School of

#### Engineering and Computer Science

Te Kura Mātai Pūkaha, Pūrorohiko

### CYBR 473 T2 2021 Malware and Reverse Engineering

Ian Welch, Harith Al-Sahaf

#### **Kernel Debugging with WinDbg**

Chapter 10: "Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software", Michael Sikorski and Andrew Honig, 2012





# WinDbg vs. OllyDbg

- OllyDbg is the most popular <u>user-mode</u> debugger for malware analysts
- WinDbg can be used in <u>either</u> user-mode or kernel-mode
- This lecture explores ways to use WinDbg for kernel debugging and rootkit analysis

# DRIVERS AND KERNEL CODE

•

•

### **Device Drivers**

- Windows device **drivers** allow third-party developers to run code in the Windows kernel
- Drivers are <u>difficult</u> to analyse

 They load into memory, stay resident, and respond to requests from applications

Applications do not <u>directly</u> access kernel drivers

 They access *device objects* which send requests to particular devices

### Devices

- Devices are <u>not physical</u> <u>hardware</u> components

   They are <u>software</u> representations of those components
- A driver creates and destroys devices, which can be accessed from user space



http://www.sharetechnote.com/html/OS\_DeviceDriver.html

## CYBR473-2022T1: Malware and Reverse Engineering

## **Example: USB Flash Drive**

- User plugs in flash drive
- Windows creates the **F**: drive device object
- Applications can now make requests to the F: drive (such as read and write)
  - They will be sent to the <u>driver</u> for that USB flash drive



 $\underline{https://www.digitalcitizen.life/how-unlock-bitlocker-encrypted-flash-drive/}{}$ 

## Loading DLLs vs. Loading Drivers

- Loading DLLs (review)
  - DLLs are loaded into processes
    - DLLs <u>export functions</u> that can be used by applications
    - Using the export table
    - When a function loads or unloads the library, it calls **DLLMain**
- Loading Drivers
  - $\circ$  Drivers must be loaded into the <u>kernel</u>
    - When a driver is first loaded, its **DriverEntry** procedure is called
    - To prepare **callback** objects
    - Just like **DLLMain** for DLLs

## **Defining a Callback Object**

A driver can create a callback object, through which other drivers can request notification of conditions defined by the CallbackName, AttFlags, NULL, NULL);
 Creating driver.

The steps involved in defining a callback object



#### **DLLs vs. Drivers**

#### • DLL

- $\,\circ\,$  Loads into memory when a process is launched
- Executes DLLMain at loadtime
- Prepares the export table

#### • Driver

- $\,\circ\,$  Loads into kernel when hardware is added
- Executes **DriverEntry** at loadtime
- Prepares callback functions and callback objects

### DriverEntry

- DLLs expose functionality through the export table; drivers don't
- Drivers must register the address for <u>callback</u> functions
  - They will be called when a user-space software component requests a service
  - **DriverEntry** routine performs this <u>registration</u>
  - Windows creates a *driver object structure*, passes it to **DriverEntry** which fills it with **callback** functions
  - DriverEntry then creates a device that can be accessed from userland

#### **Example: Normal Read**

• Normal read request

• User-mode application obtains a <u>file handle</u> to device

- Calls **ReadFile** on that handle
- Kernel processes ReadFile request

Invokes the driver's callback function handling I/O

#### **Malicious Request**

- Most common request from malware is DeviceIoControl
  - A generic request from a user-space module to a device managed by a driver
  - User-space program passes in an <u>arbitrary-length buffer</u> of input data

• Received an <u>arbitrary-length buffer</u> of data as output

#### How User-mode Calls are Handled by the Kernel



#### Ntoskrnl.exe & Hal.dll

- Malicious drivers rarely control hardware
- They interact with Ntoskrnl.exe & Hal.dll
   Ntoskrnl.exe has code for core OS functions
  - Hal.dll has code for interacting with <u>main hardware</u> components
- Malware will import functions from <u>one or both</u> of these files so it can **manipulate the kernel**

