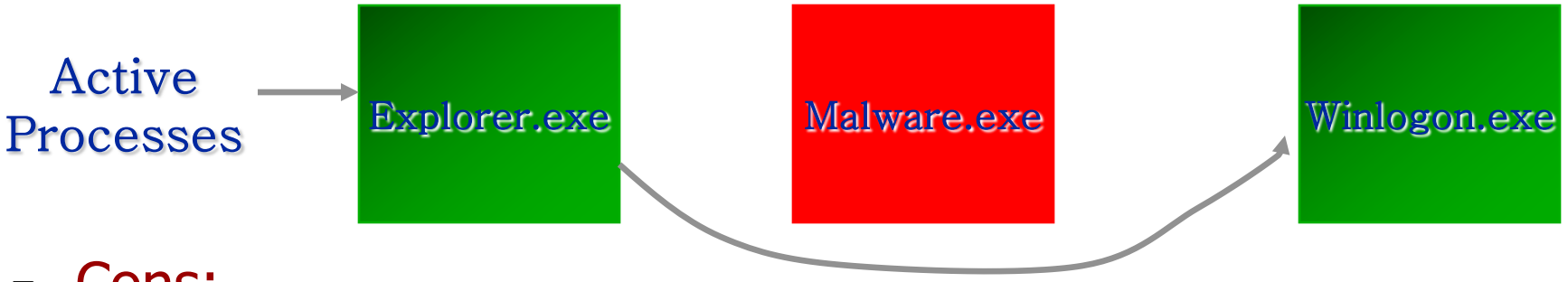
Let’s see some examples at the kernel level

Kernel-Mode Module Installation



Kernel-Mode Data Structure Manipulation

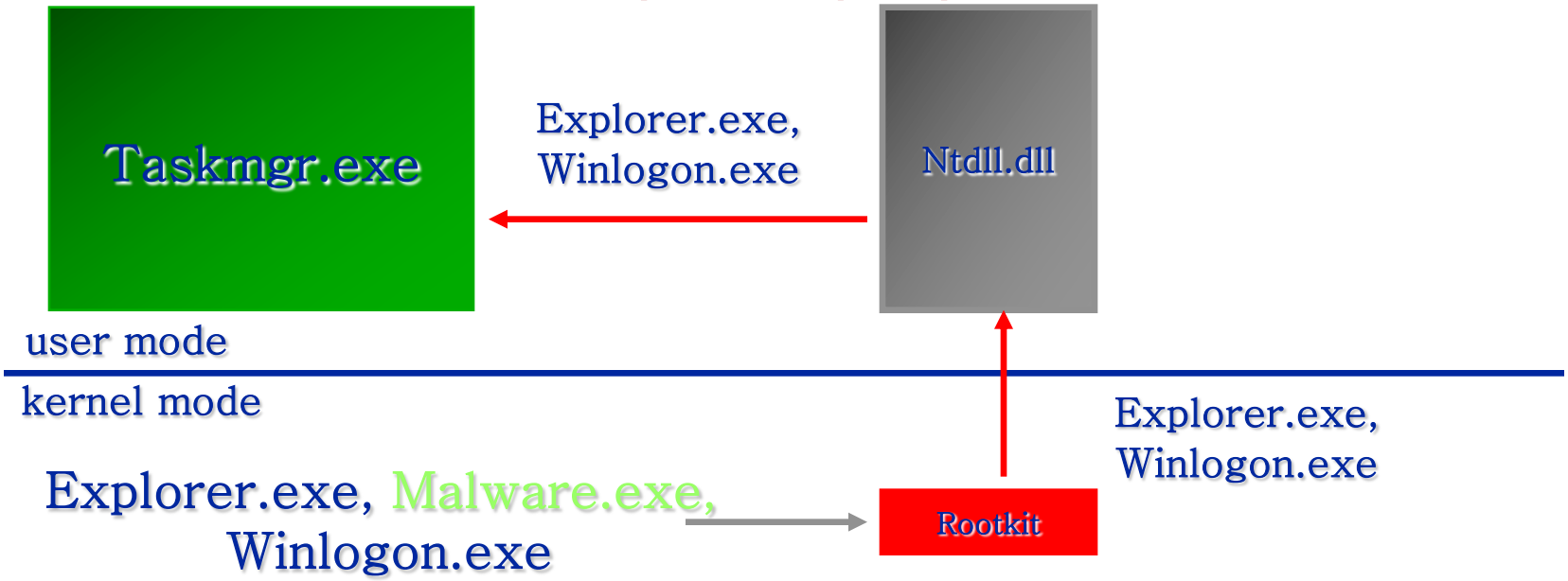
- Also called Direct Kernel Object Manipulation (DKOM)
- Attacks active process data structure
  - Query API doesn't see the process
  - Kernel still schedules process' threads



- Cons:
  - Requires admin privilege to install
  - Can cause crashes
  - Detection already developed
- Pro: more advanced variations possible
- Example: FU

Kernel-Mode API Filtering

- Attack kernel-mode system query APIs



- Cons:
  - Requires admin privilege to install
  - Difficult to write
- Pro: very thorough cloak
- Example: NT Rootkit

System Service Descriptor Table (SSDT)

- Used internally by Microsoft
  - To look up function calls into the kernel
  - Not normally used by third-party applications or drivers
- Only three ways for user space to access kernel code
  - **SYSCALL**
  - **SYSENTER**
  - **INT 0x2E**

SYSENTER

- Used by modern versions of Windows
  - Function code stored in EAX register
- More info about the three ways to call kernel code is in links Ch 10j and 10k

Example from ntdll.dll

Example 11-11. Code for `NtCreateFile` function

```
7C90D682 mov     eax, 25h        ; NtCreateFile
7C90D687  mov     edx, 7FFE0300h
7C90D68C  call    dword ptr [edx]
7C90D68E  retn    2Ch
```

The call to `dword ptr[edx]` will go to the following instructions:

```
7c90eb8b 8bd4  mov     edx,esp
7c90eb8d 0f34  sysenter
```

- EAX set to 0x25
- Stack pointer saved in EDX
- SYSENTER is called

SSDT Table Entries

Example 11-12. Several entries of the SSDT table showing `NtCreateFile`

```
 SSDT[0x22] = 805b28bc (NtCreateaDirectoryObject)
 SSDT[0x23] = 80603be0 (NtCreateEvent)
 SSDT[0x24] = 8060be48 (NtCreateEventPair)
1SSDT[0x25] = 8056d3ca (NtCreateFile)
 SSDT[0x26] = 8056bc5c (NtCreateIoCompletion)
 SSDT[0x27] = 805ca3ca (NtCreateJobObject)
```

- Rootkit changes the values in the SSDT so rootkit code is called instead of the intended function
- 0x25 would be changed to a malicious driver's function

Hooking NtCreateFile

- Rootkit calls the original NtCreateFile, then removes files it wants to hide
  - This prevents applications from getting a handle to the file
- Hooking **`NtCreateFile`** alone won't hide a file from DIR, however

Rootkit Analysis in Practice

- Simplest way to detect SSDT hooking
  - Just look at the SSDT
  - Look for values that are unreasonable
  - In this case, *ntoskrnl.exe* starts at address 804d7000 and ends at 806cd580
  - *ntoskrnl.exe* is the Kernel!
- **lm m nt**
  - Lists modules matching "nt" (Kernel modules)
  - Shows the SSDT table (not in Win 2008 in LiveKD)

SSDT Table

Example 11-13. A sample SSDT table with one entry overwritten by a rootkit

```
kd> lm m nt
...
8050122c  805c9928 805c98d8 8060aea6 805aa334
8050123c  8060a4be 8059cbbc 805a4786 805cb406
8050124c  804feed0 8060b5c4 8056ae64 805343f2
8050125c  80603b90 805b09c0 805e9694 80618a56
8050126c  805edb86 80598e34 80618caa 805986e6
8050127c  805401f0 80636c9c 805b28bc 80603be0
8050128c  8060be48 1f7ad94a4 8056bc5c 805ca3ca
8050129c  805ca102 80618e86 8056d4d8 8060c240
805012ac  8056d404 8059fba6 80599202 805c5f8e
```

- Marked entry is hooked
- To identify it, examine a clean system's SSDT

Finding the Malicious Driver

- **lm**
  - Lists open modules
  - In the kernel, they are all drivers

Example 11-14. Using the `lm` command to find which driver contains a particular address

```
kd>lm
...
f7ac7000 f7ac8580   intelide   (deferred)
f7ac9000 f7aca700   dmload     (deferred)
f7ad9000 f7ada680   Rootkit    (deferred)
f7aed000 f7aee280   vmmouse    (deferred)
...
```

Example 11-16. Listing of the rootkit hook function

```
000104A4  mov     edi, edi
000104A6  push    ebp
000104A7  mov     ebp, esp
000104A9  push    [ebp+arg_8]
000104AC  call    1sub_10486
000104B1  test    eax, eax
000104B3  jz      short loc_104BB
000104B5  pop     ebp
000104B6  jmp     NtCreateFile
000104BB ----------------------------
000104BB                  ; CODE XREF: sub_104A4+F j
000104BB  mov     eax, 0C0000034h
000104C0  pop     ebp
000104C1  retn    2Ch
```

The hook function jumps to the original `NtCreateFile` function for some requests and returns to 0xC0000034 for others. The value `0xC0000034` corresponds to `STATUS_OBJECT_NAME_NOT_FOUND`. The call at 1 contains

Setting Up Kernel Debugging

VMware

- In the virtual machine, enable kernel debugging
- Configure a virtual serial port between VM and host
- Configure WinDbg on the host machine

Boot.ini

- We can activate kernel debugging by editing Boot.ini
- But Microsoft abandoned that system after Windows XP
- The new system uses **bcdedit**

This is probably all the other things you need:

https://www.instructables.com/How-To-Setup-A-Kernel-Debugger-Over-Your-Network/

Using WinDbg

- Command-Line Commands

Reading from Memory

- d*x* *addressToRead*
- *x* can be
  - da Displays as ASCII text
  - du Displays as Unicode text
  - dd Displays as 32-bit double words
- da 0x401020
  - Shows the ASCII text starting at 0x401020

