

# Beyond the Surface: A Comprehensive Look at Windows Driver Security Analysis

Let us talk about low level security issues

Dr Baptiste David

[bdavid@ernw.de](mailto:bdavid@ernw.de)

RELEASE VERSION

COCHIN – INDIA – 10/2023

# Who am I?


- Dr David Baptiste
- I am  and I work in 
- **ERNW Enno Rey Netzwerke GmbH**
  - Computer security service in Heidelberg, Germany
  - Independent IT Security Solutions Provider
  - “Make the World a Safer Place!”
- Did many conferences
  - Black Hat USA, DefCon, Malcon, ClubHack, C0c0n
  - We organize Troopers Conference 😊

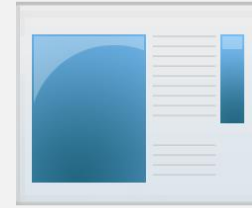




# Windows Device drivers

Let's introduce the notion, with why, how and what.

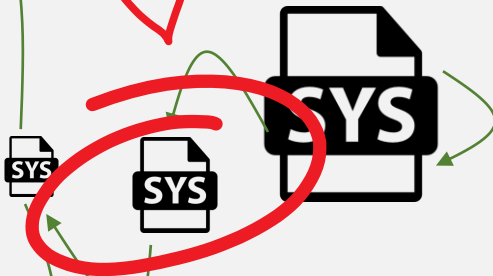
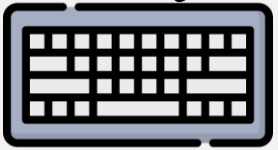
Manufacturer  
Device  
Drivers 



application.exe

Ring 3

Ring 0



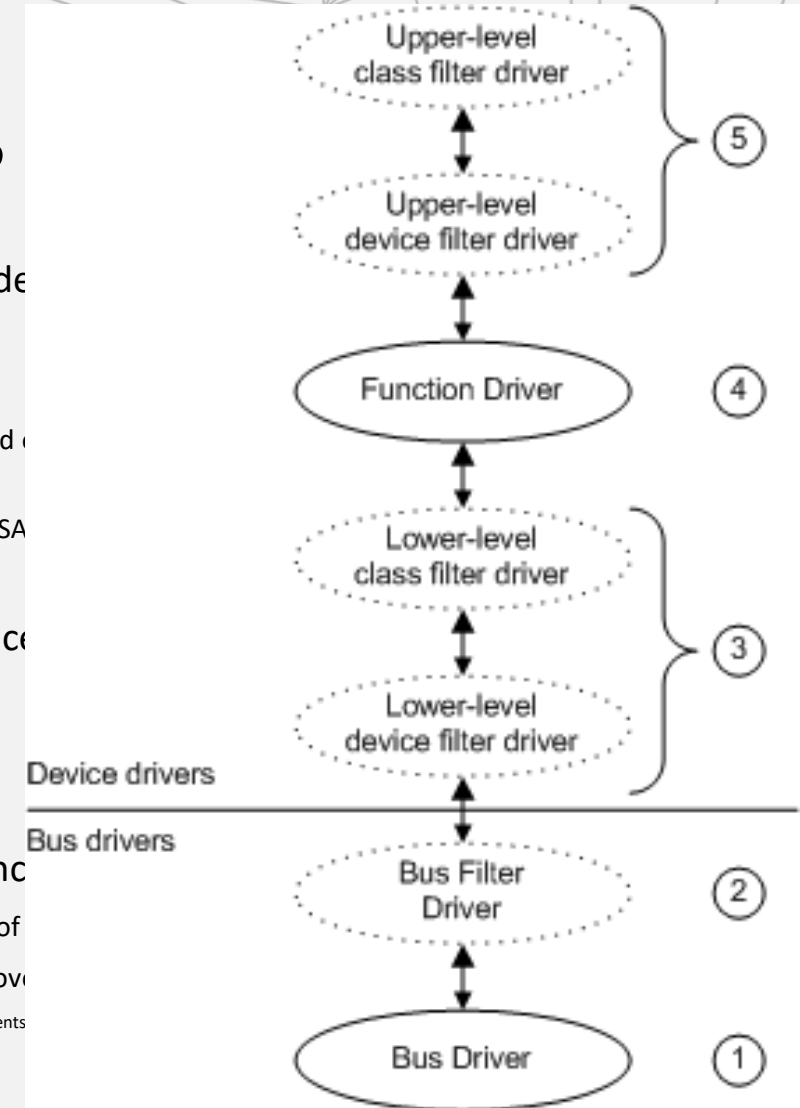
# Manufacturers' device drivers

- Do you need to write a driver?
  - Microsoft Windows contains built-in drivers for many device types.
    - If there is a built-in driver for your device type, you will not need to write your own driver.
    - Your device can use the built-in drivers.
      - Default protocols (USB, network, bus management, display, ...)
      - Human Interface Devices - *HID* (self-adapting protocol, transparently managed by embedded drivers)
  - But some device's manufacturers do it.
    - Supporting unsupported device types.
    - Adding some extra features to existing drivers.
    - ~~Adding some extra vulnerabilities to a system ...~~ 😬