#### SETTING UP KERNEL DEBUGGING

•

#### VMware

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

#### Boot.ini

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

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.
C:\Windows\system32>bcdedit /debug on
The operation completed successfully.
```

## Installing WinDbg

 You can get Debugging Tools for Windows as part of a development kit or as a standalone tool set:

#### $\circ$ $\,$ As part of the WDK $\,$

Debugging Tools for Windows is included in the Windows Driver Kit (WDK). To get the WDK, see <u>Download the Windows Driver Kit (WDK)</u>.

#### • As part of the Windows SDK

Debugging Tools for Windows is included in the Windows Software Development Kit (SDK). To download the installer or an ISO image, see <u>Windows 10 SDK</u> on Windows Dev Center.

#### • As a standalone tool set

You can install the Debugging Tools for Windows alone, without the Windows SDK or WDK, by starting installation of the Windows SDK and then selecting only Debugging Tools for Windows in the list of features to install (and clearing the selection of all other features). To download the installer or an ISO image, see <u>Windows 10 SDK</u> on Windows Dev Center.

https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/

## Installing WinDbg (cont.)

#### 🕼 Windows Software Development Kit - Windows 10.0.17763.1

#### 

#### Select the features you want to download

#### Click a feature name for more information.

Windows Performance Toolkit	Windows Performance Toolkit	
Debugging Tools for Windows	Tools to record system events by using Ev	ent Tracing for
Application Verifier For Windows	Windows, and a tool to analyze performa graphical user interface	ance data in a
.NET Framework 4.7.2 Software Development Kit	graphical aser interfacer	
Windows App Certification Kit	Includes:	
Windows IP Over USB	Windows Performance Recorder	
MSI Tools	<ul> <li>Windows Performance Analyzer</li> <li>Xperf</li> </ul>	
Windows SDK Signing Tools for Desktop Apps	·	
Windows SDK for UWP Managed Apps		
Windows SDK for UWP C++ Apps		
Windows SDK for UWP Apps Localization		
Windows SDK for Desktop C++ x86 Apps		
Windows SDK for Desktop C++ amd64 Apps		
Windows SDK for Desktop C++ arm Apps	Estimated disk space required:	157.2 MB
Windows SDK for Desktop C++ arm64 Apps	Disk space available:	46.3 GB
	Back Downloa	d Cancel

#### **Run LiveKD**

C:\Windows\system32>livekd -w

LiveKd v5.40 - Execute kd/windbg on a live system Sysinternals - www.sysinternals.com Copyright (C) 2000-2015 Mark Russinovich and Ken Johnson

Symbols are not configured. Would you like LiveKd to set the \_NT\_SYMBOL\_PATH directory to reference the Microsoft symbol server so that symbols can be obtained automatically? (y/n) \_

#### Run LiveKD (cont.)

Q

	_	_
Command - Dump C:\Windows\livekd.dmp - WinDbg:10.0.10586.567 X86	) ×	
Product: WinNt, suite: TerminalServer SingleUserTS		^
Built by: 10586.162.x86fre.th2_release_sec.160223-1728 Machine Name:		
Kernel base = 0x82002000 PsLoadedModuleList = 0x82208138		
Debug session time: Mon Apr 4 10:14:28.467 2016 (UTC - 7:00) System Untime: 0 days 0:00:43 012		
WARNING: Process directory table base 3FFF3420 doesn't match CR3 3FFF3720		
WARNING: Process directory table base 3FFF3420 doesn't match CR3 3FFF3720		
hoading kerner Symbols		
Loading User Symbols		
Loading unloaded module list		
*** ERROR: Module load completed but symbols could not be loaded for LiveKdD.SYS		
		~
	>	_
kd>		

# USING WinDbg

**Command-line Commands** 

0 0.

## **Reading from Memory and Editing Memory**

- Reading: **d***x* addressToRead
- Editing: **e***x* addressToWrite dataToWrite
- x can be
  - a Displays/Writes as **ASCII text**
  - o u Displays/Writes as Unicode text
  - d Displays/Writes as **32-bit double words**
- da 0x401020

 $\,\circ\,$  Shows the ASCII text starting at 0x401020

### **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
  - $\,\circ\,$  The second argument is the function name

### No Breakpoints with LiveKD

- LiveKD works from a memory dump
- It's read-only
- So you can't use breakpoints

## **Listing Modules**

• 1m

○ Lists all modules loaded into a process

- Including EXEs and DLLs in user space
- And the kernel drivers in kernel mode
- $\,\circ\,$  As close as WinDbg gets to a memory map
- 1m m disk
  - $\,\circ\,$  Shows the disk driver