Editing Memory

- e*x* *addressToWrite dataToWrite*
- *x* can be
  - ea Writes as ASCII text
  - eu Writes as Unicode text
  - ed Writes as 32-bit double words

Using Arithmetic Operators

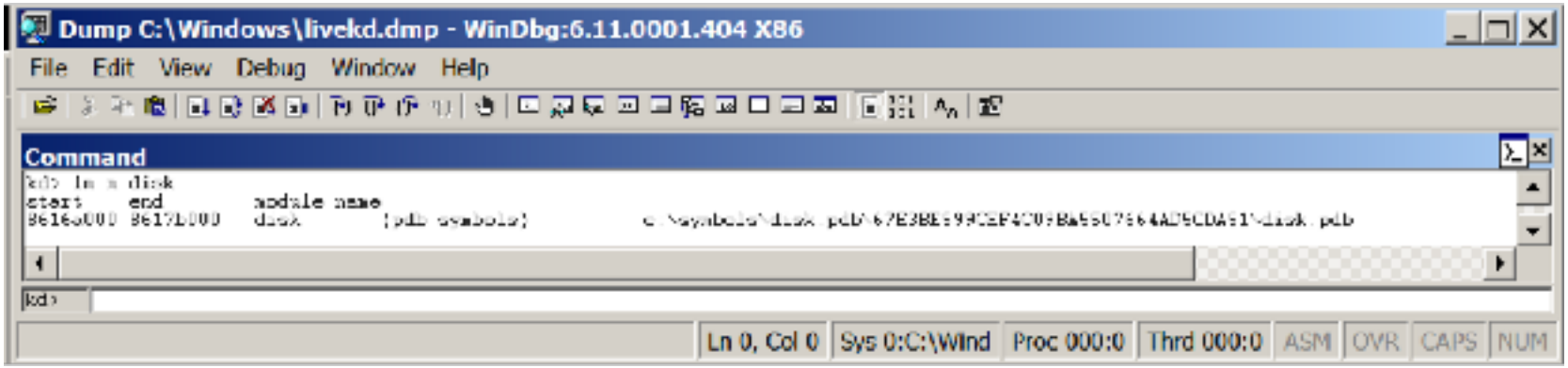
- Usual arithmetic operators + - / *
- **dwo** reveals the value at a 32-bit location pointer
- **du dwo (esp+4)**
  - Shows the first argument for a function, as a wide character string

Setting Breakpoints

- **bp** sets breakpoints
- You can specify an action to be performed when the breakpoint is hit
- **g** tells it to resume running after the action
- **bp GetProcAddress "da dwo(esp+8); g"**
  - Breaks when GetProcAddress is called, prints out the second argument, and then continues
  - The second argument is the function name

Listing Modules

- **lm**
  - Lists all modules loaded into a process
    - Including EXEs and DLLs in user space
    - And the kernel drivers in kernel mode
  - As close as WinDbg gets to a memory map
- **lm m disk**
  - Shows the disk driver



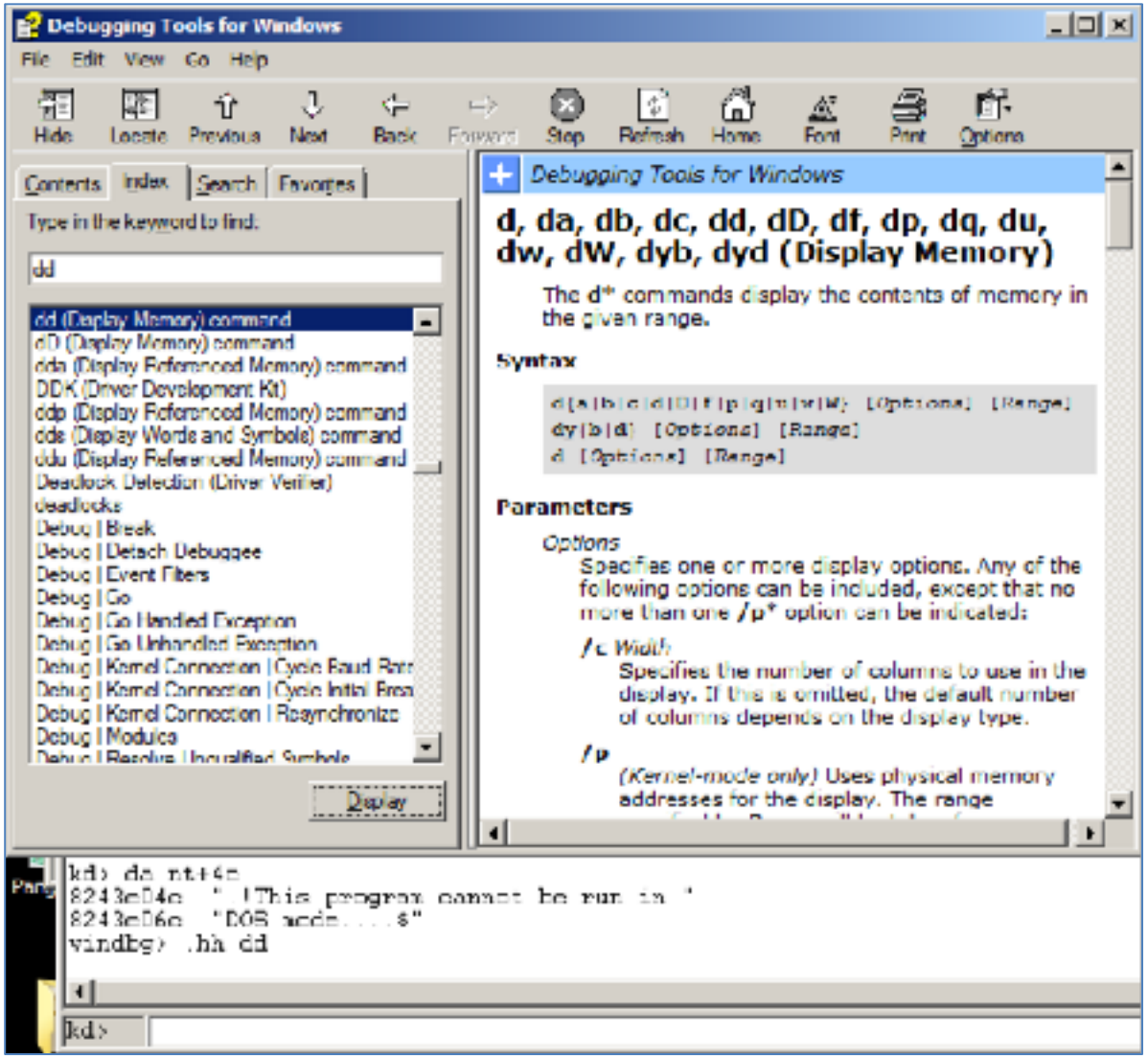
Reading from Memory

- **dd nt**
  - Shows the start of module "nt"
- **dd nt L10**
  - Shows the first 0x10 words of "nt"

```
kd> dd nt
8243e000  00905a4d 00000003 00000004 0000ffff
8243e010  000000b8 00000000 00000040 00000000
8243e020  00000000 00000000 00000000 00000000
8243e030  00000000 00000000 00000000 00000268
8243e040  0eba1f0e cd09b400 4c01b821 685421cd
8243e050  70207369 72676f72 63206d61 6f6e6e61
8243e060  65622074 6e757220 206e6920 20534f44
8243e070  65646f6d 0a0d0d2e 00000024 00000000
kd> dd nt L10
8243e000  00905a4d 00000003 00000004 0000ffff
8243e010  000000b8 00000000 00000040 00000000
8243e020  00000000 00000000 00000000 00000000
8243e030  00000000 00000000 00000000 00000268
```

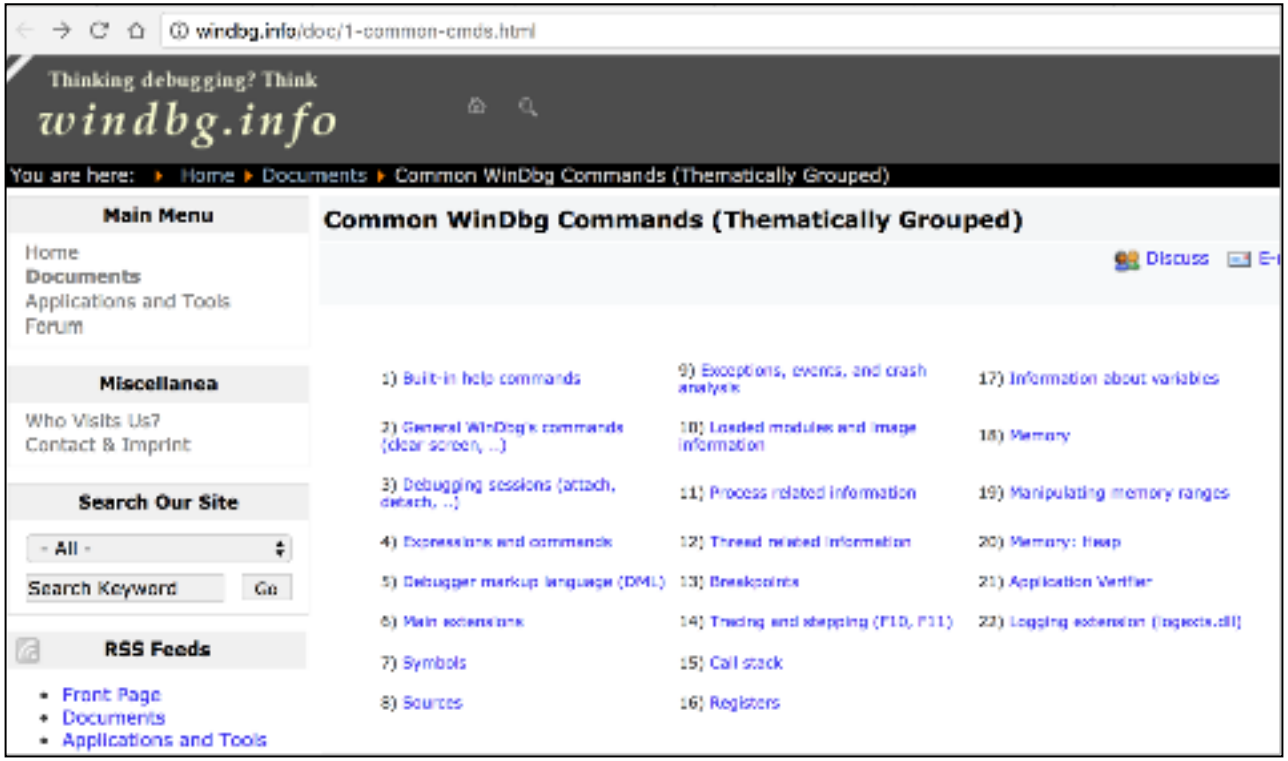
Online Help

- **.hh dd**
  - Shows help about "dd" command
  - But there are no examples



More Commands

- r
  - Dump all registers



Microsoft Symbols

Symbols are Labels

- Including symbols lets you use
  - **MmCreateProcessAddressSpace**
- instead of
  - **0x8050f1a2**