# Different kinds of device drivers

- **Bus driver** manages an individual logical or a physical I/O bus device and it provides independent.
  - It multiplexes access to the bus and how the information is exchanged.
  - It is more a transporter of information. Bus drivers also detect and report to the PnP devices that are connected to the power setting of the bus.
  - Examples of buses include PCI, IDE Controller, Simple Peripheral Bus (SPB), Secure Digital (SD) Card, USB, PnpISA IEEE 1394.
- **Function driver** manages a particular type of device by providing the operational interface.
  - It is the main driver for a device.
  - Typically written by the device vendor,
  - A single function driver can service one or more devices.
- **Filter driver** are optional drivers that interface with a specific driver of a device to enhance its functionality.
  - Able to serve one or more device at once, this one is inserted between two logical layers of drivers in the stack of drivers.
  - Considering the stack growing up from the bus driver to the function driver, this insertion can be performed above or below the function driver.
    - **Bus Filter Drivers** or **Lower-Level Filter Drivers**: Used to add value on top of a bus-driver and they can be used, for example, to implement proprietary enhancements to the bus.
    - **Lower-Level Filter Drivers**: Used to monitor or to modify (to match expected specifications usually) the behavior of device hardware.
    - **Upper-Level Filter Drivers**: Used to add value to the device.



A big structure holding all settings  
about the driver.

```
NTSTATUS DriverEntry(  
    _In_ PDRIVER_OBJECT DriverObject,  
    _In_ PUNICODE_STRING RegistryPath  
);
```

The path to the driver's  
configuration in the registry.

# Device Driver Interface

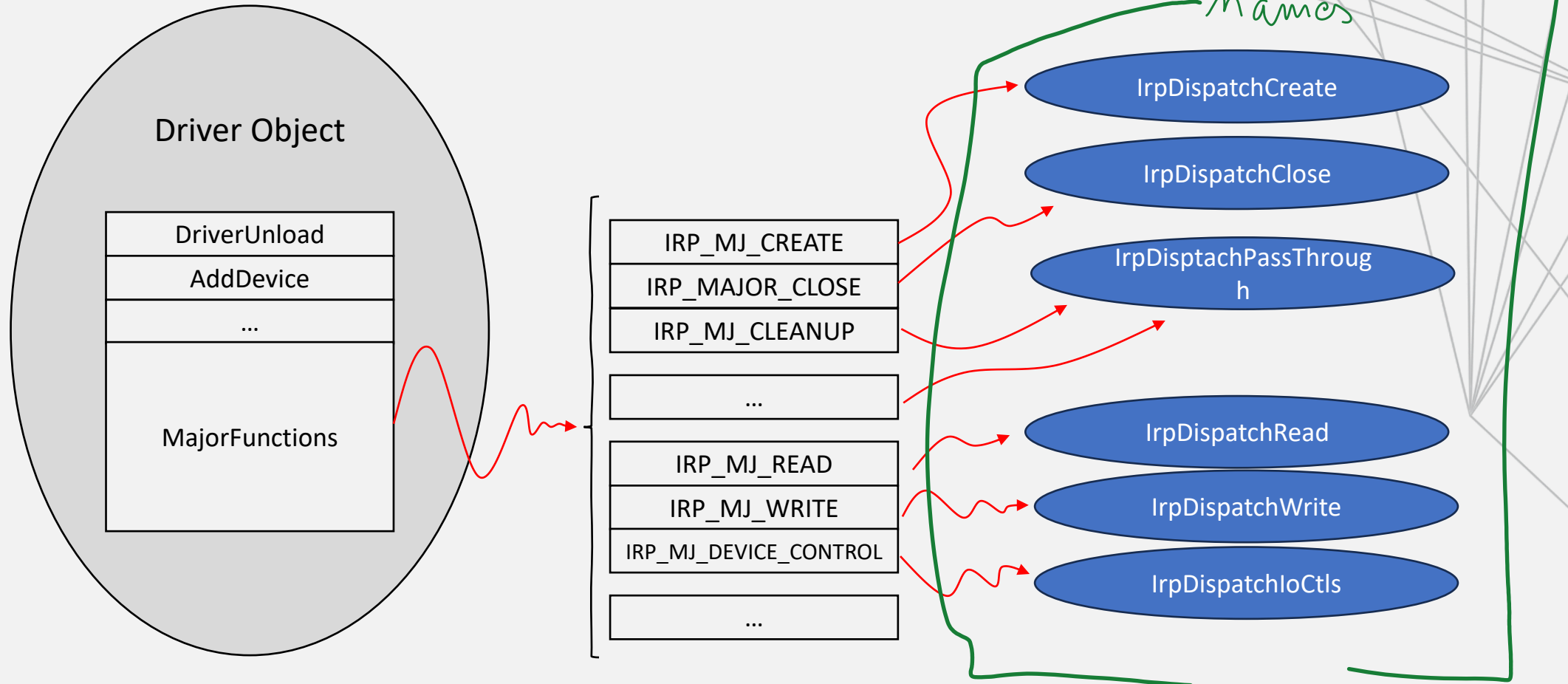
- A driver has an entry point (as any program 😊).
  - This one is called the [DriverEntry](#) function.
  - This entry point is used to:
    - Create and/or initialize various driver-wide objects, types, or resources the driver uses.
      - For a “per-device” approach, this is the [AddDevice](#) routine that should be used.
    - Load the configuration from the registry of Windows (if any).
    - Supply entry points for the driver's standard routines.
      - Driver's [AddDevice](#) routine, dispatch routines, [StartIo](#) routine, [DispatchPnP](#), [DispatchPower](#), and [Unload](#) routine.
    - Implement a device interface if any.
    - Return NTSTATUS indicating whether the driver successfully loaded and can accept and process requests from the PnP manager to configure, add, and start its devices.