💯 Dump C:\Windows\livekd.dmp - WinDbg:6.11.0001.404 X86	
File Edit View Debug Window Help	
Command	×
kd> lm m disk start end module name 8616a000 8617b000 disk (pdb symbols) c:\symbols\disk.pdb\67E3BE599CEF4C09BA5507664AD5CDA51\disk.pdb	
kd>	
Ln 0, Col 0 Sys 0:C:\Wind Proc 000:0 Thrd 000:0 ASM OVR C	CAPS NUM

# **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	000000Ъ8	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
ינ	8243e010	000000Ъ8	00000000	00000040	00000000
-	8243e020	00000000	00000000	00000000	00000000
	8243e030	00000000	00000000	00000000	00000268

## **Online Help**

#### • .hh dd

- Shows help about "dd" command
- $\,\circ\,$  But there are no examples

🛃 Debugging Tools for Windows	
File Edit View Go Help	
日本語 合 し、 一本語 A Locate Previous Next Back Formula	⊨> 😒 🚺 🚮 💒 🎒 थिँ+ nrward Stop Refresh Home Font Print Options
Contents Index Search Favorites	+ Debugging Tools for Windows
Type in the keyword to find:	d, da, db, dc, dd, dD, df, dp, dq, du, dw, dW, dyb, dyd (Display Memory)
dd dd (Display Memory) command dD (Display Memory) command dd (Display Memory) command	The <b>d</b> * commands display the contents of memory in the given range.
daa (uspiay Keterenced Wemory) command DDK (Driver Development Kit) ddp (Display Referenced Memory) command dds (Display Words and Symbols) command ddu (Display Referenced Memory) command	d{a b c d D f p q u w W} [Options] [Range] dy{b d} [Options] [Range] d [Options] [Range]
Deadlock Detection (Univer Vernier) deadlocks Debug   Break Debug   Detach Debuggee Debug   Go Debug   Go Handled Exception Debug   Go Unhandled Exception Debug   Go Unhandled Exception Debug   Kemel Connection   Cycle Baud Rate Debug   Kemel Connection   Cycle Initial Brea Debug   Kemel Connection   Resynchronize Debug   Modules	Parameters Options Specifies one or more display options. Any of the following options can be included, except that no more than one /p* option can be indicated: /c Width Specifies the number of columns to use in the display. If this is omitted, the default number of columns depends on the display type. /p (Kernel-mode only) Uses physical memory addresses for the display. The range ✓
Pare kd> da nt+4c 8243e04c ".!This program can 8243e06c "DOS mode\$" windbg>.hh dd 4 kd>	nnot be run in "

#### **More Commands**

r
○ Dump all registers

$\leftrightarrow \rightarrow \mathbb{C} \ \textcircled{0} \ \texttt{windbg.info}/doc/1-common-cmds.html$						
Thinking debugging? Thin windbg.inf	k O					
You are here:  Home Docu	ments Common WinDbg Commands	(Thematically Grouped)				
Main Menu	Common WinDbg Comman	ids (Thematically Grou	ped)			
Home <b>Documents</b> Applications and Tools Forum			🔮 Discuss 🖃 E-			
Miscellanea	1) Built-in help commands	9) Exceptions, events, and crash analysis	17) Information about variables			
Who Visits Us? Contact & Imprint	<ol> <li>General WinDbg's commands (clear screen,)</li> </ol>	10) Loaded modules and image information	18) Memory			
Search Our Site	3) Debugging sessions (attach, detach,)	11) Process related information	19) Manipulating memory ranges			
- All - 🗘	4) Expressions and commands	12) Thread related information	20) Memory: Heap			
Search Keyword Go	5) Debugger markup language (DML)	13) Breakpoints	21) Application Verifier			
	6) Main extensions	14) Tracing and stepping (F10, F11)	22) Logging extension (logexts.dll)			
C RSS Feeds	7) Symbols	15) Call stack				
<ul><li>Front Page</li><li>Documents</li><li>Applications and Tools</li></ul>	8) Sources	16) Registers				

# MICROSOFT SYMBOLS

•

### Symbols are Labels

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

## **Searching for Symbols**

• moduleName!symbolName

 $\,\circ\,$  Can be used anywhere an address is expected

- moduleName
  - $\circ$  The EXE, DLL, or SYS filename (without extension)
- symbolName

Name associated with the address

- ntoskrnl.exe is an exception, and is named nt
  - o Ex: u nt!NtCreateProcess
    - Unassembles that function (disassembly)

#### Demo

- Try these
  - o u nt!ntCreateProcess
  - $\circ$  u nt!ntCreateProcess L10
  - $\circ$  u nt!ntCreateProcess L20

kd≻ u nt∣	ntCreateProcess	-	
nt!NtCrea	ateProcess:		
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

 $\,\circ\,$  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!MmCreateProcessNotifyRoutine = <no type information>

8055a2e0 nt!PspCreateProcessNotifyRoutine = <no type information>
```