Searching for Symbols

- ***moduleName*!*symbolName***
  - Can be used anywhere an address is expected
- ***moduleName***
  - The EXE, DLL, or SYS filename (without extension)
- ***symbolName***
  - Name associated with the address
- ntoskrnl.exe is an exception, and is named **nt**
  - Ex: **u nt!NtCreateProcess**
    - Unassembles that function (disassembly)

Demo

- Try these
  - **u nt!ntCreateProcess**
  - **u nt!ntCreateProcess L10**
  - **u nt!ntCreateProcess L20**

```
kd> u nt!ntCreateProcess
nt!NtCreateProcess:
826d1f9f 8bff            mov     edi,edi
826d1fa1 55              push    ebp
826d1fa2 8bec            mov     ebp,esp
826d1fa4 33c0            xor     eax,eax
826d1fa6 f6451c01        test    byte ptr [ebp+1Ch],1
826d1faa 7401            je      nt!NtCreateProcess+0xe (826d1fad)
826d1fac 40              inc     eax
826d1fad f6452001        test    byte ptr [ebp+20h],1
```

Deferred Breakpoints

- **bu *newModule*!*exportedFunction***
  - Will set a breakpoint on *exportedFunction* as soon as a module named *newModule* is loaded
- $iment
  - Function that finds the entry point of a module
- **bu $iment(*driverName*)**
  - Breaks on the entry point of the driver before any of the driver's code runs

Searching with x

- You can search for functions or symbols using wildcards
- **x nt!*CreateProcess***
  - Displays exported functions & internal functions

```
0:003> x nt!*CreateProcess*
805c736a nt!NtCreateProcessEx = <no type information>
805c7420 nt!NtCreateProcess = <no type information>
805c6a8c nt!PspCreateProcess = <no type information>
804fe144 nt!ZwCreateProcess = <no type information>
804fe158 nt!ZwCreateProcessEx = <no type information>
8055a300 nt!PspCreateProcessNotifyRoutineCount = <no type information>
805c5e0a nt!PsSetCreateProcessNotifyRoutine = <no type information>
8050f1a2 nt!MmCreateProcessAddressSpace = <no type information>
8055a2e0 nt!PspCreateProcessNotifyRoutine = <no type information>
```

Listing Closest Symbol with **ln**

- Helps in figuring out where a call goes
- **ln *address***
  - First lines show two closest matches
  - Last line shows exact match

```
0:002> ln 805717aa
kd> ln ntreadfile
1 (805717aa)   nt!NtReadFile   |  (80571d38)   nt!NtReadFileScatter
Exact matches:
2     nt!NtReadFile = <no type information>
```

Viewing Structure Information with **dt**

- Microsoft symbols include type information for many structures
  - Including undocumented internal types
  - They are often used by malware
- **dt *moduleName*!*symbolName***
- **dt *moduleName*!*symbolName address***
  - Shows structure with data from *address*

Example 11-2. Viewing type information for a structure

```
0:000> dt nt!_DRIVER_OBJECT
kd> dt nt!_DRIVER_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Int2B
   +0x004 DeviceObject     : Ptr32 _DEVICE_OBJECT
   +0x008 Flags            : Uint4B
1  +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
```

Demo

- Try these
  - **dt nt!_DRIVER_OBJECT**
  - **dt nt!_DEVICE_OBJECT**

```
kd> dt nt!_DEVICE_OBJECT
   +0x000 Type             : Int2B
   +0x002 Size             : Uint2B
   +0x004 ReferenceCount   : Int4B
   +0x008 DriverObject     : Ptr32 _DRIVER_OBJECT
   +0x00c NextDevice       : Ptr32 _DEVICE_OBJECT
   +0x010 AttachedDevice   : Ptr32 _DEVICE_OBJECT
   +0x014 CurrentIrp       : Ptr32 _IRP
   +0x018 Timer            : Ptr32 _IO_TIMER
   +0x01c Flags            : Uint4B
   +0x020 Characteristics  : Uint4B
   +0x024 Vpb              : Ptr32 _VPB
   +0x028 DeviceExtension  : Ptr32 Void
   +0x02c DeviceType       : Uint4B
   +0x030 StackSize        : Char
   +0x034 Queue            : <unnamed-tag>
   +0x05c AlignmentRequirement : Uint4B
   +0x060 DeviceQueue      : _KDEVICE_QUEUE
   +0x074 Dpc              : _KDPC
   +0x094 ActiveThreadCount : Uint4B
   +0x098 SecurityDescriptor : Ptr32 Void
   +0x09c DeviceLock       : _KEVENT
   +0x0ac SectorSize       : Uint2B
   +0x0ae Spare1           : Uint2B
   +0x0b0 DeviceObjectExtension : Ptr32 _DEVOBJ_EXTENSION
   +0x0b4 Reserved         : Ptr32 Void
```

Show Specific Values for the "Beep" Driver

Example 11-3. Overlaying data onto a structure

```
kd> dt nt!_DRIVER_OBJECT 828b2648
   +0x000 Type             :  4
   +0x002 Size             :  168
   +0x004 DeviceObject     :  0x828b0a30 _DEVICE_OBJECT
   +0x008 Flags            :  0x12
   +0x00c DriverStart      :  0xf7adb000
   +0x010 DriverSize       :  0x1080
   +0x014 DriverSection    :  0x82ad8d78
   +0x018 DriverExtension  :  0x828b26f0 _DRIVER_EXTENSION
   +0x01c DriverName       :  _UNICODE_STRING "\Driver\Beep"
   +0x024 HardwareDatabase :  0x80670ae0 _UNICODE_STRING
"\REGISTRY\MACHINE\
HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   :  (null)
   +0x02c DriverInit       :  0xf7adb66c     long  Beep!DriverEntry+0
   +0x030 DriverStartIo    :  0xf7adb51a     void  Beep!BeepStartIo+0
   +0x034 DriverUnload     :  0xf7adb620     void  Beep!BeepUnload+0
   +0x038 MajorFunction    :  [28] 0xf7adb46a     long  Beep!BeepOpen+0
```

Initialization Function

- The **DriverInit** function is called first when a driver is loaded
  - See labelled line in previous slide
- Malware will sometimes place its entire malicious payload in this function

Configuring Windows Symbols

- If your debugging machine is connected to an always-on broadband link, you can configure WinDbg to automatically download symbols from Microsoft as needed
- They are cached locally
- **File**, **Symbol File Path**
  - **`SRC*c:\websymbols*http://msdl.microsoft.com/download/symbols`**

Manually Downloading Symbols

Kernel-Mode Code

- Set WinDbg to Verbose mode (View, Verbose Output)
  - Doesn't work with LiveKD
- You'll see every kernel module that loads
- Kernel modules are not loaded or unloaded often
  - Any loads are suspicious

In the following example, we see that the *FileWriter.sys* driver has been loaded in the kernel debugging window. Likely, this is the malicious driver.

```
ModLoad: f7b0d000 f7b0e780   FileWriter.sys
```

NOTE

When using VMware for kernel debugging, you will see KMixer.sys *frequently loaded and unloaded. This is normal and not associated with any malicious activity.*

Kernel-Mode Code

- **!drvobj** command shows driver object

Example 11-7. Viewing a driver object for a loaded driver

```
kd> !drvobj FileWriter
Driver object (1827e3698) is for:
Loading symbols for f7b0d000   FileWriter.sys ->   FileWriter.sys
*** ERROR: Module load completed but symbols could not be loaded for
FileWriter.sys
 \Driver\FileWriter
Driver Extension List: (id , addr)

Device Object list:
826eb030
```

Kernel-Mode Code

- **dt** command shows structure

Example 11-8. Viewing a device object in the kernel

```
kd>dt nt!_DRIVER_OBJECT 0x827e3698
nt!_DRIVER_OBJECT
   +0x000 Type             : 4
   +0x002 Size             : 168
   +0x004 DeviceObject     : 0x826eb030 _DEVICE_OBJECT
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xf7b0d000
   +0x010 DriverSize       : 0x1780
   +0x014 DriverSection    : 0x828006a8
   +0x018 DriverExtension  : 0x827e3740 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\FileWriter"
   +0x024 HardwareDatabase : 0x8066ecd8 _UNICODE_STRING
"\REGISTRY\MACHINE\
                             HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xf7b0dfcd     long  +0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : 0xf7b0da2a     void  +0
   +0x038 MajorFunction    : [28] 0xf7b0da06     long  +0
```

Kernel-Mode Filenames

- Tracing this function, it eventually creates this file
  - \DosDevices\C:\secretfile.txt
- This is a *fully qualified object name*
  - Identifies the root device, usually \DosDevices