# Device Driver interface implementation

- The goal is to [create a device object](#).
  - Usually one for each physical, logical, or virtual device it interfaces.
    - Usually, it creates one to offer an interface with the driver itself.
  - Used to handle “messages” forwarded through the driver: **I/O Requests (IRP)**.
- Usually, the [IoCreateDevice](#) function is used to create a device object.
  - It is highly recommended to use the [IoCreateDeviceSecure](#) function.
  - It is possible to control the access to the device object.
    - When the PnP manager calls the driver's [AddDevice](#) routine.
    - If the device has a security descriptor setting in the registry, it is applied to every object in the device stack.
    - Specify the [default security descriptor](#) and class GUID for that device.
      - For instance: "D:P(A;;GA;;;SY)(A;;GA;;;BA),, : Limits access to system and admins only.
      - By default: “free to play for every logged user”.

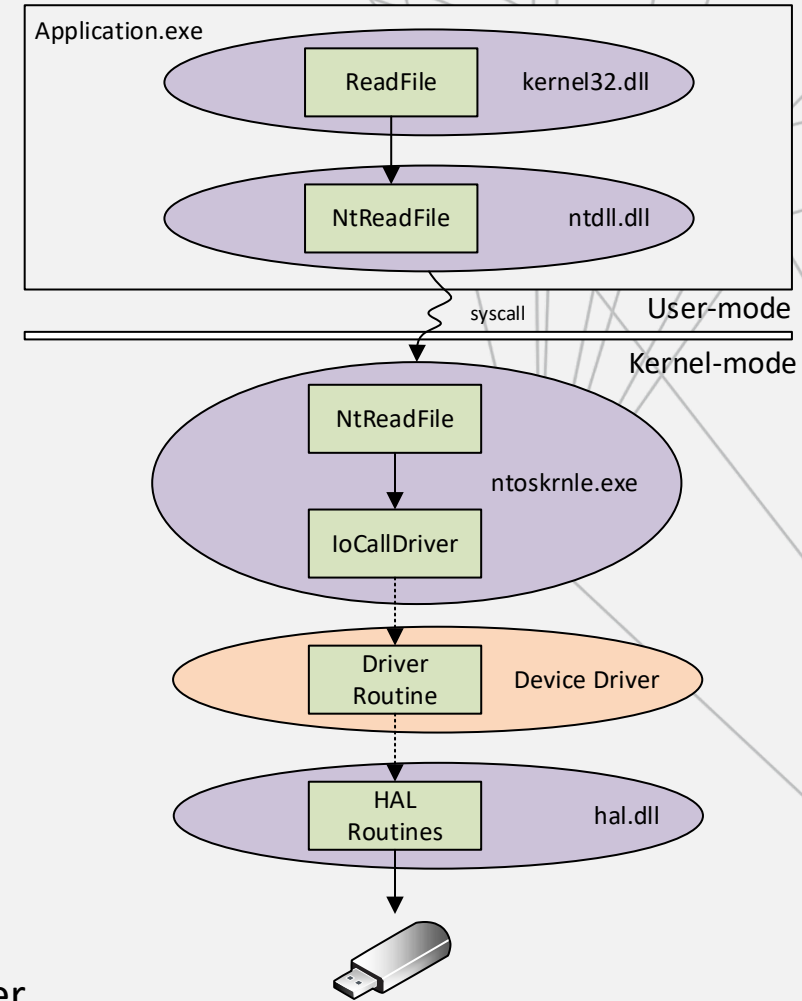
Here comes the fun 😁





# Interfacing with a device driver

- Interfacing with a driver is like ... interfacing with a file.
  - Getting access to the driver with [CreateFile](#) function.
    - Open the MS-DOS name 😊.
    - This is where access check is performed.
      - Security Descriptor during device object creation.
      - IRP\_MJ\_CREATE dispatcher routine.
  - Interface with the driver.
    - With [ReadFile](#) / [WriteFile](#).
      - IRP\_MJ\_READ / IRP\_MJ\_WRITE.
    - The [DeviceIoControl](#) function is the most generic one.
      - Handled within the IRP\_MJ\_DEVICE\_CONTROL dispatcher.
      - Also, there is the IRP\_MJ\_INTERNAL\_DEVICE\_CONTROL dispatcher.



The handle to the driver previously retrieved with CreateFile.

```

BOOL DeviceIoControl(
    [in] HANDLE hDevice,
    [in] DWORD dwIoControlCode,
    [in, optional] LPVOID lpInBuffer,
    [in] DWORD nInBufferSize,
    [out, optional] LPVOID lpOutBuffer,
    [in] DWORD nOutBufferSize,
