恶意代码分析与防治技术

第9章 WinDBG内核调试

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院
2023/2024

知识点

- 系统内核与驱动
  - 难点：设备（Device）、驱动（Driver）、物理设备（Physical Device）
- WinDbg
- Microsoft Symbols
- 内核调试实战
- Rootkits
  - 难点：SSDT、IDT

系统内核与驱动

主观题 10分

设置

Windows内核会不会被计算机病毒感染？ 如果可以被感染，举例说明内核病毒有哪些行为。

作答

WinDbg vs. OllyDbg

- What is the difference between WinDbg and OllyDbg?
- Which type of malware we should use WinDbg for analysis?

WinDbg vs. OllyDbg

- OllyDbg is the most popular user-mode debugger for malware analysts
  - Ghidra NSA
  - BinaryNinja
- WinDbg can be used in either user-mode or kernel-mode
- This chapter explores ways to use WinDbg for kernel debugging and rootkit analysis

投票 最多可选1项

设置

Windows内核中的代码是不是全是微软公司开发的?

A 是

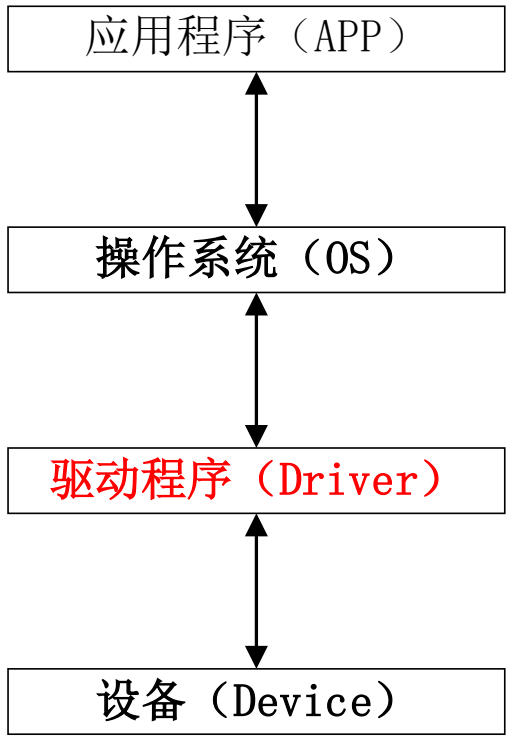
B 不是

提交

南開大學 Nankai University

驱动（Drivers）

- 驱动是一个**软件组件**（Software Component），实现操作系统与设备之间的通信。
  - 应用程序通过Windows API访问设备
  - Windows将访问请求转发给驱动程序，调用驱动程序提供的功能函数
  - 驱动程序由设备制造商开发，完成对设备硬件的访问。



单选题 10分

设置

是不是所有的驱动都是由设备制造商提供的?

A 是

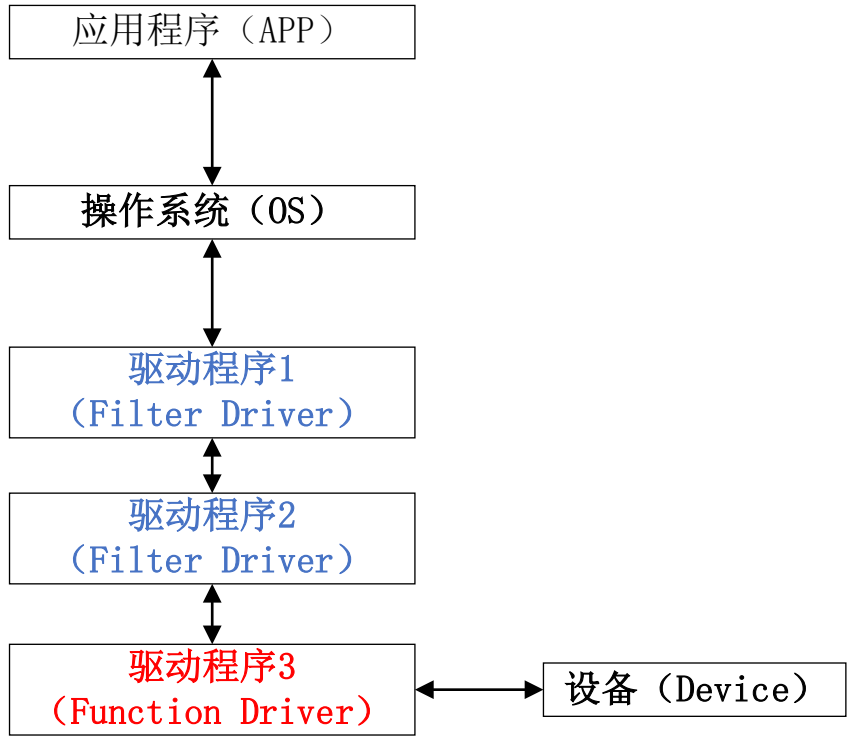
B 不是

提交

南開大學 Nankai University

驱动栈（Driver Stack）

- 并不是所有的驱动都直接与设备进行通信
- 一次设备的访问过程通常会经过多个驱动
  - 驱动栈（Driver Stack）
  - 过滤驱动（Filter Driver）
    - 杀毒软件、防火墙、入侵检测等
  - 功能驱动（Function Driver）
    - 一个设备栈最多只有一个功能驱动



单选题 10分

设置

驱动栈中，过滤驱动一定在功能驱动的上面？

A 正确

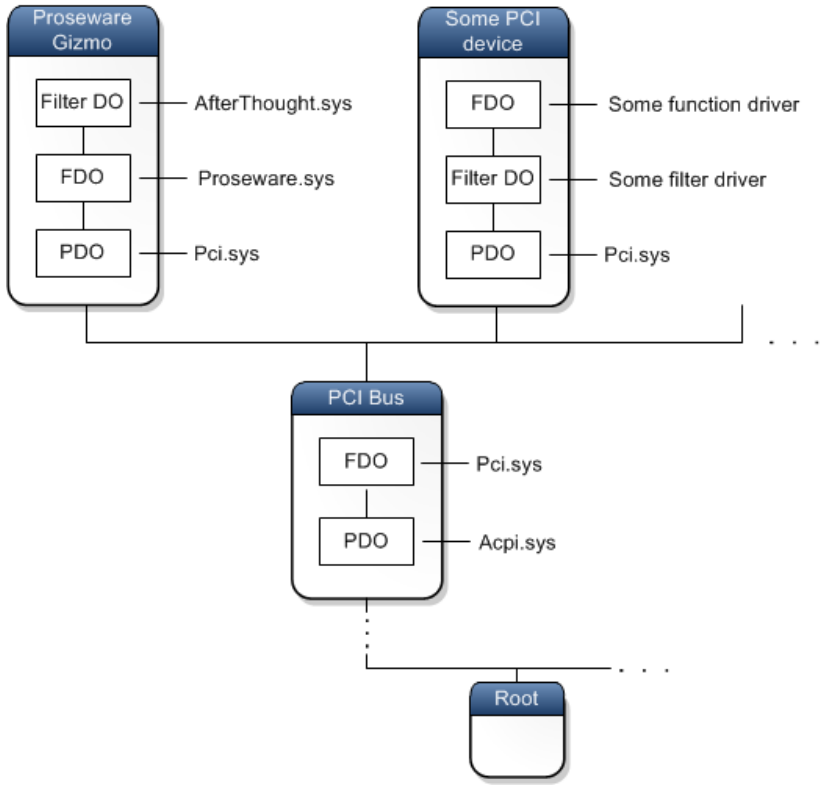
B 错误

提交

南開大學 Nankai University

过滤驱动

- 上层过滤驱动（Upper Filter Driver）
  - 在功能驱动之前的过滤驱动
- 下层过滤驱动（Lower Filter Driver）
  - 在功能驱动之后的过滤驱动



https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/device-nodes-and-device-stacks

单选题 10分

设置

应用程序可以直接访问驱动程序吗?