Finding Driver Objects

- Applications work with *devices*, not drivers
- Look at user-space application to identify the interesting *device object*
- Use *device object* in User Mode to find *driver object* in Kernel Mode
- Use **!devobj** to find out more about the *device object*
- Use **!devhandles** to find application that use the driver

Rootkit Analysis in Practice

- Simplest way to detect SSDT hooking
  - Just look at the SSDT
  - Look for values that are unreasonable
  - In this case, *ntoskrnl.exe* starts at address 804d7000 and ends at 806cd580
  - *ntoskrnl.exe* is the Kernel!
- **lm m nt**
  - Lists modules matching "nt" (Kernel modules)
  - Shows the SSDT table (not in Win 2008 in LiveKD)

SSDT Table

Example 11-13. A sample SSDT table with one entry overwritten by a rootkit

```
kd> lm m nt
...
8050122c  805c9928 805c98d8 8060aea6 805aa334
8050123c  8060a4be 8059cbbc 805a4786 805cb406
8050124c  804feed0 8060b5c4 8056ae64 805343f2
8050125c  80603b90 805b09c0 805e9694 80618a56
8050126c  805edb86 80598e34 80618caa 805986e6
8050127c  805401f0 80636c9c 805b28bc 80603be0
8050128c  8060be48 1f7ad94a4 8056bc5c 805ca3ca
8050129c  805ca102 80618e86 8056d4d8 8060c240
805012ac  8056d404 8059fba6 80599202 805c5f8e
```

- Marked entry is hooked
- To identify it, examine a clean system's SSDT

Finding the Malicious Driver

- **lm**
  - Lists open modules
  - In the kernel, they are all drivers

Example 11-14. Using the `lm` command to find which driver contains a particular address

```
kd>lm
...
f7ac7000 f7ac8580   intelide   (deferred)
f7ac9000 f7aca700   dmload     (deferred)
f7ad9000 f7ada680   Rootkit    (deferred)
f7aed000 f7aee280   vmmouse    (deferred)
...
```

Example 11-16. Listing of the rootkit hook function

```
000104A4  mov     edi, edi
000104A6  push    ebp
000104A7  mov     ebp, esp
000104A9  push    [ebp+arg_8]
000104AC  call    1sub_10486
000104B1  test    eax, eax
000104B3  jz      short loc_104BB
000104B5  pop     ebp
000104B6  jmp     NtCreateFile
000104BB ----------------------------
000104BB                  ; CODE XREF: sub_104A4+F j
000104BB  mov     eax, 0C0000034h
000104C0  pop     ebp
000104C1  retn    2Ch
```

The hook function jumps to the original `NtCreateFile` function for some requests and returns to 0xC0000034 for others. The value `0xC0000034` corresponds to `STATUS_OBJECT_NAME_NOT_FOUND`. The call at 1 contains

Interrupts

- Interrupts allow hardware to trigger software events
- Driver calls `IoConnectInterrupt` to register a handler for an interrupt code
- Specifies an Interrupt Service Routine (ISR)
  - Will be called when the interrupt code is generated
- Interrupt Descriptor Table (IDT)
  - Stores the ISR information
  - **!idt** command shows the IDT

Example 11-17. A sample IDT

```
kd> !idt

37:   806cf728 hal!PicSpuriousService37
3d:   806d0b70 hal!HalpApcInterrupt
41:   806d09cc hal!HalpDispatchInterrupt
50:   806cf800 hal!HalpApicRebootService
62:   8298b7e4 atapi!IdePortInterrupt (KINTERRUPT 8298b7a8)
63:   826ef044 NDIS!ndisMIsr (KINTERRUPT 826ef008)
73:   826b9044 portcls!CKsShellRequestor::`vector deleting destructor'+0x26
      (KINTERRUPT 826b9008)
            USBPORT!USBPORT_InterruptService (KINTERRUPT 826df008)
82:   82970dd4 atapi!IdePortInterrupt (KINTERRUPT 82970d98)
83:   829e8044 SCSIPORT!ScsiPortInterrupt (KINTERRUPT 829e8008)
93:   826c315c i8042prt!I8042KeyboardInterruptService (KINTERRUPT 826c3120)
a3:   826c2044 i8042prt!I8042MouseInterruptService (KINTERRUPT 826c2008)
b1:   829e5434 ACPI!ACPIInterruptServiceRoutine (KINTERRUPT 829e53f8)
b2:   826f115c serial!SerialCIsrSw (KINTERRUPT 826f1120)
c1:   806cf984 hal!HalpBroadcastCallService
d1:   806ced34 hal!HalpClockInterrupt
e1:   806cff0c hal!HalpIpiHandler
e3:   806cfc70 hal!HalpLocalApicErrorService
fd:   806d0464 hal!HalpProfileInterrupt
fe:   806d0504 hal!HalpPerfInterrupt
```

Interrupts going to unnamed, unsigned, or suspicious drivers could indicate a rootkit or other malicious software.

Driver Signing

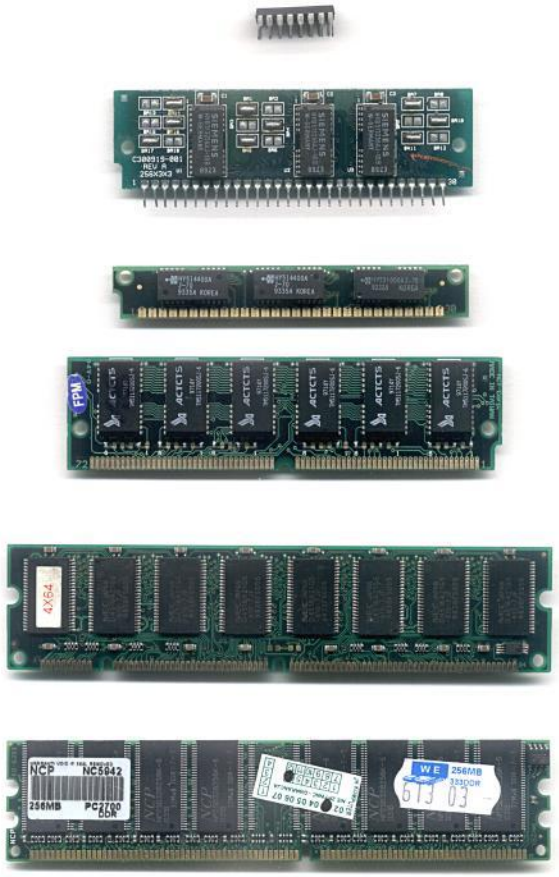
- Enforced in all 64-bit versions of Windows starting with Vista
- Only digitally signed drivers will load
- Effective protection!
- Kernel malware for x64 systems is practically nonexistent
  - You can disable driver signing enforcement by specifying **`nointegritychecks`** in *BCDEdit*

Another approach: memory forensics

What is memory?

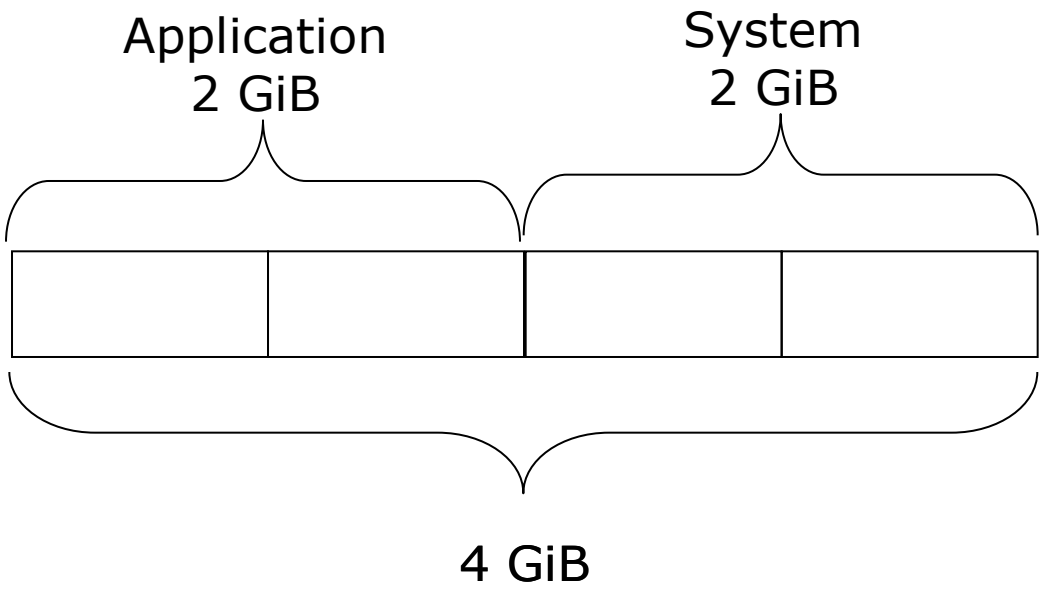
Physical memory is the short-term memory of a computer

- Rapid decay of information as soon as memory module is disconnected from power and clock sources.
  - More on the rapid decay later!