# Listing Closest Symbol with ln

- Helps in figuring out where a call goes
- ln *address*



## Viewing Structure Information with dt

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

#### **Example: Viewing Type Information for a Structure**

0:0	000> <b>dt</b>	nt!_DRIVER_OBJECT	Г	
kd	> dt nt	_DRIVER_OBJECT		
	+0x000	Туре	:	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-taq></unnamed-taq>
+0x05c AlignmentReguirement : Uint4B
+0x060 DeviceQueue : KDEVICE QUEUE
$+0 \times 074$ Dpc : KDPC
+0x094 ActiveThreadCount : Uint4B
+0x098 SecurityDescriptor : Ptr32 Void
+0x09c DeviceLock : KEVENT
+0x0ac SectorSize : Uint2B
+0x0ae Spare1 Uint2B
+0x0b0 DeviceObjectExtension Ptr32 DEVOBL EXTENSION
+0x0b4 Reserved · Ptr32 Void

## Show Specific Values for the "Beep" Driver

kd> dt nt!_DRIVER_OBJECT 82	8b	2648
+0x000 Type	:	4
+0x002 Size	:	168
+0x004 DeviceObject	:	0x828b0a30 _DEVICE_OBJECT
+0x008 Flags	:	0x12
+0x00c DriverStart	:	0xf7adb000
+0x010 DriverSize	:	0×1080
+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	:	10xf7adb66c long Beep!DriverEntry+0
+0x030 DriverStartIo	:	0xf7adb51a void Beep!BeepStartIo+0
+0x034 DriverUnload	:	0xf7adb620 void Beep!BeepUnload+0
+0x038 MajorFunction	:	<pre>[28] 0xf7adb46a long Beep!BeepOpen+0</pre>

### **Initialization Function**

- The DriverInit function is called first when a driver is loaded
  - $\,\circ\,$  See labelled line in previous slide
- Malware will sometimes place its <u>entire</u> 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

o SRC\*c:\websymbols\*http://msdl.microsoft.com
/download/symbols

#### **Manually Downloading Symbols**

← → C 🖍 🗋 msdn.microsoft.com/en-us/windows/hardware/gg463028.aspx

4ª @ ☆ 🥹 🚟 🌩 👻 🐌 🚍 🖸 🗿 Ξ

#### Download Windows Symbol Packages

The easiest way to get Windows symbols is to use the Microsoft Symbol Server. The symbol server makes symbols available to your debugging tools as needed. After a symbol file is downloaded from the symbol server it is cached on the local computer for quick access.

If you prefer to download the entire set of symbols for Windows 8.1 Preview, Windows Server 2012 R2 Preview, Windows 8, Windows Server 2012, Windows 7, Windows Server 2008 R2, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, or Windows 2000, then you can download a symbol package and install it on your computer.

# KERNEL DEBUGGING IN PRACTICE

0

0.

### **Kernel Mode and User Mode Functions**

- We'll examine a program that writes to files <u>from kernel</u> space
  - $\,\circ\,$  An unusual thing to do
  - Fools some security products
  - Kernel mode programs <u>cannot</u> call user-mode functions like

CreateFile and WriteFile

O Must use NtCreateFile and NtWriteFile

#### **User-Space Code**

Creates a service with the CreateService function
 dwServiceType is 0x01 (Kernel driver)