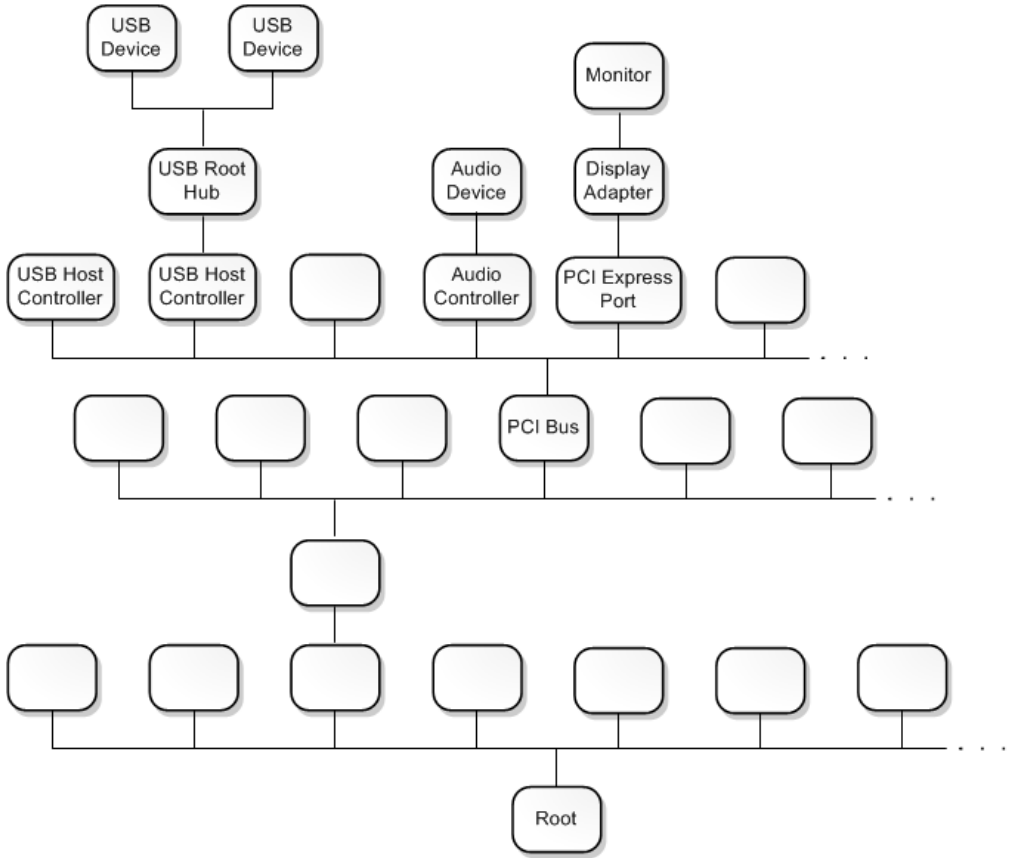
A 可以

B 不可以

提交

设备树

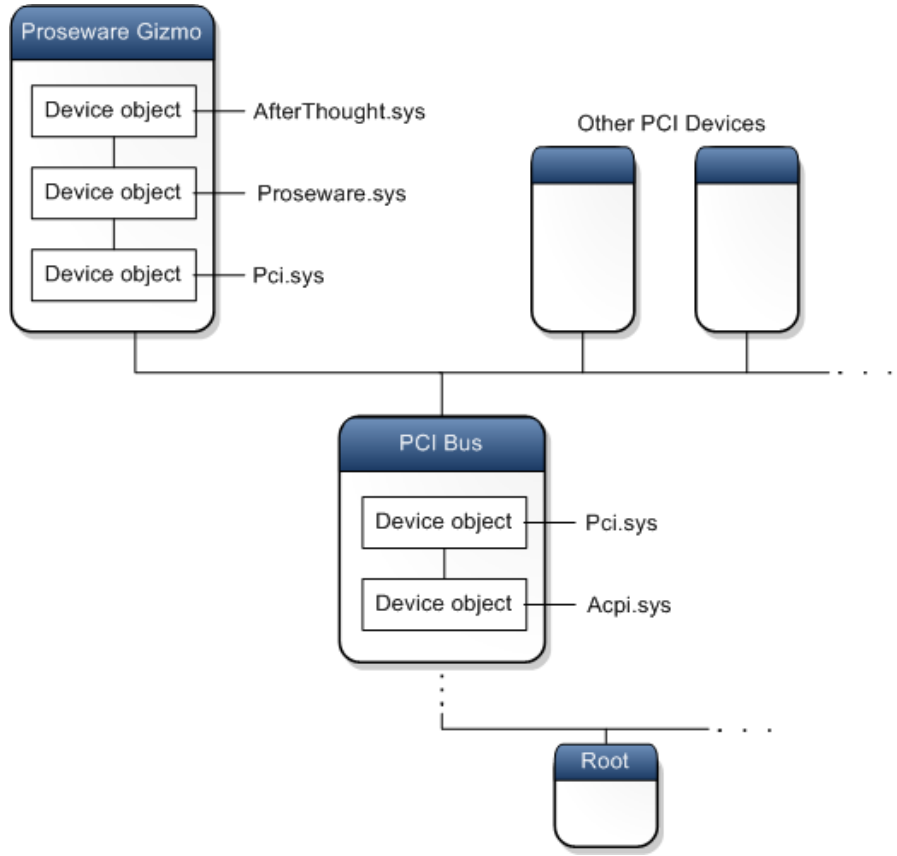
- Windows使用即插即用管理器(Plug and Play Manager)，创建设备树(Device Tree)，来管理系统中的设备
  - 设备节点（Device Node）
    - 不是所有的设备节点都对应物理设备
    - 软件组件
  - 根设备节点（Root Device Node）
    - 根节点在树的最下面
  - 继承关系(Parent/Child Relationships)
    - PCI总线-->USB Host Controller-->USB Root Hub -->USB Device



https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/device-nodes-and-device-stacks

设备栈

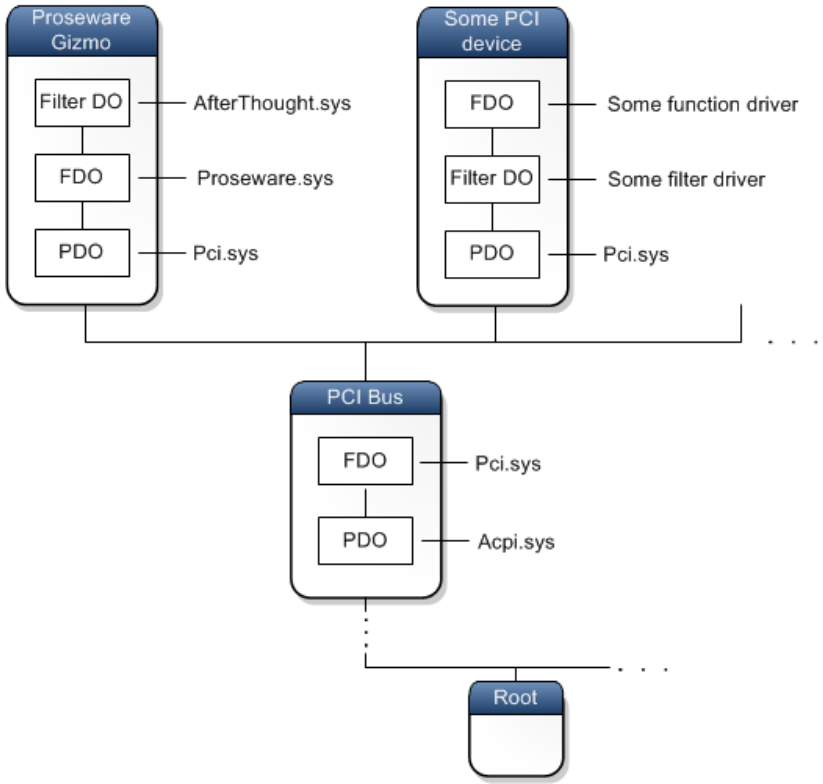
- 每个设备节点是一个有序的设备对象列表
  - 每个设备对象关联一个驱动
  - <设备对象、驱动>
  - Proseware Gizmo设备栈中有3个设备对象
  - PCI Bus设备栈有2个设备对象



https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/device-nodes-and-device-stacks

设备对象（Device Objects）

- PnP管理器控制总线驱动枚举挂在其上的设备
  - 发现的设备创建物理设备对象（Physical Device Object, PDO）
  - 遍历注册表，找到PDO对应的驱动，创建设备栈
  - PDO始终是设备栈的底部设备对象



https://learn.microsoft.com/zh-cn/windows-hardware/drivers/kernel/introduction-to-device-objects

USB Flash Drive

- User plugs in flash drive
- Windows creates the "F: drive" *device object*
- Applications can now make requests to the F: drive
  - They will be sent to the driver for USB flash drives

多选题 10分

设置

下面那个选项可以被用户空间的应用程序直接访问?

A 物理设备 physical hardware

B 设备驱动 device driver

C 设备对象 device object

D Windows 内核

提交

Loading Drivers

- Drivers must be loaded into the kernel
  - Just as DLLs are loaded into processes
- When a driver is first loaded, its **DriverEntry** procedure is called
  - Just like **DLLMain** for DLLs

DriverEntry

- DLLs expose functionality through the export table
- Drivers must register the address for **callback functions**

DriverEntry

- They will be called when a user-space software component requests a service
- DriverEntry performs this registration
  - Windows creates a driver object structure, passes it to DriverEntry which fills it with callback functions
  - DriverEntry then creates a device object that can be accessed from user-land

Example: Normal Read

- Normal read request
  - User-mode application obtains a file handle 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 arbitrary-length buffer of input data
  - Received an arbitrary-length buffer of data as output

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 main hardware components
- Malware will import functions from one or both of these files so it can manipulate the kernel

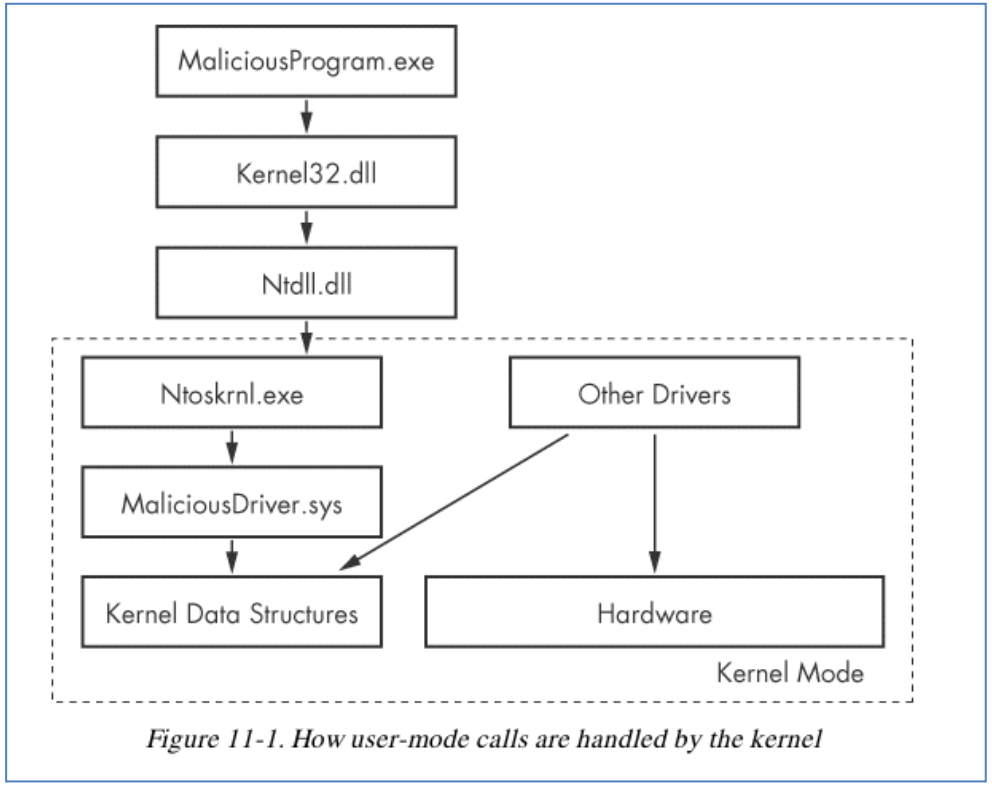


Figure 11-1. How user-mode calls are handled by the kernel

多选题 10分

设置

Which are supported by WinDbg?