    [out, optional] LPDWORD lpBytesReturned,
    [in, out, optional] LPOVERLAPPED lpOverlapped
);

```

Public ones, defined by Microsoft for specific documented operations (SCSI Port, USB, device management ...).

Called **IOCTLs**, it is just a regular "value". In practice, there are two kinds of IOCTLs.

Private ones, defined by vendor's software for internal communication. They are almost never documented.

Overlapped structure, useful when the operation must be performed asynchronously.

Output buffer, as returned by the driver's dispatch routine. There is a size provided and size returned, allowing padding.

Input buffer provided. Usually an address to a structure, with the size of the buffer, in bytes.

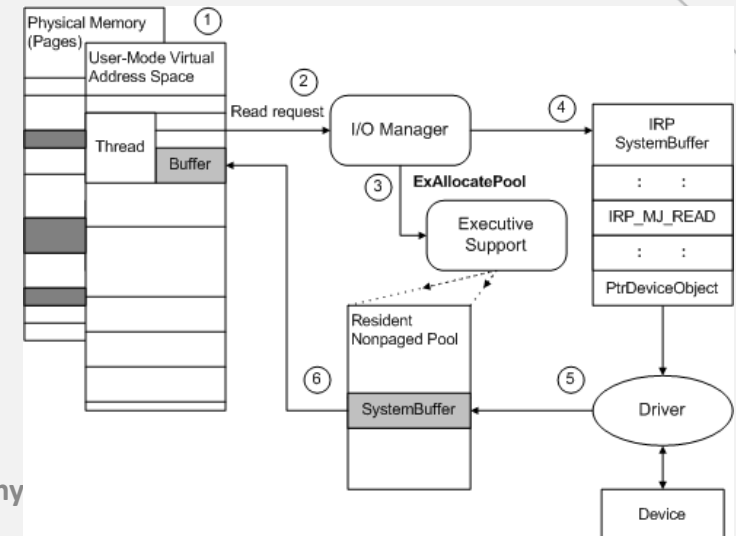
# Methods for Accessing Data Buffers

- When interfacing with DeviceloControl, the user-mode application ...
  - Provides user-mode buffers (address  $\leq 0x7FFF0000$  |  $0x7FFFFFFF$  `FF0000).
  - Interfacing them in kernel mode requires:
    - Executing in the **context** of the **calling process**.
    - Retrieving them in a **secure** way is necessary!
      - Providing kernel-addresses “just to test”.
      - Time-to-check vs time-to-use (**TOCTOU**).
      - (...)
- There are many ways to access data buffers in the context of IOCTLs.
  - Buffered I/O
  - Direct I/O
  - Neither Buffered Nor Direct I/O

# Methods for Accessing Data Buffers

## • Buffered I/O

- The system creates a nonpaged system buffer, equal in size to the application's buffer.
  - User-mode content is copied in an allocated kernel-mode memory.
  - This operation is performed for input buffers and for output buffers at the end.
  - This is the most convenient way to proceed for small and interactive transfers.
    - Video, keyboard, mouse, serial, and parallel drivers.
- This is the best approach for security.
  - “Everything” is performed by the kernel.
  - But it has a real impact about performances.
  - Not suitable for “real time” or hardware scenarios.