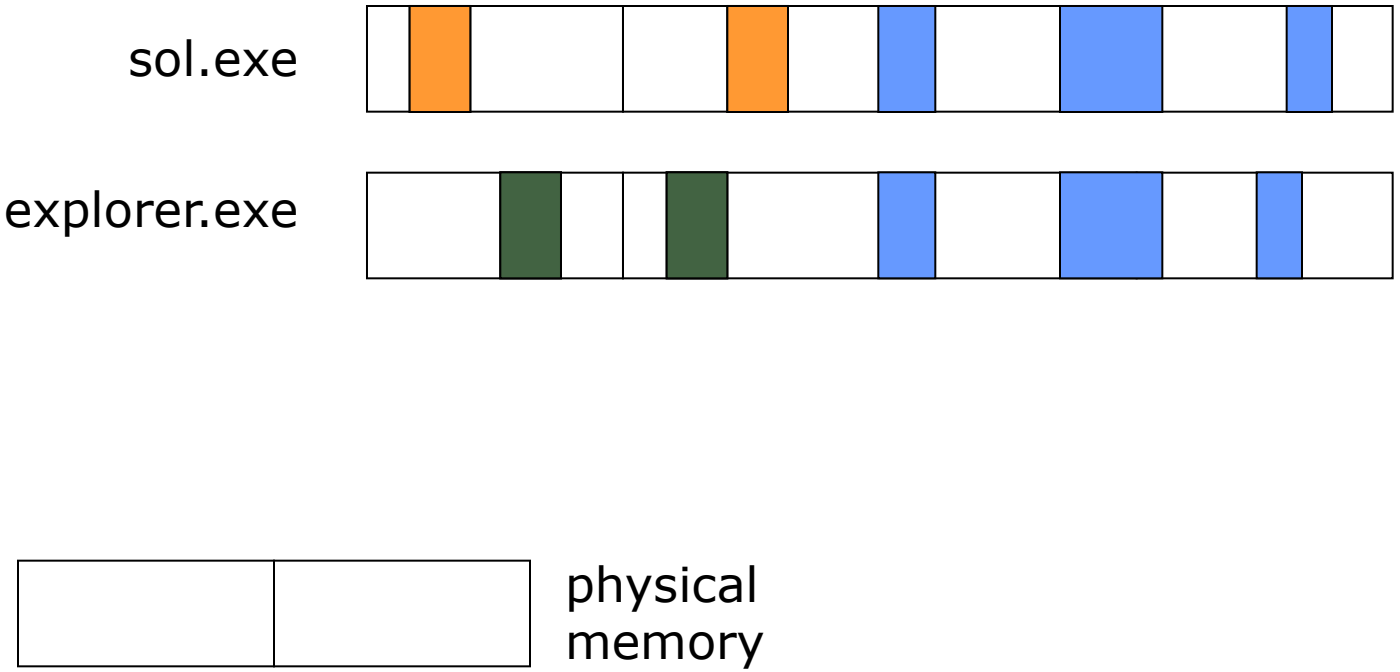
Virtual memory

4 GiB of (virtual) address space per process split into halves



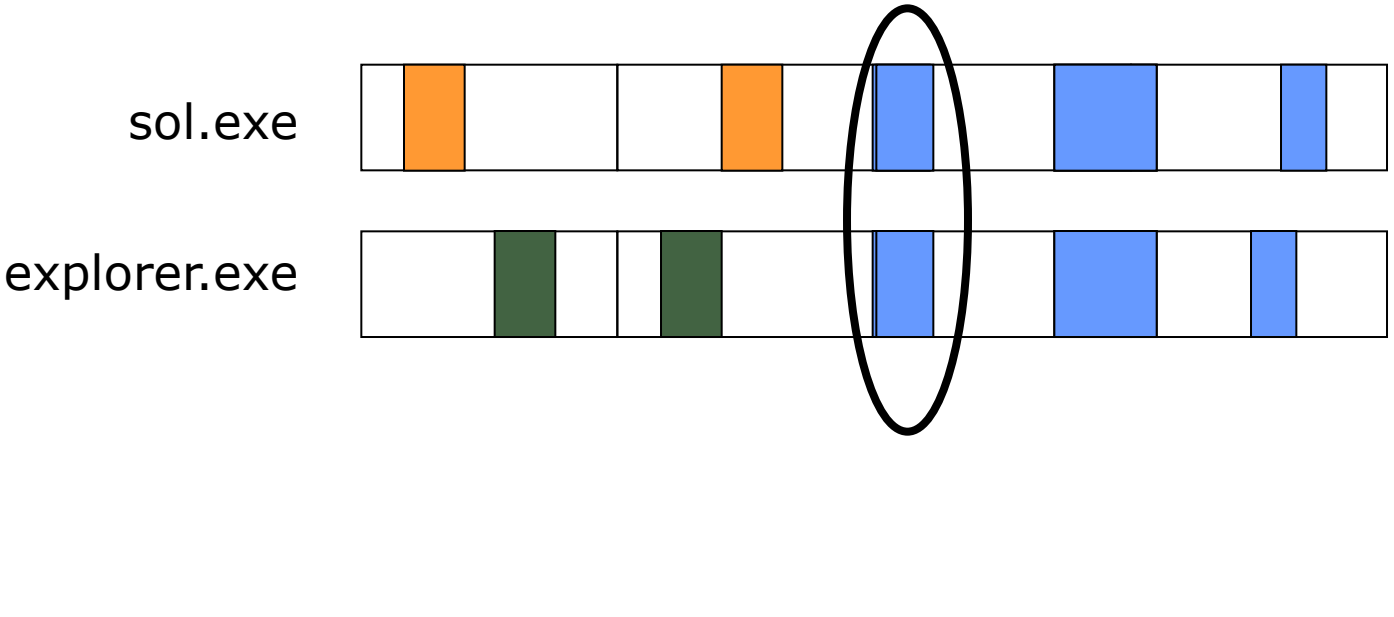
Physical memory

Physical memory is divided into so called “pages” and allocated virtual memory is mapped onto physical memory page by page.



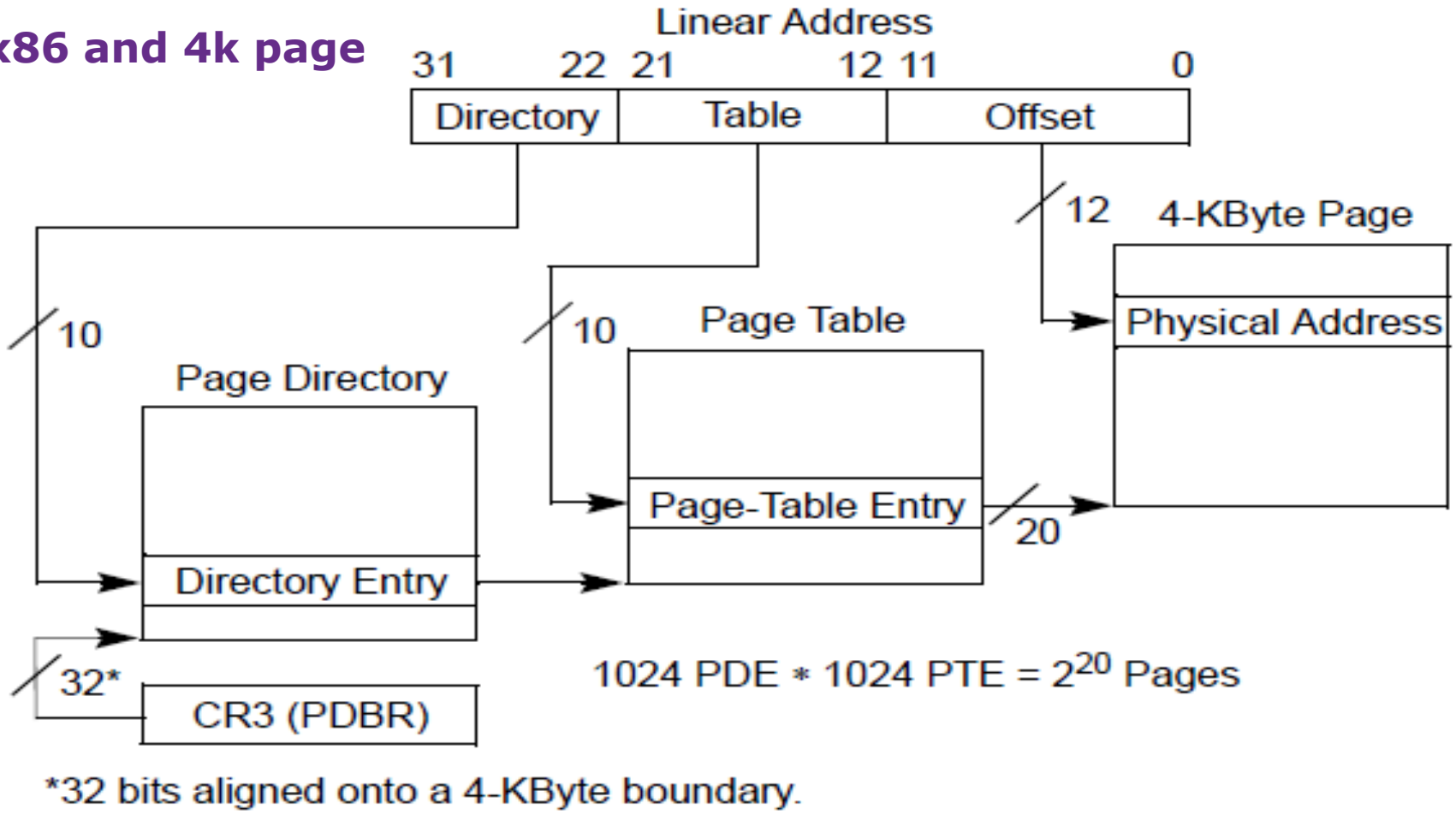
Sharing the same physical page

The same page of physical memory can appear at different locations within the same address space or in different address spaces.