A user-mode debugging

B kernel-mode debugging

C rootkit debugging

D application debugging

提交

多选题 10分

设置

Which statements are true for driver?

A creates and destroys device objects

B loaded into kernel

C can be accessed from user space

D has a DriverEntry procedure

提交

南開大學 Nankai University

多选题 10分

设置

Which statements are true for DriverEntry?

A a procedure in driver

B registers callback functions

C creates device

D initializes driver object structure

提交

南開大學 Nankai University

多选题 10分

设置

Which items are usually manipulated by malicious drivers?

A kernel32.dll

B hardware

C ntoskrnl.exe

D hal.dll

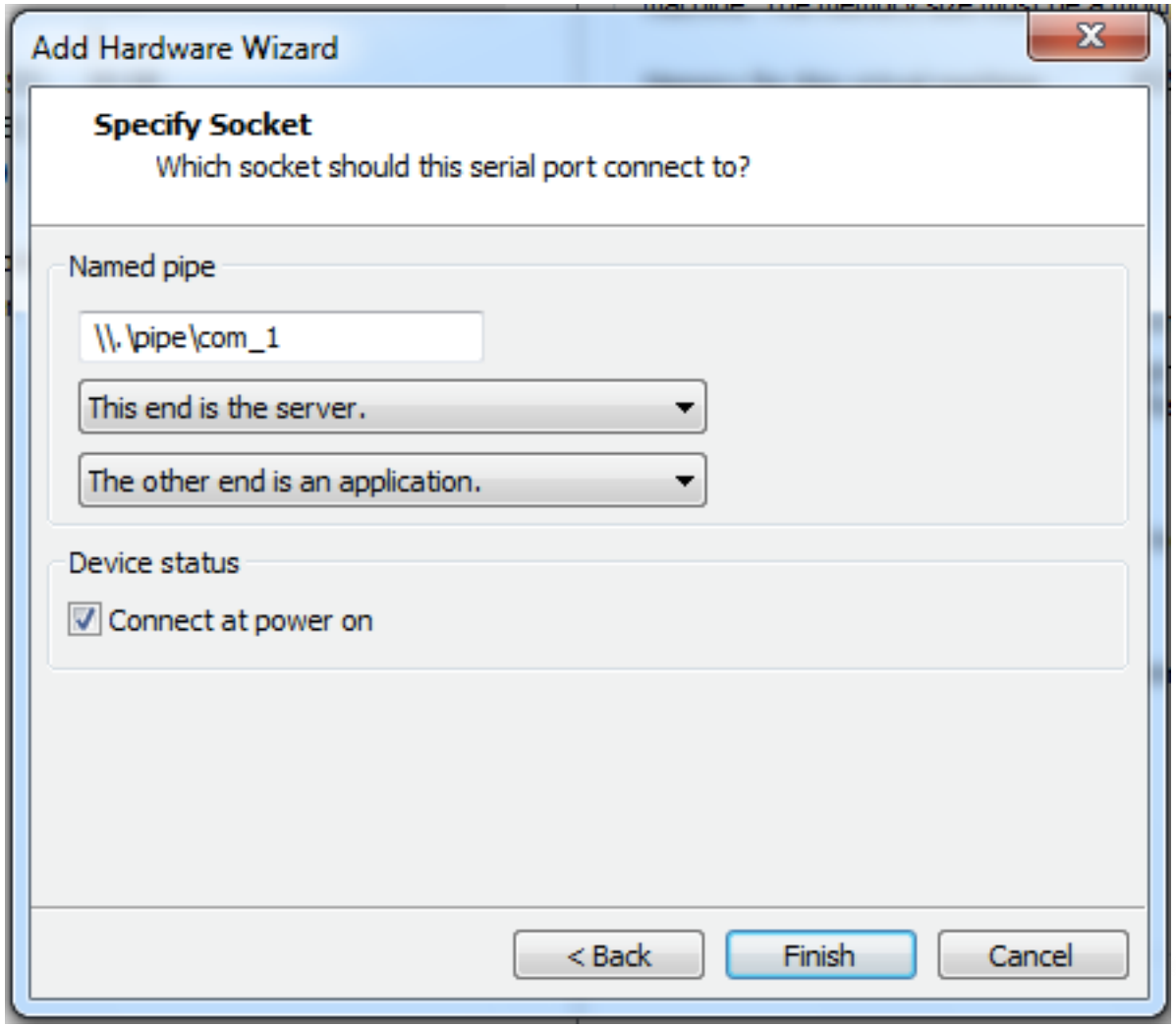
提交

WinDGB

VMware

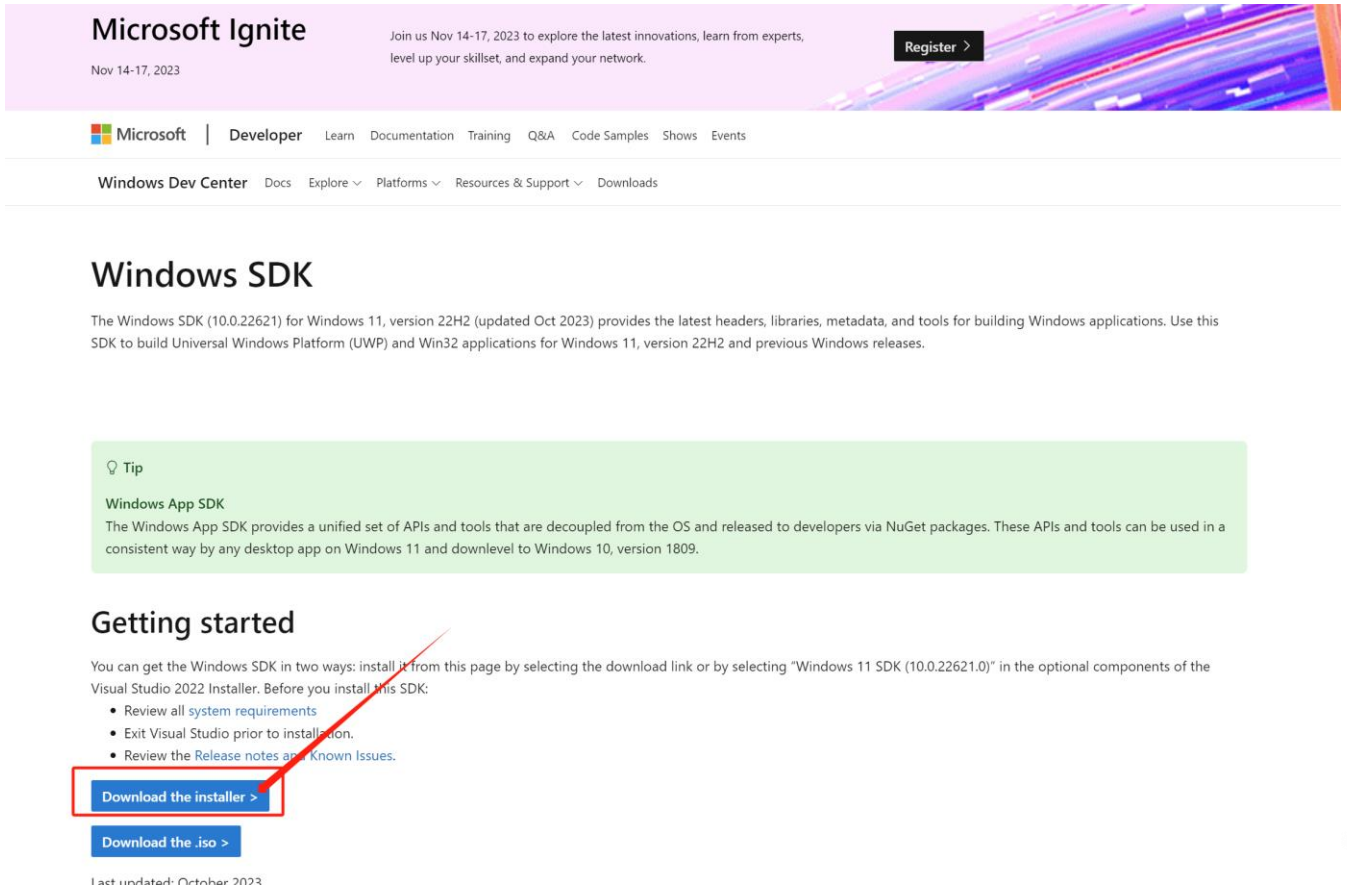
- In the virtual machine, enable kernel debugging
- Configure a virtual serial port between VM and host
- Configure WinDbg on the host machine