#### Creating a service to load a kernel driver

04001B3D push esi ; lpPassword 04001B3E push esi ; lpServiceStartName 04001B3F push esi ; lpDependencies 04001B40 push esi ; lpdwTagId 04001B41 push esi ; lpLoadOrderGroup 04001B42 push [ebp+lpBinaryPathName] ; lpBinaryPathName 04001B45 push 1 ; dwErrorControl 04001B47 push 3 ; dwStartType 04001B49 push <b>1</b> 1 ; dwServiceType 04001B48 push 0F01FFh ; dwDesiredAccess 04001B50 push [ebp+lpDisplayName] ; lpDisplayName 04001B53 push [ebp+lpDisplayName] ; lpServiceName 04001B56 push [ebp+hSCManager] ; hSCManager 04001B59 call ds:imp_CreateServiceA@52			
04001B3Epushesi; lpServiceStartName04001B3Fpushesi; lpDependencies04001B40pushesi; lpdwTagId04001B41pushesi; lpLoadOrderGroup04001B42push[ebp+lpBinaryPathName] ; lpBinaryPathName04001B45push1; dwErrorControl04001B47push3; dwStartType04001B49push1; dwServiceType04001B48push0F01FFh; dwDesiredAccess04001B50push[ebp+lpDisplayName] ; lpDisplayName04001B53push[ebp+lpDisplayName] ; lpServiceName04001B56push[ebp+hSCManager] ; hSCManager04001B59callds:impCreateServiceA@52	04001B3D	push	esi ; lpPassword
04001B3Fpushesi; lpDependencies04001B40pushesi; lpdwTagId04001B41pushesi; lpLoadOrderGroup04001B42push[ebp+lpBinaryPathName] ; lpBinaryPathName04001B45push1; dwErrorControl04001B47push3; dwStartType04001B49push1; dwServiceType04001B48push0F01FFh; dwDesiredAccess04001B50push[ebp+lpDisplayName] ; lpDisplayName04001B53push[ebp+lpDisplayName] ; lpServiceName04001B56push[ebp+hSCManager] ; hSCManager04001B59callds:imp_CreateServiceA@52	04001B3E	push	esi ; lpServiceStartName
04001B40pushesi; lpdwTagId04001B41pushesi; lpLoadOrderGroup04001B42push[ebp+lpBinaryPathName] ; lpBinaryPathName04001B45push1; dwErrorControl04001B47push3; dwStartType04001B49push1; dwServiceType04001B49push1; dwDesiredAccess04001B50push[ebp+lpDisplayName] ; lpDisplayName04001B53push[ebp+lpDisplayName] ; lpServiceName04001B56push[ebp+hSCManager] ; hSCManager04001B59callds:imp_CreateServiceA@52	04001B3F	push	esi ; lpDependencies
04001B41pushesi; lpLoadOrderGroup04001B42push[ebp+lpBinaryPathName]; lpBinaryPathName04001B45push1; dwErrorControl04001B47push3; dwStartType04001B49push11; dwServiceType04001B48push0F01FFh; dwDesiredAccess04001B50push[ebp+lpDisplayName]; lpDisplayName04001B53push[ebp+lpDisplayName]; lpServiceName04001B56push[ebp+hSCManager]; hSCManager04001B59callds:imp_CreateServiceA@52	04001B40	push	esi ; lpdwTagId
04001B42 push [ebp+lpBinaryPathName]; lpBinaryPathName 04001B45 push 1 ; dwErrorControl 04001B47 push 3 ; dwStartType 04001B49 push 11 ; dwServiceType 04001B4B push 0F01FFh ; dwDesiredAccess 04001B50 push [ebp+lpDisplayName]; lpDisplayName 04001B53 push [ebp+lpDisplayName]; lpServiceName 04001B56 push [ebp+hSCManager]; hSCManager 04001B59 call ds:impCreateServiceA@52	04001B41	push	esi ; lpLoadOrderGroup
04001B45 push 1 ; dwErrorControl 04001B47 push 3 ; dwStartType 04001B49 push 11 ; dwServiceType 04001B48 push 0F01FFh ; dwDesiredAccess 04001B50 push [ebp+lpDisplayName]; lpDisplayName 04001B53 push [ebp+lpDisplayName]; lpServiceName 04001B56 push [ebp+hSCManager]; hSCManager 04001B59 call ds:imp_CreateServiceA@52	04001B42	push	<pre>[ebp+lpBinaryPathName] ; lpBinaryPathName</pre>
04001B47push3; dwStartType04001B49push11; dwServiceType04001B4Bpush0F01FFh; dwDesiredAccess04001B50push[ebp+lpDisplayName]; lpDisplayName04001B53push[ebp+lpDisplayName]; lpServiceName04001B56push[ebp+hSCManager]; hSCManager04001B59callds:imp_CreateServiceA@52	04001B45	push	1 ; dwErrorControl
04001B49push1; dwServiceType04001B4Bpush0F01FFh; dwDesiredAccess04001B50push[ebp+lpDisplayName] ; lpDisplayName04001B53push[ebp+lpDisplayName] ; lpServiceName04001B56push[ebp+hSCManager] ; hSCManager04001B59callds:impCreateServiceA@52	04001B47	push	3 ; dwStartType
04001B4B push 0F01FFh ; dwDesiredAccess 04001B50 push [ebp+lpDisplayName]; lpDisplayName 04001B53 push [ebp+lpDisplayName]; lpServiceName 04001B56 push [ebp+hSCManager]; hSCManager 04001B59 call ds:impCreateServiceA@52	04001B49	push	1 ; dwServiceType
04001B50 push [ebp+lpDisplayName]; lpDisplayName 04001B53 push [ebp+lpDisplayName]; lpServiceName 04001B56 push [ebp+hSCManager]; hSCManager 04001B59 call ds:impCreateServiceA@52	04001B4B	push	0F01FFh ; dwDesiredAccess
04001B53 push [ebp+lpDisplayName]; lpServiceName 04001B56 push [ebp+hSCManager]; hSCManager 04001B59 call ds:impCreateServiceA@52	04001B50	push	[ebp+lpDisplayName] ; lpDisplayName
04001B56 push [ebp+hSCManager] ; hSCManager 04001B59 call ds:impCreateServiceA@52	04001B53	push	<pre>[ebp+lpDisplayName] ; lpServiceName</pre>
04001B59 call ds:impCreateServiceA@52	04001B56	push	[ebp+hSCManager] ; hSCManager
	04001B59	call	ds:impCreateServiceA@52