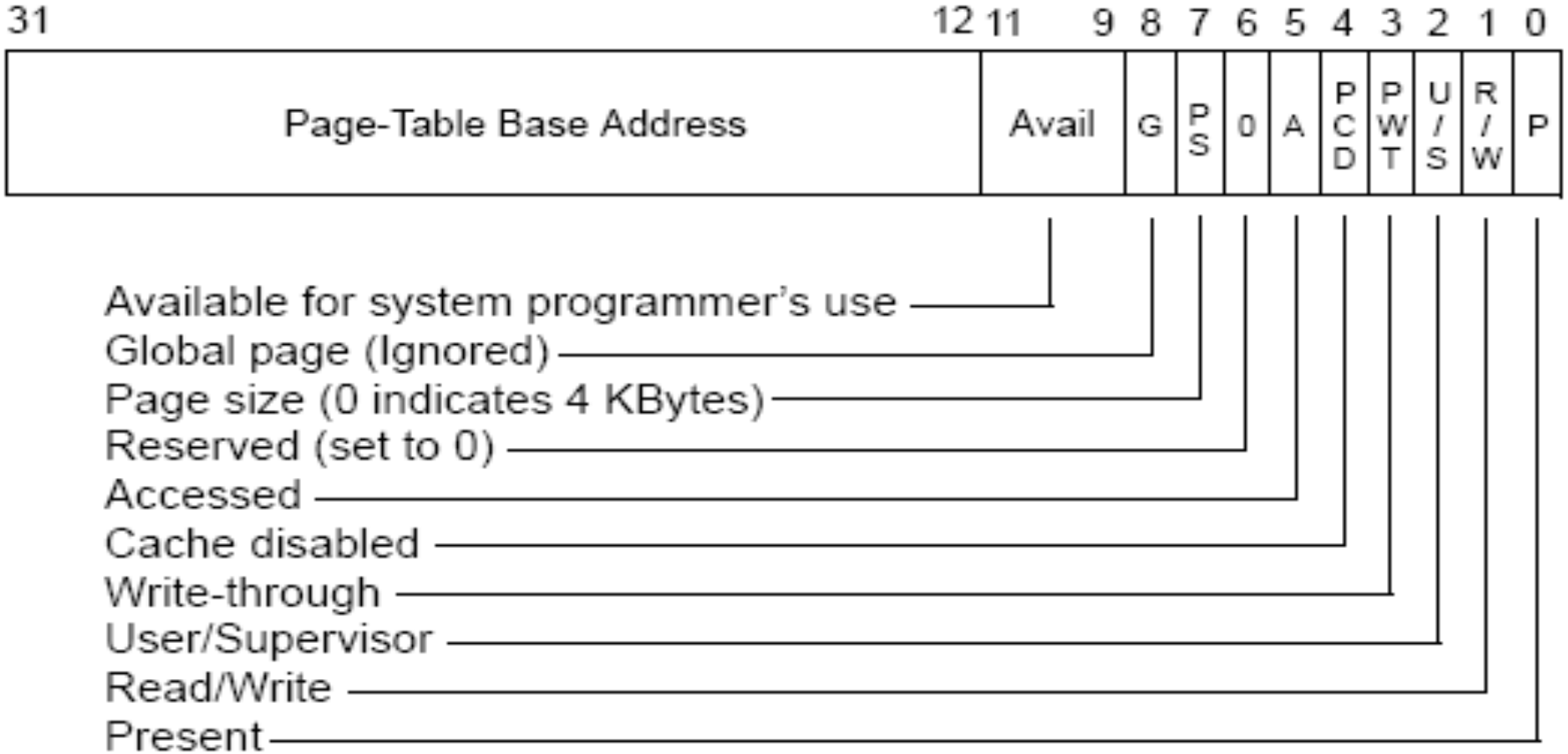
Virtual to Physical memory translation

x86 and 4k page



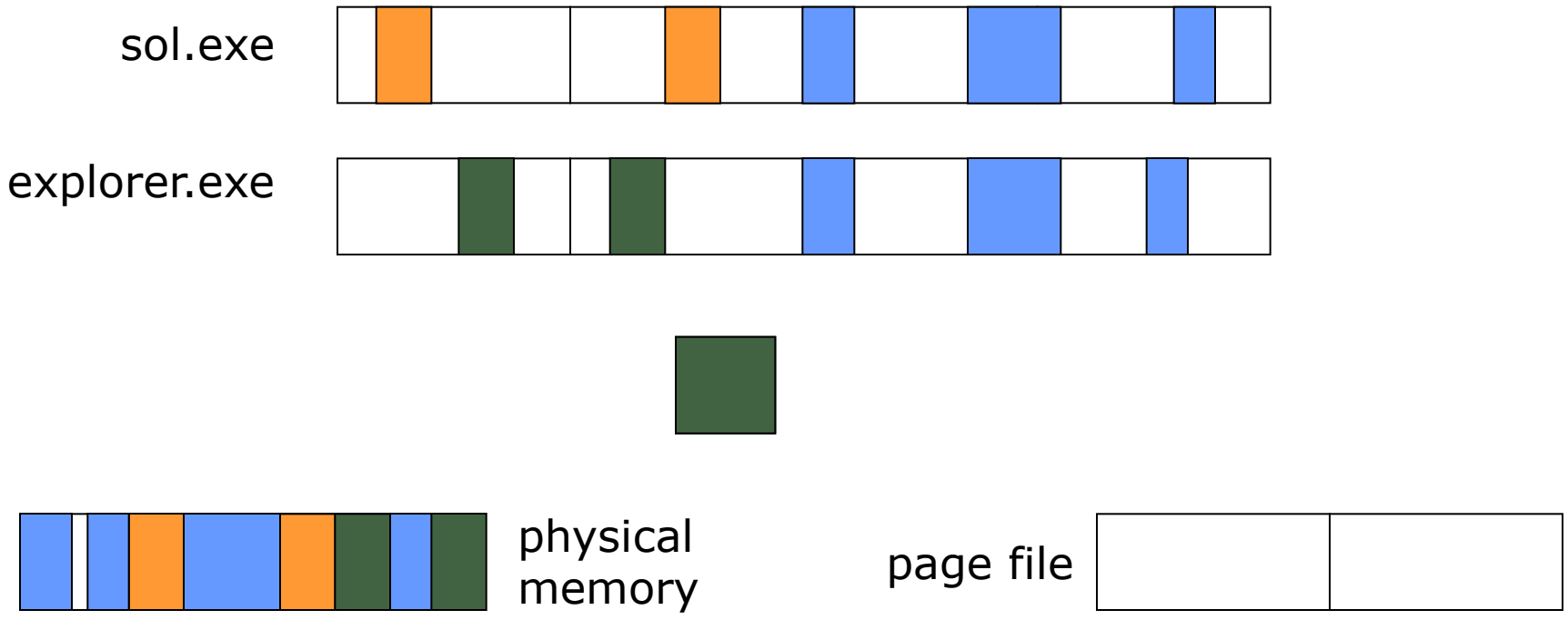
*32 bits aligned onto a 4-KByte boundary.

Important bits in the PTBD



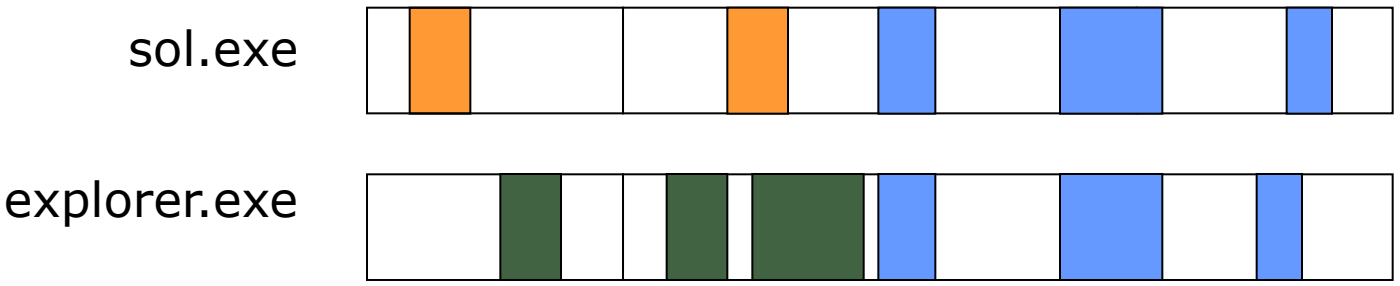
Page file

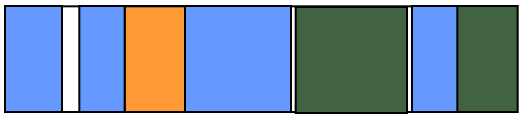
Data can be moved from physical memory into a page file to clear some space



Freed pages

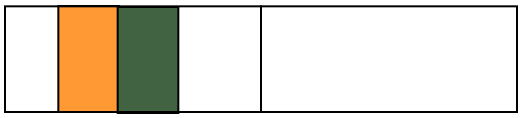
Memory does not get over written when it is marked as free





physical memory

page file



Dumping the memory

Software or Hardware

- Executable code running on the machine or dedicated hardware using DMA to capture the memory

High or low atomicity of the memory dump

Format of the memory dump

- 1:1 copy of the physical memory or a Microsoft crash dump

User rights or Administrator Privileges required

Dumping the memory using dd

Windows makes physical memory accessible through the \\.\PhysicalMemory and \\.\DebugMemory devices.

- Port by George. M. Garner Jr.
  http://users.erols.com/gmgarner/forensics/
- X-Ways Capture (does a lot of other things, too)

Dumping the memory by loading a driver

mdd – ManTech Memory DD
http://sourceforge.net/projects/mdd/

- Works on all Windows versions from Windows 2000 to Windows Server 2008

win32dd - Matthieu Suiche
http://win32dd.msuiche.net/

- Mainly a kernel mode application that does everything with native functions

KnTDD

GMG Systems, Inc. (George M. Garner Jr)
http://www.gmgsystemsinc.com/knttools/

Available to law enforcement and CERT organizations

Also obtains for later analysis

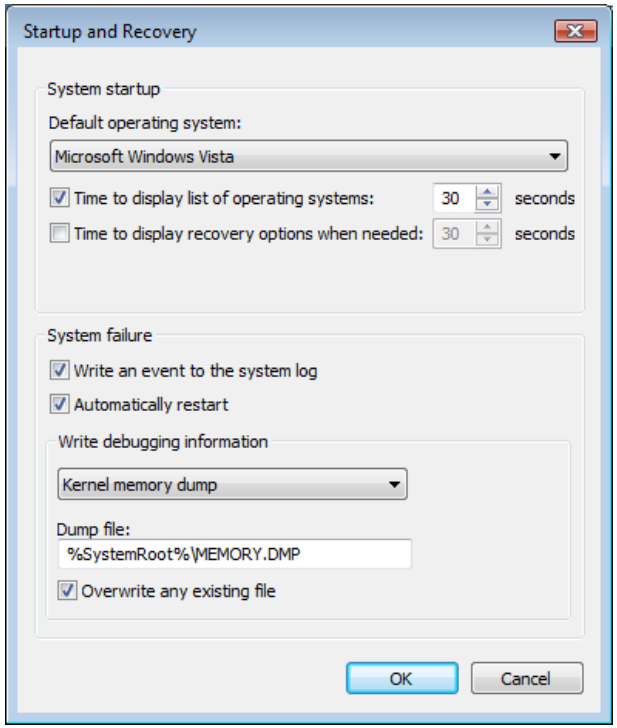
- kernel and network driver binaries
- system status as seen from userland

Enterprise edition allows for digitally signed work packages and encrypted evidence

Microsoft Crash Dump

Configure Windows to write the memory to a file incase of a Blue Screen of Death

- High atomicity of the memory image
- The machine stops temporarily to function



How to generate a Crash Dump

- Kill csrss.exe (Client Server Subsystem) or write your own driver that calls nt!KeBugCheck or nt!KeBugCheckEx.
- NotMyFault from Sysinternals
  http://download.sysinternals.com/Files/Notmyfault.zip
- SystemDump from Citrix (Dimitry Vostokov)
  http://support.citrix.com/article/CTX111072
- Bang from OSR
  http://www.osronline.com/article.cfm?article=153
- Activate crash sequence in PS/2 keyboard driver (USB supported in Windows 2003 SP 1).