Add a Virtual Serial Port

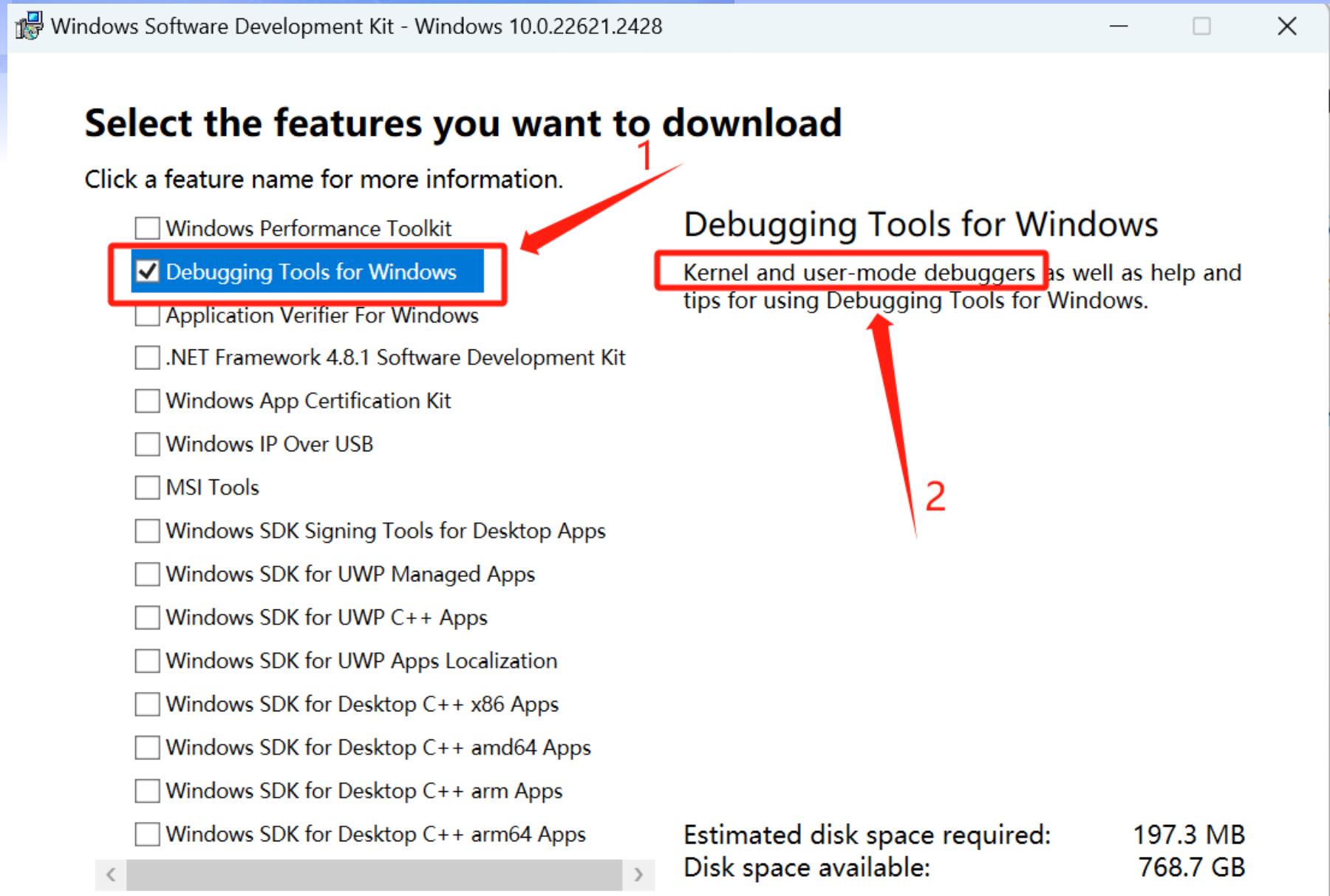


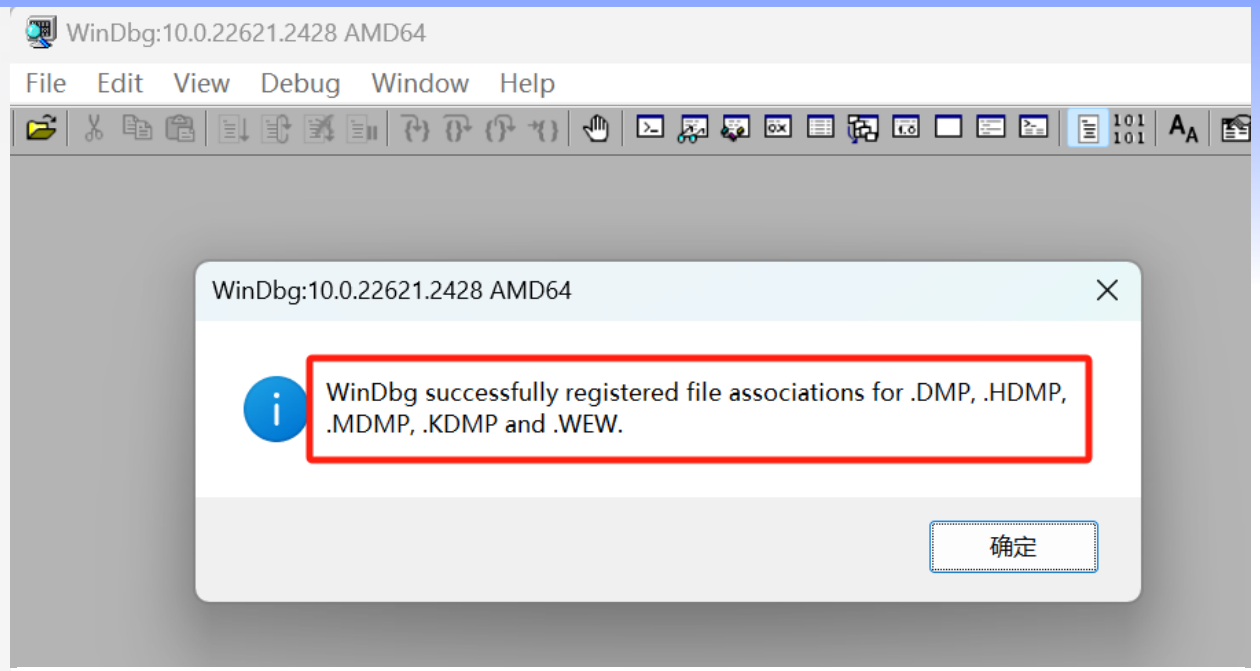
WinDBG 安装

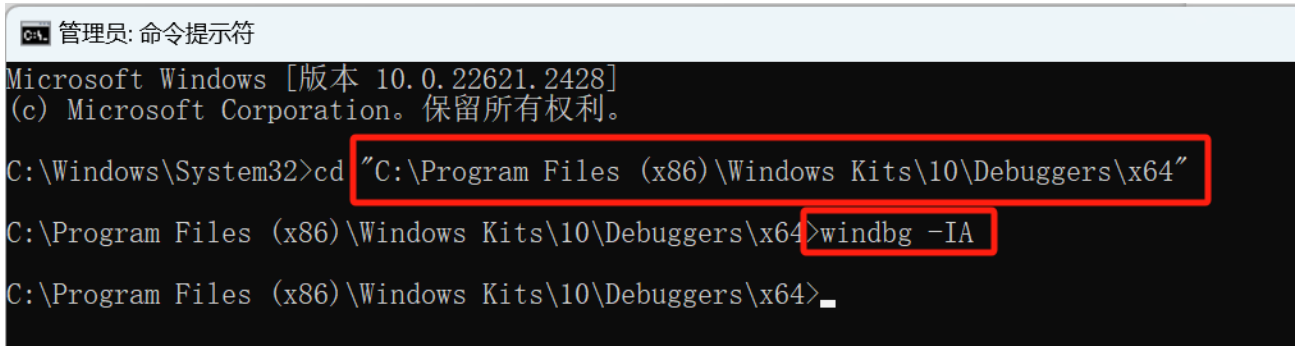
- 下载Windows SDK安装程序 winsdksetup.exe
  - https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/

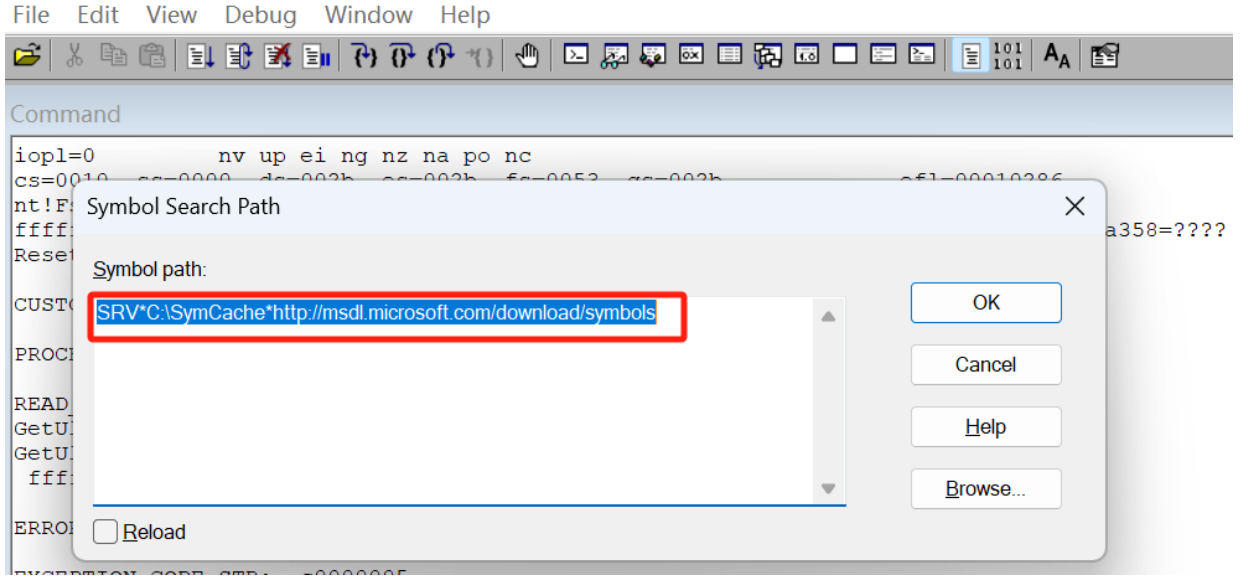


- 运行winsdksetup.exe
  - 安装Debugging Tool sfor Windows
  - Kernel and user-mode debuggers