# Methods for Accessing Data Buffers

- **Neither Buffered Nor Direct I/O**

- The I/O manager passes the **original user-space virtual addresses** in IRPs that it sends to the driver.
  - Must be executed in the context of the calling process.
  - Not waste of time to allocate/copy content into system buffer.
  - But highly dangerous 😱.
    - This is why it's widely used 😊.
- The security of the buffer is driver's responsibility.
  - This is not so easy, and, in the end, it is about reimplementing the Buffer I/O method.
  - There is a [tutorial](#) about how to do that correctly.
  - But it is far to be obvious to do it correctly ... 🤔

# Methods for Accessing Data Buffers

- How do we secure access to data buffers in neither method?
  - Check the **validity** of the user **buffer's address range**.
    - User-mode addresses are  $[0x10000 \leq \text{address} \leq 0x7FFF0000 \mid 0x7FFFFFFF \text{..} 0x0000]$ .
    - They must be aligned in memory  $[0xXXXX1, 0xXXXX2, 0xXXXX3, \dots]$  are invalid.
  - Check whether the appropriate **read or write access** is permitted.
    - With [ProbeForRead](#) and [ProbeForWrite](#) support routines.
  - Enclose access to the buffer's address range within a driver-supplied **exception handler** (`__try / __except(.) { (...) }`).
    - User thread could change the access rights for the buffer while the driver is accessing memory 😬.
  - Do that **every time** you access the buffer!!!

# Historical examples

Bring your own vulnerable driver 😊





# Generalities

- Writing a driver at **a professional level** is not a mass sport...
  - At least, good/reliable/efficient drivers 😊.
  - Most of the time, drivers are written in C.
    - Hiring a good C developer can be expensive.
    - Also, with a strong knowledge in hardware devices.
    - Also, with a strong knowledge in security.
    - Also, with a strong knowledge of the operating system.
    - Also, with (...)
  - Why?
    - Historical reasons and many samples are provided with this language.
    - C++ possibilities, also Rust but it is only for a few products, in practice.

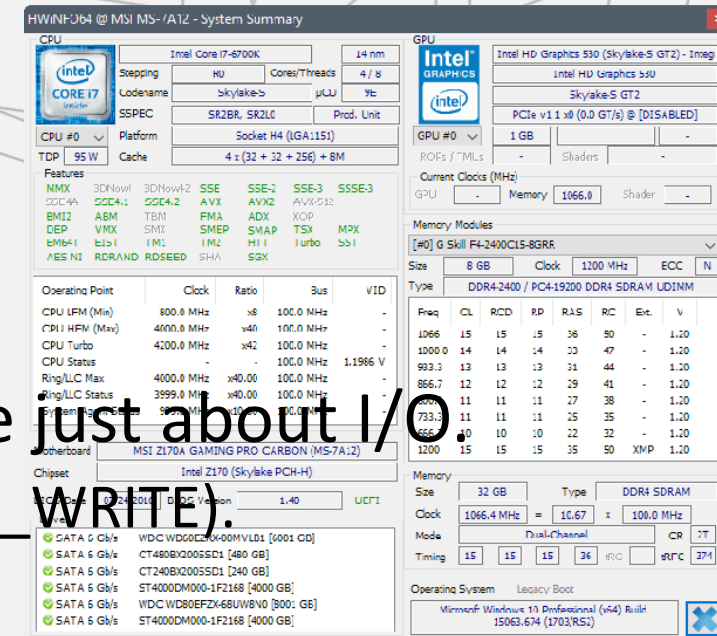
# Generalities

- A lot of devices manufacturers are not so focus about security.
  - But they like to develop drivers for their devices...
  - Sometimes for dubious reasons or fancy/useless purposes... 🙄
    - Is the colored LEDs blighting a “must have” feature? 🤔
  - Sometimes it can be useful ... especially supporting proprietary protocols.
- Some manufacturers like to hire driver software “providers”.
  - As in any industry, there are the good and the bad providers...
  - But some of them are somehow “creative”:
    - Unlimited super-power with copy & past 😊.