LiveKD

LiveKD allows you to run the Kd and Windbg Microsoft kernel debuggers, which are part of the Debugging Tools for Windows package, locally on a live system

- The .dump command generate a crash dump on a live system
- Requires machine specific symbols in order to work

Anti-forensic techniques (1)

Shadow Walker by Sparks and Butler (2005)
http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf

- Controls the contents of memory viewed by another application or driver.
- Modifies page fault handler, marks page as not present, then flushes the Translation Lookaside Buffer (TLB).

Anti-forensic techniques (2)

Ddefy by Darren Bilby (2006)
http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Bilby-up.pdf

- Hooks entry for `nt!NtMapViewofSection` in System Service Descriptor Table (SSDT).
- Monitors access to \\.\PhysicalMemory

Dumping the memory using DMA

Tribble by Brian Carrier and Joe Grand (2004)
http://www.digital-evidence.org/papers/tribble-preprint.pdf

Copilot by Komoku (2004)
http://www.usenix.org/events/sec04/tech/full_papers/petroni/petroni.pdf

- PCI add-in card (requires installation before the incident)
- Not available to the public

Using FireWire to dump the memory

OHCI controller can read and write the first 4 GiB of main memory

- frequently found on laptops
- rarely installed on desktop computers

Adam Boileau
http://www.storm.net.nz/projects/16

Anti-forensic techniques - DMA

Redirecting physical memory access by J. Rutkowska (2007)
http://invisiblethings.org/papers/cheating-hardware-memory-acquisition-updated.ppt

- Manipulates configuration of Northbridge
- At the same physical address CPU and DMA see different Memory

Analyzing the raw memory dump

Different methods to enumerate information

1. Look for a printable string
2. Reconstruct internal data structures
3. Search for static signatures of kernel data structures

Using strings

Sysinternals strings - defaults to Unicode and ASCII, minimum length 3 characters
http://www.microsoft.com/technet/sysinternals/utilities/strings.mspx

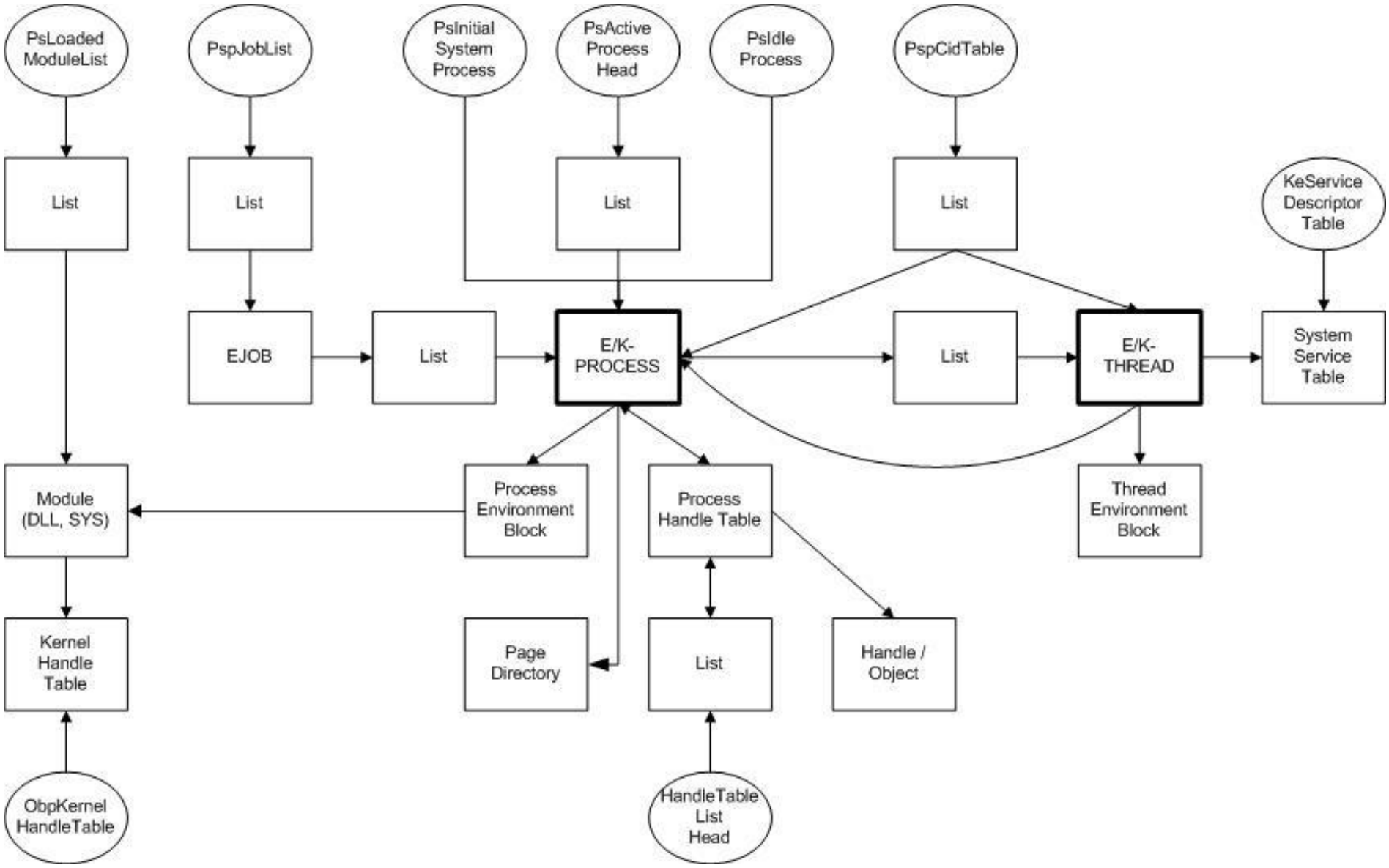
- No context, difficult to interpret
- A lot of interesting information is not in a printable format:
  - Timestamps (FILETIME, uint32)
  - IP addresses

Reconstruct internal data structures

Most data is kept in Lists and Trees.

- From a known starting point reconstruct and follow the list/tree and enumerate the objects found (aka “list-walking”).
- The most important structure is: _LIST_ENTRY, a double-linked list element.

```
kd> dt _LIST_ENTRY
   +0x000 Flink                : Ptr32 _LIST_ENTRY
   +0x004 Blink                : Ptr32 _LIST_ENTRY
```


PsLoaded ModuleList
PspJobList
PsInitial System Process
PsActive Process Head
PsIdle Process
PspCidTable
KeService Descriptor Table
List
List
List
List
EJOB
List
E/K- PROCESS
List
E/K- THREAD
System Service Table
Module (DLL, SYS)
Process Environment Block
Process Handle Table
Thread Environment Block
Kernel Handle Table
Page Directory
List
Handle / Object
ObpKernel HandleTable
HandleTable List Head

Enumerating internal data structures

Pmondump Joe Stewart
http://www.secureworks.com/research/tools/pmodump.pl.gz

lspi - LiSt Process Image by Harlan Carvey
http://windowsir.blogspot.com

Windows Memory Forensic Toolkit (WMFT) by Mariusz Burdach
http://forensic.seccure.net

Search for static signatures of kernel data structures

Simple, brute-force searching

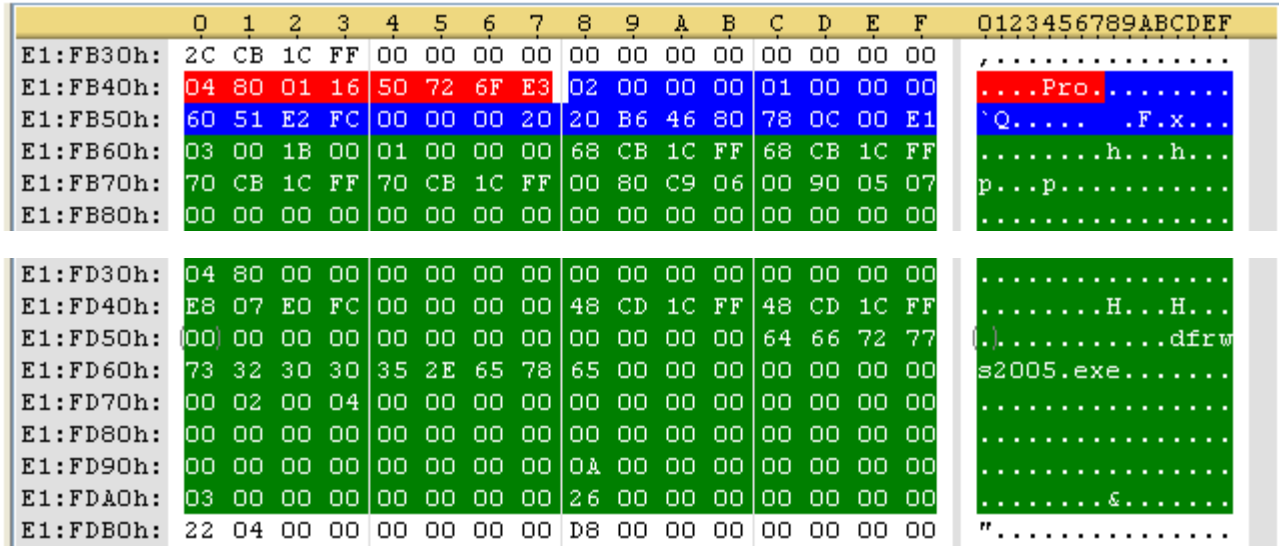
- Largely independent from the dump file format
- Fast, low memory requirements