保存Windows符号路径： File > Save WorkSpace

WinDBG

```
.386
.model flat, stdcall
option casemap :none

include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib

.data
    str_hello BYTE "Hello World!", 0

.code
start:
    invoke StdOut, addr str_hello
    invoke ExitProcess, 0
END start
```

```
PS C:\masm32> .\bin\ml.exe  /c /Zd /coff  .\hello.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: .\hello.asm

**********
ASCII build
**********

PS C:\masm32> .\bin\link.exe /SUBSYSTEM:CONSOLE .\hello.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

LINK : fatal error LNK1104: cannot open file "hello.exe"
PS C:\masm32>
```

列出所有模块

- **lm** 列出当前内存空间装在的所有模块
  - 包括当前内存空间中装载的所有exe和dll程序
  - user-mode 只显示用户空间的模块信息
  - kernel-mode 显示所有的模块信息，包括内核、驱动、用户空间的exe和dll

```
0:000> lm
start             end                 module name
00000000`00400000 00000000`00404000   hello      (deferred)
00000000`77660000 00000000`77811000   ntdll32    (deferred)
00007ffb`4bbf0000 00007ffb`4bc06000   wow64con   (deferred)
00007ffb`4cd00000 00007ffb`4cd8b000   wow64win   (deferred)
00007ffb`4cf80000 00007ffb`4cfd7000   wow64      (deferred)
00007ffb`4cfe0000 00007ffb`4cfe9000   wow64base   (deferred)
00007ffb`4de90000 00007ffb`4e0a7000   ntdll      (pdb symbols)
```

查看模块的PE文件头信息

- !dh 查看指定模型的PE文件头
  - !dh –a hello
  - 入口点的RVA是1000
  - image base是400000h
    - lm命令

```
0:000> !dh -a hello

File Type: EXECUTABLE IMAGE
FILE HEADER VALUES
     14C machine (i386)
       3 number of sections
65479C92 time date stamp Sun Nov  5 21:45:54 2023

       0 file pointer to symbol table
       0 number of symbols
      E0 size of optional header
     10F characteristics
            Relocations stripped
            Executable
            Line numbers stripped
            Symbols stripped
            32 bit word machine

OPTIONAL HEADER VALUES
     10B magic #
    5.12 linker version
     200 size of code
     400 size of initialized data
       0 size of uninitialized data
    1000 address of entry point
    1000 base of code
           ----- new -----
0000000000400000 image base
```

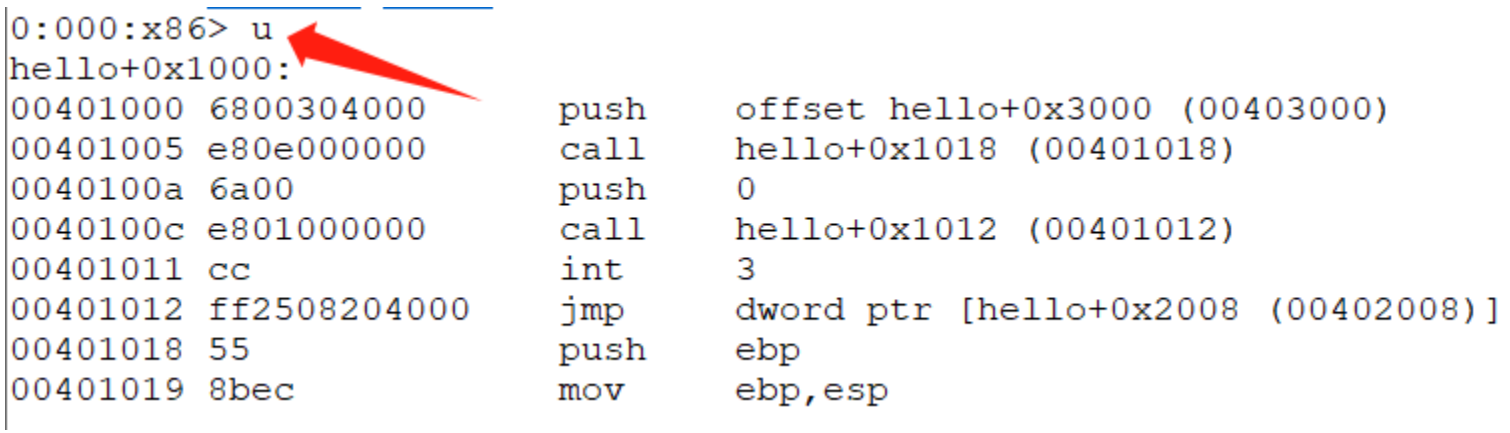
设置断点

- **bp**命令设置内存断点
  - bl命令查看断点列表
  - bc删除断点
  - bd禁用断点
  - be恢复禁用断点
- **g**命令恢复程序的执行，断点处中断程序执行
- **bp 401000**
  - 在hello.exe入口点设置断点

```
0:000> bp 00401000
*** WARNING: Unable to verify checksum for C:\masm32\hello.exe
0:000> g
ModLoad: 00000000`77650000 00000000`7765a000   C:\WINDOWS\System32\w
ModLoad: 00000000`76590000 00000000`76680000   C:\WINDOWS\SysWOW64\K
ModLoad: 00000000`76e40000 00000000`770b4000   C:\WINDOWS\SysWOW64\K
(274c.1e44): WOW64 breakpoint - code 4000001f (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
ntdll32!LdrpDoDebuggerBreak+0x2b:
77718147 cc              int     3
0:000:x86> g
SetContext failed, 0x80004005
Breakpoint 0 hit
hello+0x1000:
00401000 6800304000      push    offset hello+0x3000 (00403000)
```

反汇编

- u命令显示当前地址的反汇编代码
  - u address 显示指定位置反汇编代码



```
0:000:x86> u
hello+0x1000:
00401000 6800304000      push    offset hello+0x3000 (00403000)
00401005 e80e000000      call    hello+0x1018 (00401018)
0040100a 6a00            push    0
0040100c e801000000      call    hello+0x1012 (00401012)
00401011 cc              int     3
00401012 ff2508204000    jmp     dword ptr [hello+0x2008 (00402008)]
00401018 55              push    ebp
00401019 8bec            mov     ebp,esp
```