# Generalities

- A lot of “device driver” (or *related device drivers*) are just about I/O.
  - Managing I/O to the device, first (IRP\_MJ\_READ / IRP\_MJ\_WRITE).
  - But also providing specific commands to the device.
    - Change the color of keyboard’s keys ... ☹️
    - Overclock the GPUs of the graphic card.
    - Give me internal power information to display fancy stuff on screen.
- In fact, a lot of drivers are based on some “well-known” projects.
  - The “fabulous three” 😊:
    - Old Windows DDK samples [<https://winworldpc.com/product/windows-sdk-ddk/nt-40>]
    - WinRing0 by hiyohiyo & GermanAizek [<https://github.com/GermanAizek/WinRing0>]
    - WinIO by Yariv Kaplan [<https://github.com/starofrainnight/winio>]



Code coming from  
Windows 98, 3.5 ... and  
it is still useable!

# Windows SDK & DDK NT 4.0

The Microsoft Windows Software Development Kits (SDK) provide sample program code, extra libraries, and documentation to aid application developers producing Windows applications. Microsoft Windows Driver Development Kits are similar sets of samples and libraries but specific to device driver development, and much more in-depth.

Available releases

- Windows 1.x
- Windows 2.x
- Windows 3.0
- NT 3.x
- Windows 3.1
- WfW 3.11
- Windows 95
- NT 4.0
- XP (NT 5.1)
- 2003 (NT 5.2)

## Information

### Product type

DevTool

### Vendor

Microsoft

### Release date

1996

### User interface

GUI





### Platform

Windows

### Download count

5 (0 for release)

## Downloads

Download name	Version	Language	Architecture	File size	Downloads
 <a href="#">Microsoft Windows NT 4.0 DDK (4.0.1381)</a>	4.0.1381	English		31.12MB	0
 <a href="#">Microsoft Windows NT 4.0 SDK (1996)</a>	4.0	English		196.48MB	0

## Comments

Leave a comment

Comment As ...



SomeGuy  
May 12

Ok, fixed. Thanks.

# WinRing0

- WinRing0 is a hardware access library for Windows.
- WinRing0 library allows x86/x64 Windows applications to access
  - I/O port Direct device communication
  - PCI Direct device communication
  - MSR (Model-Specific Register) Major disclosure of information
    - Bypassing R3/R0 protection
    - ASLR bypass
    - Memory isolation
- Set of Driver / Dll to give a direct access to hardware resources.
  - “Useful” to control piece of hardware without all the kernel complexity.
  - Base support for hardware vendors to “directly drive” their device.

# WinIO

- The WinIo library allows 32-bit and 64-bit Windows applications to directly access I/O ports and physical memory.
- Among the different features proposed:
  - Direct I/O port access Direct device communication
  - Mapping of physical memory Windows DDK (1993) Read / Write What Where Bypassing R3/R0 protection R3/R0 isolation
- Why trying to understand the complexity of the kernel?
  - When there are some “project” doing the work “simply” ... 🙄
  - “Since it works” ...

```
/*++  
  
Copyright (c) 1993 Microsoft Corporation  
  
Module Name:  
  
    mapmem.c  
  
Abstract:  
  
    A simple driver sample which shows how to map physical memory  
    into a user-mode process's address space using the  
    ZwMapViewOfSection APIs.  
  
Environment:  
  
    kernel mode only  
  
Notes:  
  
    For the sake of simplicity this sample does not attempt to  
    recognize resource conflicts with other drivers/devices. A  
    real-world driver would call IoReportResource usage to  
    determine whether or not the resource is available, and if  
    so, register the resource under it's name.  
  
Revision History:  
  
--*/
```



# A second example

- Exploits about RTCore driver have been found by “hFireF0x”.
  - RTCore is a name of kernel mode driver used by MSI Afterburner software (<https://www.msi.com/page/afterburner>).
  - CVE-2019-16098 (published in 01/2020)

## MSI Afterburner

MSI Afterburner is the world's most recognized and widely used graphics card overclocking utility which gives you full control of your graphics cards.