## **User-Space Code (cont.)**

Not shown: edi being set to
 \\.\FileWriter\Device

#### Obtaining a handle to a device object

04001893	хог	eax, eax	
04001895	push	eax	; hTemplateFile
04001896	push	80h	; dwFlagsAndAttributes
0400189B	push	2	; dwCreationDispositio
0400189D	push	eax	; lpSecurityAttributes
0400189E	push	eax	; dwShareMode
0400189F	push	ebx	; dwDesiredAccess
040018A0	2push	edi	; lpFileName
040018A1	1call	esi ; Create	FileA

## **User-Space Code (cont.)**

 Once the malware has a handle to the device, it uses the DeviceIoControl function to send data to the

driver.

Using **DeviceIoControl** to communicate from user space to kernel space

04001910	push	Θ	;	lp0verlapped
04001912	sub	eax, ecx		
04001914	lea	ecx, [ebp+Bytes	?e	turned]
0400191A	push	ecx	;	lpBytesReturned
0400191B	push	64h	;	nOutBufferSize
0400191D	push	edi	;	lpOutBuffer
0400191E	inc	eax		
0400191F	push	eax	;	nInBufferSize
04001920	push	esi	;	lpInBuffer
04001921	push	9C402408h	;	dwIoControlCode
04001926	push	[ebp+hObject]	;	hDevice
0400192C	call	ds:DeviceIoContr	-o	11 (

#### Kernel-Mode Code

- Set WinDbg to <u>Verbose</u> 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 <u>suspicious</u>
  - Except Kmixer.sys in VMware machines

### Kernel-Mode Code (cont.)

 Example: we see 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 (cont.)