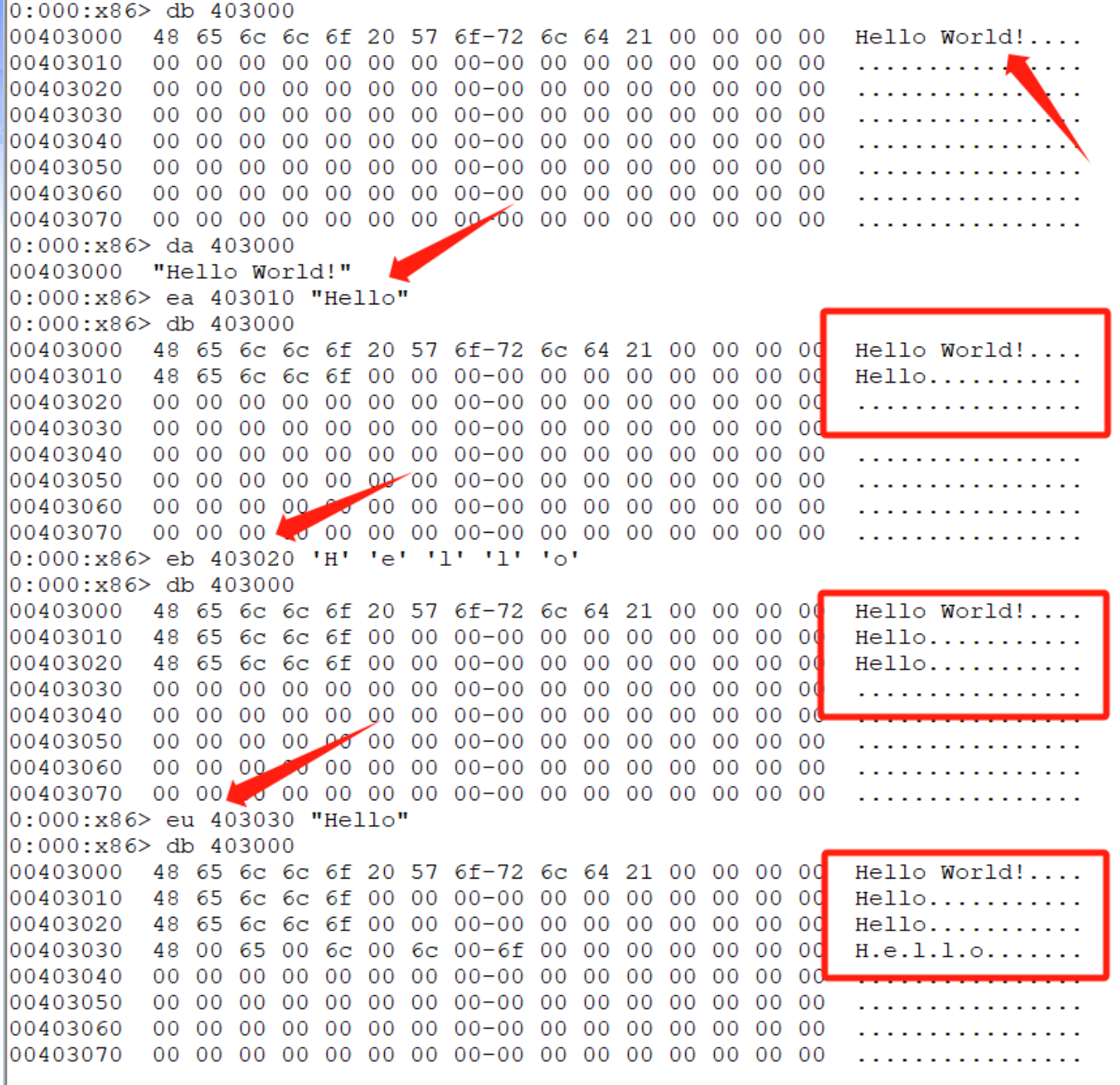
读内存

- d* 显示内存内容
  - da命令以ASCII编码显示
  - du命令以Unicode编码显示
  - db命令以16进制和ASCII编码显示

```
0:000:x86> db 403000
00403000  48 65 6c 6c 6f 20 57 6f-72 6c 64 21 00 00 00 00  Hello World!....
00403010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00403020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00403030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00403040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00403050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00403060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00403070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0:000:x86> da 403000
00403000  "Hello World!"
```

写内存

- e* 将指定值输入到内存中
  - ea命令以ASCII编码写入
  - eu命令以Unicode编码写入
  - eb命令以16进制字节写入



单步执行

- p命令执行一条指令
  - pc执行到下一条call指令
  - pt执行到下一条ret指令
  - pct执行到下一条call或者ret指令

```
0:000:x86> u
hello+0x1000:
00401000 6800304000      push    offset hello+0x3000 (00403000)
00401005 e80e000000      call    hello+0x1018 (00401018)
0040100a 6a00            push    0
0040100c e801000000      call    hello+0x1012 (00401012)
00401011 cc              int     3
00401012 ff2508204000    jmp     dword ptr [hello+0x2008 (00402008)]
00401018 55              push    ebp
00401019 8bec            mov     ebp,esp
0:000:x86> p
SetContext failed, 0x80004005
hello+0x1005:
00401005 e80e000000      call    hello+0x1018 (00401018)
0:000:x86> u
hello+0x1005:
00401005 e80e000000      call    hello+0x1018 (00401018)
0040100a 6a00            push    0
0040100c e801000000      call    hello+0x1012 (00401012)
00401011 cc              int     3
00401012 ff2508204000    jmp     dword ptr [hello+0x2008 (00402008)]
00401018 55              push    ebp
00401019 8bec            mov     ebp,esp
0040101b 83c4f4          add     esp,0FFFFFFF4h
```

栈信息

- dp显示指针指向的内存信息，esp指向栈顶
  - dp @esp， 显示栈信息
  - dpa @esp，以字符串形式显示
  - dps @esp，以符号形式显示
  - dps @esp L4， 显示4行

```
0:000:x86> dpa @esp L4
0019ff74  00403000 "Hello World!"
0019ff78  75d07ba9 "P.....u.....u^..t..."
0019ff7c  003bf000 ""
0019ff80  75d07b90 "..U..V....u..u.....0".u..P.....u.....u^..t..."
0:000:x86> dps @esp L4
0019ff74  00403000 hello+0x3000
0019ff78  75d07ba9 KERNEL32!BaseThreadInitThunk+0x19
0019ff7c  003bf000
0019ff80  75d07b90 KERNEL32!BaseThreadInitThunk
```

进程信息

- |命令显示当前进程ID和进程名
  - 进程ID 3fd0，进程名“C:\masm32\hello.exe”
- ~显示当前进程所有线程的状态
  - 1个线程，线程ID 3fd0.4b88，状态 Suspend，TEB地址 003c0000

```
0:000:x86> |
.  0  id: 3fd0      create name: C:\masm32\hello.exe
0:000:x86> ~
.  0  Id: 3fd0.4b88 Suspend: 1 Teb: 003c0000 Unfrozen
```

单选题 10分

设置

How to read a Unicode string starting at 0x00402000

A da 0x00402000

B du 0x00402000

C ea 0x00402000

D eu 0x00402000

提交

南開大學 Nankai University

单选题 10分

设置

Which command could write a ASCII string into memory?

A da

B ea

C du

D lm

提交

南開大學 Nankai University

单选题 10分

设置

How to set a breakpoint at LoadLibrary?

A bp LoadLibrary

B ex LoadLibrary

C lm LoadLibrary

D da LoadLibray

提交

单选题 10分

设置

Which command could continue execution after breakpoint?

A g

B bp

C da

D eu

提交

Microsoft Symbols

主观题 10分

设置

病毒分析过程为什么需要Microsoft Symbols？ Symbols有什么作用?

作答

对内存地址的符号化表示

- 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)

Deferred Breakpoints

- **bu *newModule*!*exportedFunction***
  - 设置未解析断点
  - 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
```

Example 11-3. Overlaying data onto a structure

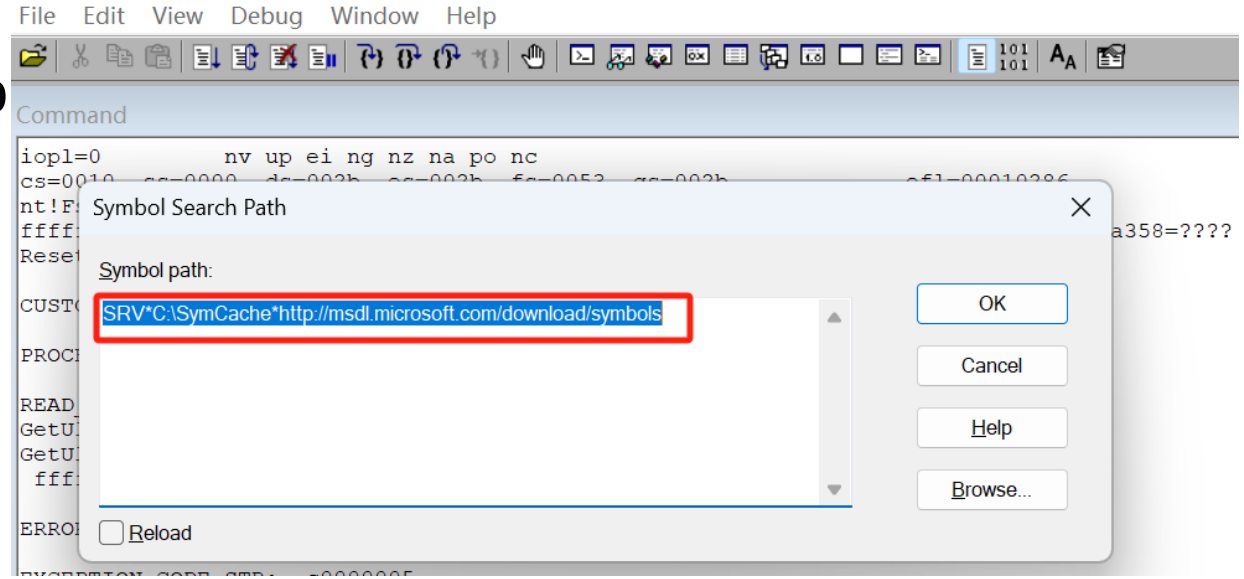
```
kd> dt nt!_DRIVER_OBJECT 828b2648
   +0x000 Type             :  4
   +0x002 Size             :  168
   +0x004 DeviceObject     :  0x828b0a30 _DEVICE_OBJECT
   +0x008 Flags            :  0x12
   +0x00c DriverStart      :  0xf7adb000
   +0x010 DriverSize       :  0x1080
   +0x014 DriverSection    :  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
- 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 Microsoft as needed
- They are cached locally
- **File, Symbol File Path**
  - **`SRC*c:\websymbols*http://msdl.microsoft.com/download/symbols`**



单选题 10分

设置

Which module’s name is nt in WinDbg?

A kernel32.dll

B ntoskrnl.exe

C hal.dll

D ntdll.dll

提交

南開大學 Nankai University

单选题 10分

设置

How to list closest symbol at a specified address?

A ln

B lm

C du

D dt

提交

单选题 10分

设置

How to view a structure information at a specified memory address?

A dt

B da

C du

D dd

提交

南開大學 Nankai University

单选题 10分

设置

Which function is called first when a driver is loaded?