Problems:

- Assuring a sufficient selectivity
- Signature should be based on essential data, otherwise it can be easily defeated

Memory pool allocations

- Memory management – POOL_HEADER
- Object management – OBJECT_HEADER
- Object – EPROCESS in this example



```
            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
E1:FB30h: 2C CB 1C FF 00 00 00 00 00 00 00 00 00 00 00 00 ,...............
E1:FB40h: 04 80 01 16 50 72 6F E3 02 00 00 00 01 00 00 00 ....Pro.........
E1:FB50h: 60 51 E2 FC 00 00 00 20 20 B6 46 80 78 0C 00 E1 `Q.....  .F.x...
E1:FB60h: 03 00 1B 00 01 00 00 00 68 CB 1C FF 68 CB 1C FF ........h...h...
E1:FB70h: 70 CB 1C FF 70 CB 1C FF 00 80 C9 06 00 90 05 07 p...p...........
E1:FB80h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

E1:FD30h: 04 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
E1:FD40h: E8 07 E0 FC 00 00 00 00 48 CD 1C FF 48 CD 1C FF ........H...H...
E1:FD50h: 00 00 00 00 00 00 00 00 00 00 00 00 64 66 72 77 (.)...........dfrw
E1:FD60h: 73 32 30 30 35 2E 65 78 65 00 00 00 00 00 00 00 s2005.exe.......
E1:FD70h: 00 02 00 04 00 00 00 00 00 00 00 00 00 00 00 00 ................
E1:FD80h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
E1:FD90h: 00 00 00 00 00 00 00 00 0A 00 00 00 00 00 00 00 ................
E1:FDA0h: 03 00 00 00 00 00 00 00 26 00 00 00 00 00 00 00 ........&.......
E1:FDB0h: 22 04 00 00 00 00 00 00 D8 00 00 00 00 00 00 00 "...............
```

Enumerating static kernel data structures

PTFinder and PoolTools by Andreas Schuster
http://computer.forensikblog.de

- Enumerates pool allocations in the memory dump
  - Even exited ones!

Volatity by Aaron Walters and Nick L. Petroni
https://www.volatilesystems.com/default/volatility

- Dumps running processes, threads, loaded modules and much, much more

Windows Debugger

Multipurpose debugger from Microsoft that can be used to debug user mode applications, drivers, and the operating system itself in kernel mode

- Operates on a live system and on crash dumps
- Public symbol server from Microsoft that has most of the public symbols for Windows 2000 and later versions
- Uses extensions to execute custom commands from within the debugger

Converting a raw memory dump to a crash dump

Volatility 1.3
https://www.volatilesystems.com/default/volatility

- Currently supports conversions between different memory formats on Windows XP SP2 and SP3

KntDD
http://www.gmgsystemsinc.com/knttools/

- Saves system state so that a memory dump later can be converted into a crash dump

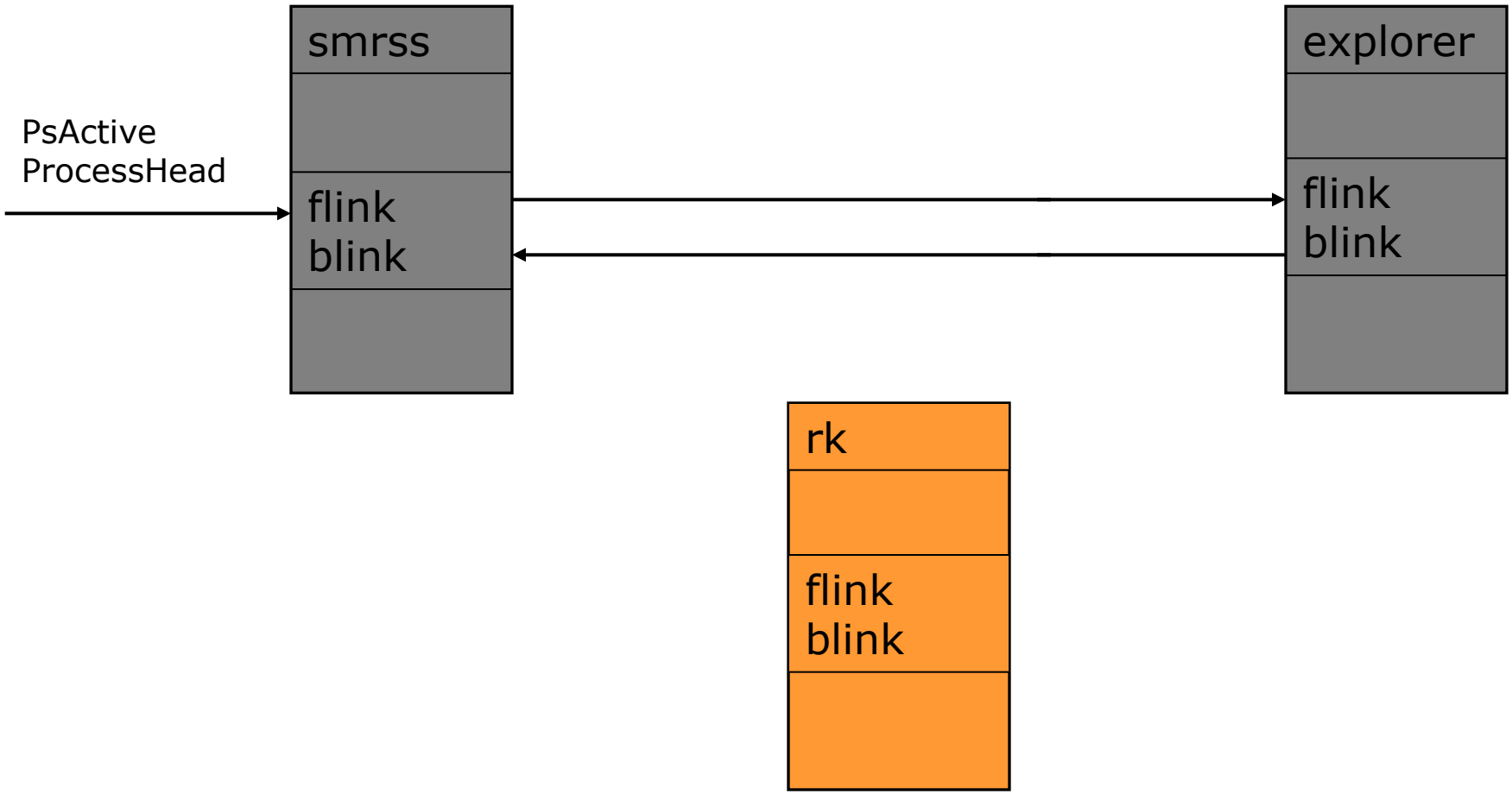
Different rootkit techniques and how we detect it

Three different types of rootkits we will discuss

- DKOM rootkits
- Injecting in a running processes
- Hooking

DKOM rootkits

Works by unlinking doubly linked lists in Windows



Detecting DKOM rootkits

List all loaded objects by enumerating memory pool allocations

- Processes
- Threads
- Drivers

Compare with list enumerated from following doubly linked lists

- Cross view detection

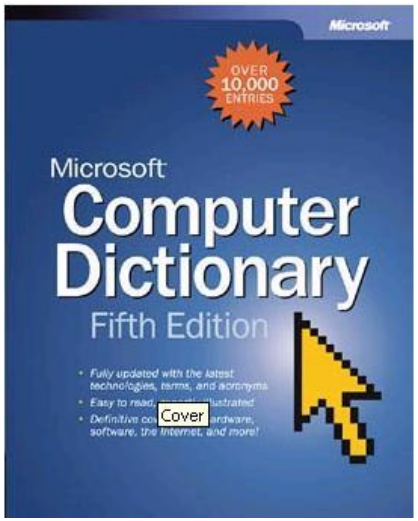
Injecting threads in a running processes

The threads in a processes are the ones that gets execution time. Not the process itself.

- leaching the process

Hooking

"hook n. A location in a routine or program in which the programmer can connect or insert other routines for the purpose of debugging or enhancing functionality"



- Hooking of a single program (API hooking)
- Hooking of system tables or exported functions
- Hooking unexported functions

Hooking exported functions

Some of the popular functions and tables to hook

- GDT (Global Descriptor Table)
- LDT (Local Descriptor Table)
- IDT (interrupt Descriptor Table)
- SSDT (System Service Dispatch Table)
- EAT (Export Address Table)
- IAT (Import Address Table)
- IRP (I/O Request Packet)

Detecting hooking

Highly dependent of the type of function that is being hooked

- `kd> dps win32k!W32pServiceTable`
- `kd> !drvobj Tcpip 0x3`

The `!chkimg` command compares the binary on disk with the one loaded into memory

- Disk image of the loaded drivers must also be collected so the debugger have something to compare with

Hooking unexported functions

Works by changing code deep down in the kernel. Also referred to as “Stealth by design”.

- Deepdoor by Joanna Rutkowska
  http://www.invisiblethings.org
  - Patches deep down in the NDIS structure
  - Deepdoor idea implemented by the uay rootkit

Detecting hooking of unexported function

Detection is generally very difficult.

- Using specific debugger extension