• !drvobj command shows driver object

```
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 (cont.)

 dt command shows structure

kd>dt_nt!_DRIVER_OBJECT_0x	82	7e3698
nti DRIVER OBJECT	52	105050
		4
+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 <u>drivers</u>
- 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

#### ROOTKITS

•

•

#### **Rootkit Basics**

- Rootkits modify the internal functionality of the OS to <u>conceal</u> themselves
  - Hide processes, network connections, and other resources from running programs
  - Difficult for antivirus, administrators, and security analysts to discover their malicious activity
- Most rootkits modify the kernel
- Most popular method:

• System Service Descriptor Table (SSDT) hooking

## 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 <u>user space</u> to <u>access</u> <u>kernel code</u>
   <u>SYSCALL</u>

#### **O SYSENTER**

- Used by modern versions of Windows
- Function code stored in EAX register
- **INT 0x2E**

### Example from ntdll.dll

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

7C90D682	1mov	eax, 25h	;	NtCreateFile	
7C90D687	MOV	edx, 7FF	E0300h		
7C90D68C	call	dword pt	r [edx]		
7C90D68E	retn	2Ch			
The call	to dword	ptr[ed>	x] will go	to the following	instructions:
7c90eb	8b 8bd4	mov e	edx,esp		
7c90eb	8d 0f34	sysenter			

### **SSDT Table Entries**

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

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

Several entries of the SSDT table showing NtCreateFile

#### 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
  - $\,\circ\,$  Just look at the SSDT
  - $\,\circ\,$  Look for values that are unreasonable
  - In this case, ntoskrnl.exe starts at address 804d7000 and ends at 806cd580
  - o ntoskrnl.exe is the Kernel!
- 1m m nt

Lists modules matching "nt" (Kernel modules)
 Shows the SSDT table

#### **SSDT** Table

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

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	1 8056bc50	: 805ca3ca
8050129c	805ca102	80618e86	8056d4d8	8060c240
805012ac	8056d404	8059fba6	80599202	805c5f8e

A sample SSDT table with one entry overwritten by a rootkit

# **Finding the Malicious Driver**

#### • 1m

Lists open modules
In the kernel, they are all drivers

Using the **lm** command to find which drive contains a particular address

#### kd>lm

#### . . .

f7ac8580	intelide	(deferred)
f7aca700	dmload	(deferred)
f7ada680	Rootkit	(deferred)
f7aee280	vmmouse	(deferred)
	f7ac8580 f7aca700 f7ada680 f7aee280	f7ac8580 intelide f7aca700 dmload f7ada680 Rootkit f7aee280 vmmouse

. . .

### Listing of the Rootkit Hook Function

000104A4	mov	edi, edi
000104A6	push	ebp
000104A7	mov	ebp, esp
000104A9	push	[ebp+arg_8]
000104AC	call	<mark>1</mark> sub_10486
000104B1	test	eax, eax
000104B3	jz	short loc_104BB
000104B5	рор	ebp
000104B6	jmp	NtCreateFile
000104BB		
000104BB		; CODE XREF: sub_104A4+F j
000104BB	mov	eax, 0C0000034h
000104C0	рор	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 <u>hardware</u> to trigger <u>software</u> 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
  - o !idt command shows the IDT

### 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: 806d0604 hal!HalpPerfInterrupt

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

## **Loading Drivers**

 If you want to load a driver to test it, you can download the OSR Driver Loader tool

OSR Driver Loader		? x		
Open Syster 105 Route 1 Amherst, NH Ph: (603) 50 Fax: (603) 5 Ver: V3.0 -	ns Resources, Inc. 01A Suite 19 03031 55-5500 35-6503 Sept 6, 2007	Exit Help ServiceGroupOrder Active Services		
Registry Key:	yourdrivername			
Driver Path:	C:\Windows\system32\drivers\sample.sys	Browse		
Driver Version:				
Driver Size:				
Driver File Time:				
Display Name:	yourdrivername			
Service Start:	Demand			
Load Group:	None 🔻	aroup Load Order		
Order In Group:	1 🕂 Type: Driver 🔻 Em	or: Normal 🛛 🔻		
Depend On Group(s):	AudioGroup Base Boot Bus Extender Boot File System	•		
Last Status:				
MiniFilter Settings				
Default Instance:	Altitude: 0			
AltitudeAndFlags	Flags: 0			
<u>R</u> egister Service	Unregister Service Start Service S	top Service		

- Uses BCDedit instead of boot.ini
- x64 versions starting with XP have **PatchGuard** 
  - $\,\circ\,$  Prevents third-party code from modifying the kernel
  - Including kernel code itself, SSDT, IDT, etc.
  - Can interfere with debugging, because debugger patches code when inserting breakpoints
- There are 64-bit kernel debugging tools

## **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



# END OF LECTURE. THANK YOU.