Micro-Star International

<https://www.msi.com> › Graphics-Card

GeForce® GT 1030 AERO ITX 4GD4 OC - MSI



```
1 NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
2 {
3     NTSTATUS status; // eax
4     PDEVICE_OBJECT DeviceObject; // [rsp+40h] [rbp-38h] BYREF
5     struct _UNICODE_STRING DestinationString; // [rsp+48h] [rbp-30h] BYREF
6     struct _UNICODE_STRING SymbolicLinkName; // [rsp+58h] [rbp-20h] BYREF
7
8     RtlInitUnicodeString(&DestinationString, DeviceName); // \Device\RTCore64
9     RtlInitUnicodeString(&SymbolicLinkName, MsDosName); // '\DosDevices\RTCore64'
10    status = IoCreateDevice(DriverObject, 0, &DestinationString, 0x22u, 0, 0, &DeviceObject);
11    if ( status >= 0 )
12    {
13        status = IoCreateSymbolicLink(&SymbolicLinkName, &DestinationString);
14        if ( status >= 0 )
15        {
16            DriverObject->MajorFunction[IRP_MJ_CREATE] = IrpDispatcher;
17            DriverObject->MajorFunction[IRP_MJ_CLOSE] = IrpDispatcher;
18            DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IrpDispatcher;
19            DriverObject->DriverUnload = DriverUnload;
20            return 0;
21        }
22    }
23    return status;
24 }
```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```

25 // The driver is inherently insecure
26 // Let's reduce the attack surface by allowing only Administrators to access the driver
27 NTSTATUS status = IoCreateDeviceSecure(
28     DriverObject,                // Our Driver Object
29     0,                          // We don't use a device extension
30     &ntDeviceName,               // Device name
31     OLS_TYPE,                   // Device type
32     FILE_DEVICE_SECURE_OPEN,    // Device characteristics
33     FALSE,
34     &SDDL_DEVOBJ_SYS_ALL_ADM_ALL,
35     NULL,                       // Device class GUID
36     &deviceObject);             // Returned ptr to Device Object
37
38 if (!NT_SUCCESS(status))
39 {
40     refCount = (ULONG)(-1);
41     return status;
42 }
43 else
44 {
45     refCount = 0;
46 }
47
48 // Initialize the driver object with this driver's entry points.
49 DriverObject->MajorFunction[IRP_MJ_CREATE] = OlsDispatch;
50 DriverObject->MajorFunction[IRP_MJ_CLOSE] = OlsDispatch;
51 DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = OlsDispatch;
52 DriverObject->DriverUnload = Unload;

```

winio / Source / Drv / WinIo.c

Code Blame 478 lines (346 loc) · 12 KB

```

46 RtlInitUnicodeString (&DeviceNameUnicodeString, L"\\Device\\WinIo");
47
48 // Create a device object
49
50 ntStatus = IoCreateDevice (DriverObject,
51     0,
52     &DeviceNameUnicodeString,
53     FILE_DEVICE_WINIO,
54     0,
55     FALSE,
56     &DeviceObject);
57
58 if (NT_SUCCESS(ntStatus))
59 {
60     // Create dispatch points for device control, create, close.
61
62     DriverObject->MajorFunction[IRP_MJ_CREATE] =
63     DriverObject->MajorFunction[IRP_MJ_CLOSE] =
64     DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = WinIoDispatch;
65     DriverObject->DriverUnload = WinIoUnload;
66 }

```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```

25 // The driver is inherently insecure
26 // Let's reduce the attack surface by allowing only Administrators to access the driver
27 NTSTATUS status = IoCreateDeviceSecure(
28     DriverObject,                // Our Driver Object
29     0,                          // We don't use a device extension
30     &ntDeviceName,               // Device name
31     OLS_TYPE,                   // Device type
32     FILE_DEVICE_SECURE_OPEN,    // Device characteristics
33     FALSE,
34     &SDDL_DEVOBJ_SYS_ALL_ADM_ALL,
35     NULL,                       // Device class GUID
36     &deviceObject);
37
38 if (!NT_SUCCESS(status))
39 {
40     refCount = (ULONG)(-1);
41     return status;
42 }
43 else
44 {
45     refCount = 0;
46 }
47
48 // Initialize the driver object with this driver's entry points.
49 DriverObject->MajorFunction[IRP_MJ_CREATE] = OlsDispatch;
50 DriverObject->MajorFunction[IRP_MJ_CLOSE] = OlsDispatch;
51 DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = OlsDispatch;
52 DriverObject->DriverUnload = Unload;

```

SDDL\_DEVOBJ\_SYS\_ALL\_ADM\_ALL

"D:P(A;;GA;;;SY)(A;;GA;;;BA)"

SDDL\_DEVOBJ\_SYS\_ALL\_ADM\_ALL allows the kernel, system, and administrator complete control over the device. No other users may access the device.

device control, create, close.

winio / Source / Drv / WinIo.c

Code Blame 478 lines (346 loc) · 12 KB

```

46 RtlInitUnicodeString (&DeviceNameUnicodeString, L"\\Device\\WinIo");
47
48 // Create a device object
49
50 ntStatus = IoCreateDevice (DriverObject,
51     0,
52     &DeviceNameUnicodeString,
53     FILE_DEVICE_WINIO,