A DriverInit

B DriverEntry

C DllMain

D WinMain

提交

南開大學 Nankai University

内核调试实战

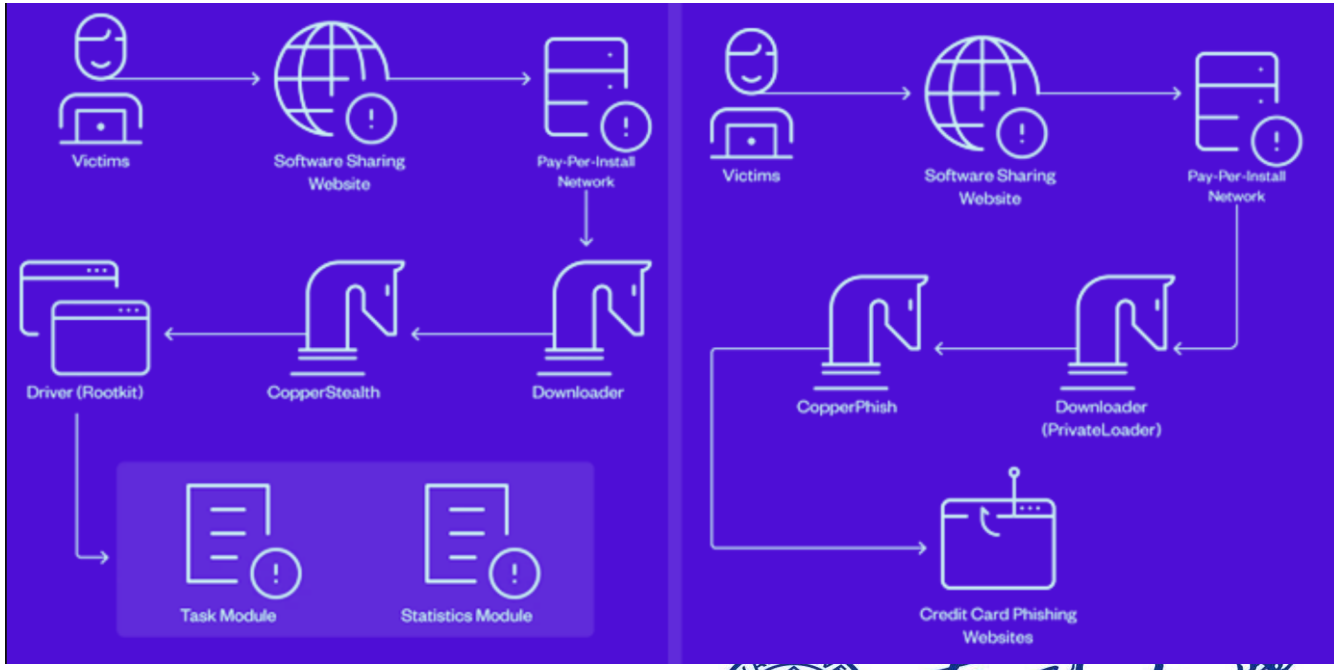
恶意代码的发展趋势

- “我们已进入与高级威胁对抗时代，需要通过底层技术对抗这类威胁”，“由已知恶意代码向高级恶意代码发展，包括Rootkit、Bootkit等高级内核级攻击”—朱颖（安芯网盾）遇见2023网安行业发展新趋势

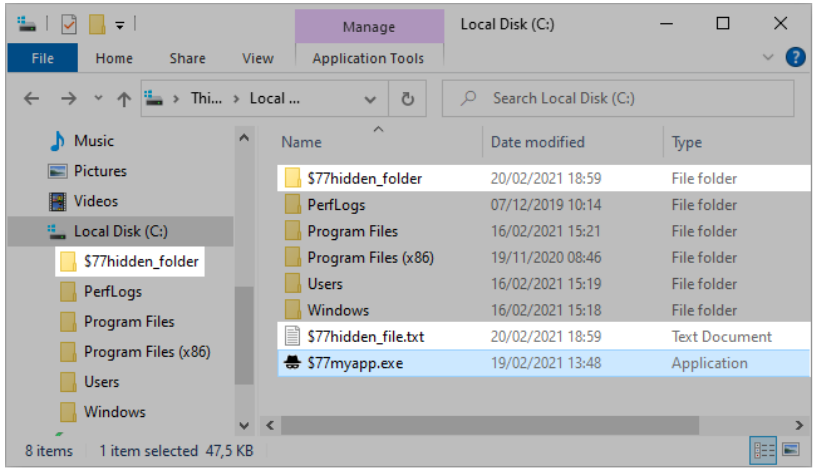
CopperStealer恶意软件使用新的Rootkit和网络钓鱼套件模块

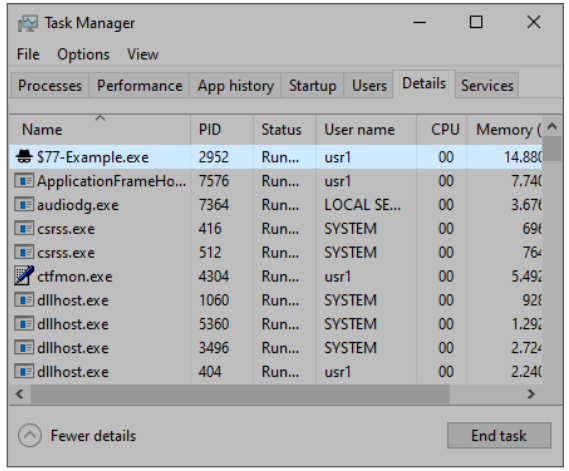
日期：2023-05-22
标签：CopperStealer恶意软件

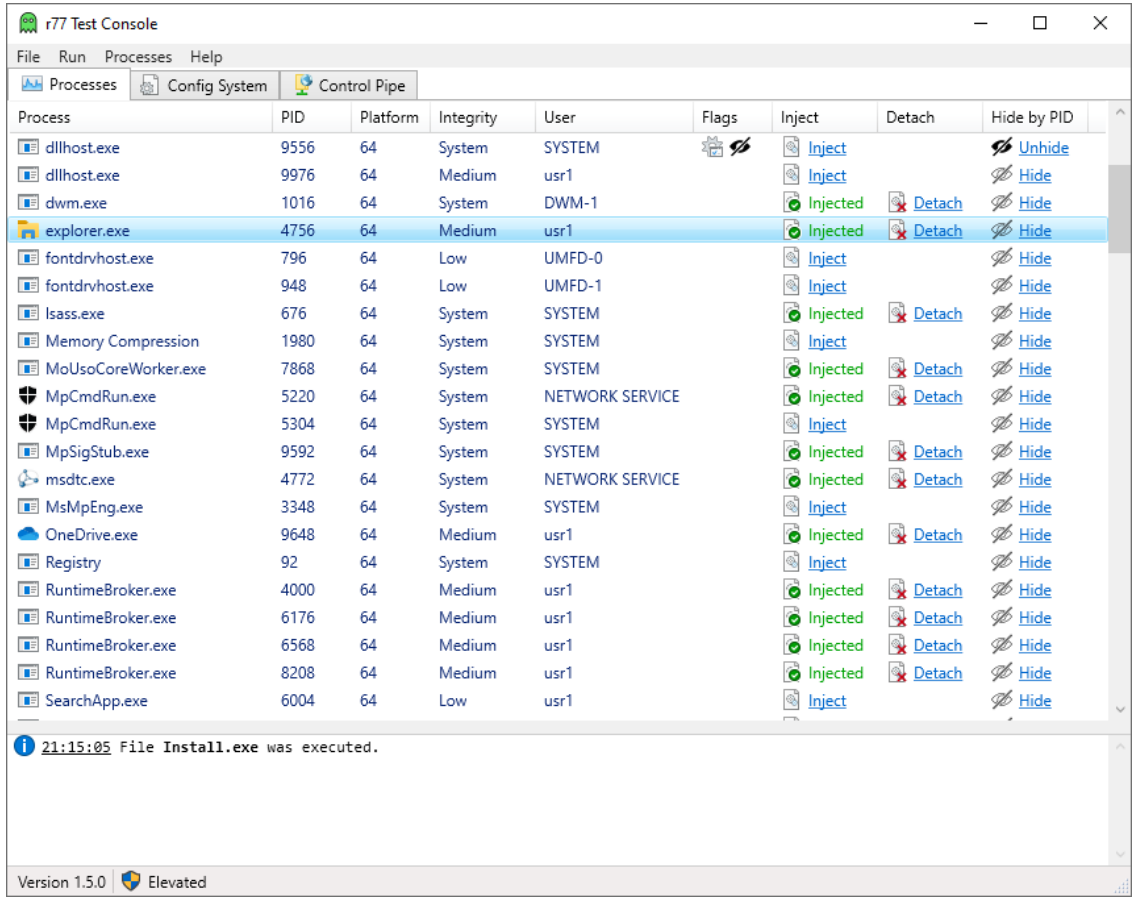
网络安全公司Trend Micro在2023年3月和4月发现CopperStealer恶意软件的威胁者重新出现，使用新负载CopperStealth和CopperPhish进行两个攻击活动。Trend Micro将这一以金钱为动机的组织命名为Water Orthrus。这个对手还负责另一个被称为Scranos的活动。Water Orthrus自2021年以来一直活跃，在使用“按装获利(PPI)”网络将被攻击者拦截的受害者重定向到破解软件下载站，在那里释放被称为CopperStealer的信息窃取软件的过程中拥有丰富的经验。最新的攻击活动使用了与前面攻击类似的技术，通过包装成中国软件共享网站的免费工具安装器来传播CopperStealth。CopperPhish恶意软件则通过PPI网络以类似的方式传播，利用网络钓鱼攻击窃取信用卡信息。这些攻击活动是威胁者策略演化的一部分，表明他们正在尝试添加新的工具来扩展其金融收益。

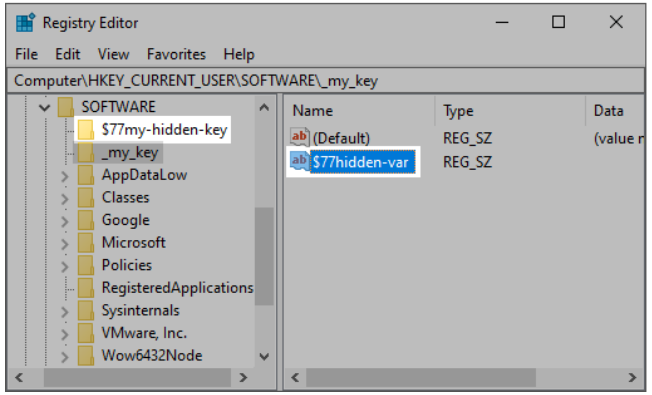


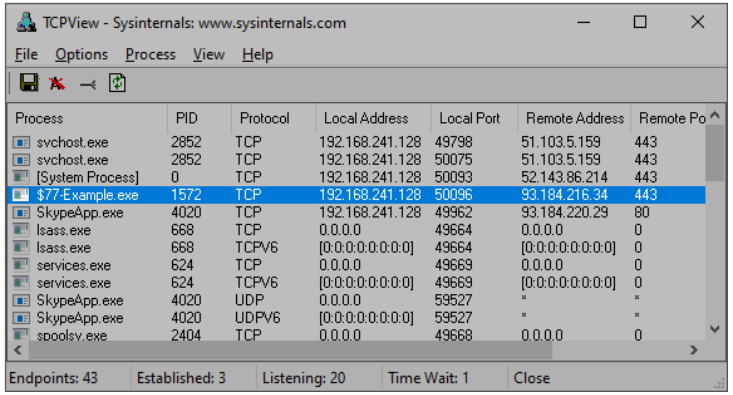
R77 Rootkit











https://github.com/bytecode77/r77-rootkit

全民携手 共筑网络安全坚强防线

主观题 10分

设置

2023年国家网络安全宣传周的主题是什么？

作答

南開大學 Nankai University

国家网络安全宣传周


国家网络安全
宣传周
China Cybersecurity Week
网络安全为人民
网络安全靠人民
2023年国家网络安全宣传周
主办单位：
中央宣传部、中央网信办、教育部、工业和信息化部、公安部、
中国人民银行、国家广播电视总局、全国总工会、共青团中央、全国妇联

Kernel Mode and User Mode Functions

- We'll examine a program that writes to files from kernel space
  - Kernel mode programs cannot call user-mode functions like **CreateFile** and **WriteFile**
  - Must use **NtCreateFile** and **NtWriteFile**

User-Space Code

Example 11-4. 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    11                  ; dwServiceType
04001B4B  push    0F01FFh             ; dwDesiredAccess
04001B50  push    [ebp+lpDisplayName] ; lpDisplayName
04001B53  push    [ebp+lpDisplayName] ; lpServiceName
04001B56  push    [ebp+hSCManager] ; hSCManager
04001B59  call    ds:__imp__CreateServiceA@52
```

Creates a service with the CreateService function

dwServiceType is 0x01 (Kernel driver)

User-Space Code

Example 11-5. Obtaining a handle to a device object

```
04001893                xor     eax, eax
04001895                push    eax             ; hTemplateFile
04001896                push    80h             ; dwFlagsAndAttributes
0400189B                push    2               ; dwCreationDisposition
0400189D                push    eax             ; lpSecurityAttributes
0400189E                push    eax             ; dwShareMode
0400189F                push    ebx             ; dwDesiredAccess
040018A0                push    edi             ; lpFileName
040018A1                call    esi ; CreateFileA
```

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

User-Space Code

Once the malware has a handle to the device, it uses the `DeviceIoControl` function at 1 to send data to the driver as shown in Example 11-6.

Example 11-6. Using `DeviceIoControl` to communicate from user space to kernel space

```
04001910  push    0                ; lpOverlapped
04001912  sub     eax, ecx
04001914  lea     ecx, [ebp+BytesReturned]
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:DeviceIoControl1
```

Kernel-Mode Code

- Set kernel-mode debugger to Verbose mode
- You'll see every kernel module that loads
- Kernel modules are not loaded or unloaded often
  - **Any loads are suspicious**
  - Except **Kmixer.sys** in VMware machines

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
```

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 *driver object*
- Use **!devhandles** to find application that use the driver

填空题 10分

设置

Kernel mode programs [填空1] (can or cannot) call user-mode functions

正常使用填空题需3.0以上版本雨课堂

作答

Rootkits

Rootkit Basics

- Rootkits could modify the internal functionality of the OS to conceal themselves
  - Hide processes, files, network connections, drivers, services and other os components.
  - Reside at the user or kernel level in the operating system or lower, including hypervisor, Master Boot Record, or System Firmware.
  - Difficult for antivirus, administrators, and security analysts to discover their malicious activity

Rootkit Basics

- Most rootkits modify the OS 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

SSDT

- 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

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 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!

Rootkit Analysis in Practice

- **lm m nt**
  - Lists modules matching "nt" (Kernel modules)
  - Shows the SSDT table

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

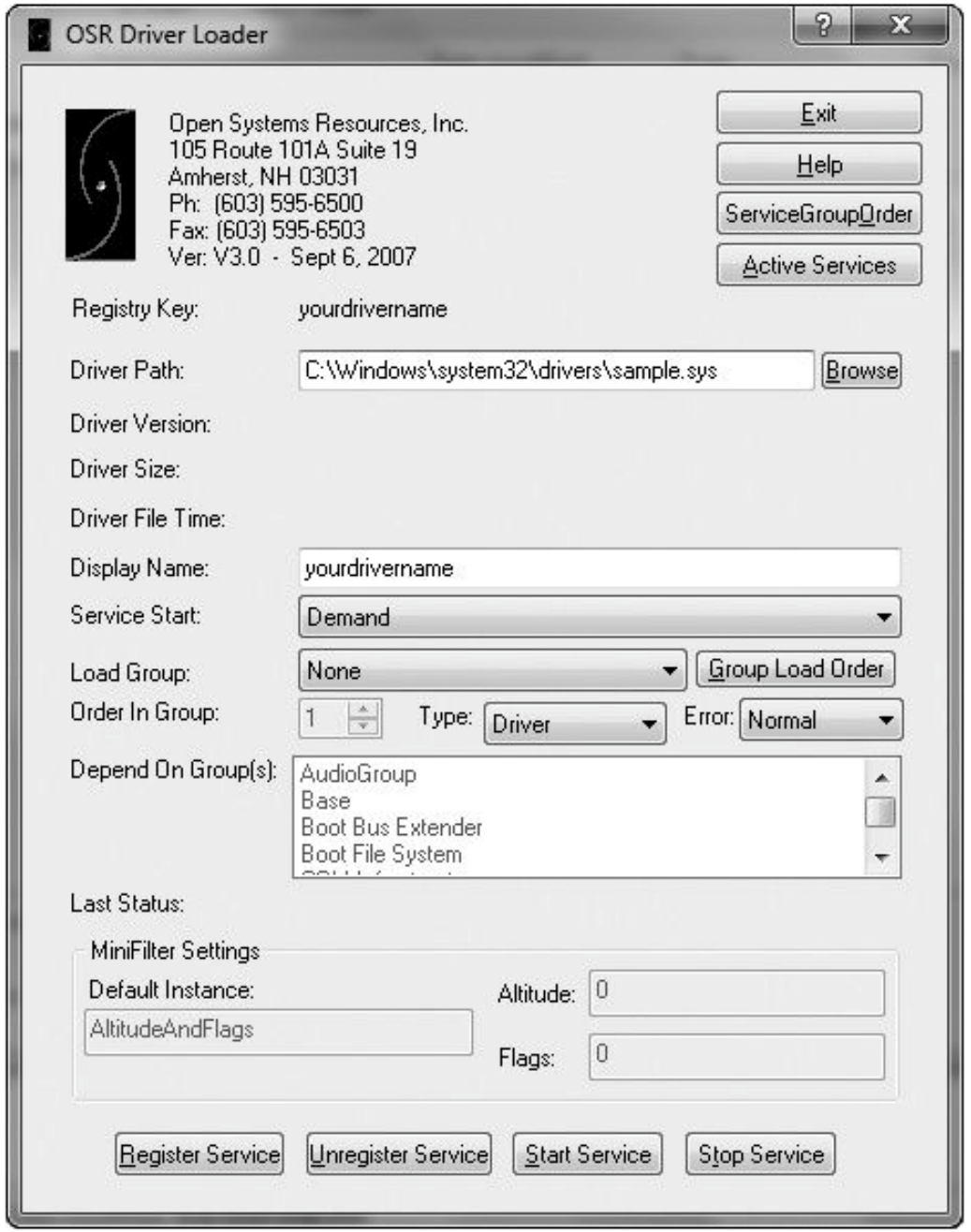
```
kd> !idt

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:   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



Driver Signing

- Enforced in all 64-bit versions of Windows starting with Vista
- Only digitally signed drivers will load
- Effective protection!

R77 隐藏效果的验证实验

- 运行R77程序，实现对指定的进程、文件、注册表、网络连接的隐藏。
- 实验效果进行截图，完成实验报告
- R77使用了Windows的Detours机制。Detours机制的原理将在隐蔽执行章节介绍。

恶意代码分析与防治技术

第9章 WinDBG内核调试

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院
2023/2024

知识点

- 系统内核与驱动
  - 难点：设备（Device）、驱动（Driver）、物理设备（Physical Device）
- WinDbg
- Microsoft Symbols
- 内核调试实战
- Rootkits
  - 难点：SSDT、IDT