```

```

61
62 DriverObject->MajorFunction[IRP_MJ_CREATE] =
63 DriverObject->MajorFunction[IRP_MJ_CLOSE] =
64 DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = WinIoDispatch;
65 DriverObject->DriverUnload = WinIoUnload;
66

```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```

25 // The driver is inherently insecure
26 // Let's reduce the attack surface by allowing only Administrators to access the driver
27 NTSTATUS status = IoCreateDeviceSecure(
28     DriverObject,                // Our Driver Object
29     0,                          // We don't use a device extension
30     &ntDeviceName,               // Device name
31     OLS_TYPE,                   // Device type
32     FILE_DEVICE_SECURE_OPEN,    // Device characteristics
33     FALSE,
34     &SDDL_DEVOBJ_SYS_ALL_ADM_ALL,
35     NULL,                       // Device class GUID
36     &deviceObject);             // Returned ptr to Device Object
37
38 if (!NT_SUCCESS(status))
39 {
40     refCount = (ULONG)(-1);
41     return status;
42 }
43 else
44 {
45     refCount = 0;
46 }
47
48 // Initialize the driver object with this driver's entry points.
49 DriverObject->MajorFunction[IRP_MJ_CREATE] = OlsDispatch;
50 DriverObject->MajorFunction[IRP_MJ_CLOSE] = OlsDispatch;
51 DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = OlsDispatch;
52 DriverObject->DriverUnload = Unload;

```

winio / Source / Drv / WinIo.c

Code Blame 478 lines (346 loc) · 12 KB

```

46 RtlInitUnicodeString (&DeviceNameUnicodeString, L"\\Device\\WinIo");
47
48 // Create a device object
49
50 ntStatus = IoCreateDevice (DriverObject,
51     0,
52     &DeviceNameUnicodeString,
53     FILE_DEVICE_WINIO,
54     0,
55     FALSE,
56     &DeviceObject);
57
58 if (NT_SUCCESS(ntStatus))
59 {
60     // Create dispatch points for device control, create, close.
61
62     DriverObject->MajorFunction[IRP_MJ_CREATE] =
63     DriverObject->MajorFunction[IRP_MJ_CLOSE] =
64     DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = WinIoDispatch;
65     DriverObject->DriverUnload = WinIoUnload;
66 }

```





```
1 NTSTATUS __stdcall IrpDispatcher(struct _DEVICE_OBJECT *DeviceObject, struct _IRP *Irp)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     Irp->IoStatus.Status = 0;
6     Irp->IoStatus.Information = 0i64;
7     CurrentStackLocation = Irp->Tail.Overlay.CurrentStackLocation;
8     SystemBuffer = Irp->AssociatedIrp.SystemBuffer;
9     InputBufferLength = CurrentStackLocation->Parameters.DeviceIoControl.InputBufferLength;
10    OutputBufferLength = CurrentStackLocation->Parameters.DeviceIoControl.OutputBufferLength;
11    if ( CurrentStackLocation->MajorFunction == 14 )
12    {
13        switch ( CurrentStackLocation->Parameters.DeviceIoControl.IoControlCode )
14        {
15            case 0x80002000:
16                status = WinIo(
17                    DeviceObject,
18                    (PHYSICAL_ADDRESS *)SystemBuffer,
19                    (unsigned int)InputBufferLength,
20                    OutputBufferLength);
21                Irp->IoStatus.Status = status;
22                if ( status < 0 )
23                {
24                    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
25                }
26                else
27                {
28                    Irp->IoStatus.Information = 8i64;
29                    break;
30                }
31            case 0x80002004:
32                if ( (unsigned int)InputBufferLength < 8 )
33                {
34                    Irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
35                }
36                else
37                {
38                    Irp->IoStatus.Status = ZwUnmapViewOfSection((HANDLE)0xFFFFFFFFFFFFFFFFi64, *(PVOID *)SystemBuffer);
39                }
40                break;
41        }
42    }
43}
```

```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace, &TranslatedAddress);
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddress);
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
            ViewSize = SizeView;

```

```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     _int64 InputBufferLength,
5     _int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace, &TranslatedAddress);
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddress);
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;

```

```

//
// Our user mode app will pass an initialized structure like this
// down to the kernel mode driver
//

```

```

typedef struct
{
    INTERFACE_TYPE    InterfaceType; // Isa, Eisa, etc....
    ULONG              BusNumber;     // Bus number
    PHYSICAL_ADDRESS  BusAddress;     // Bus-relative address
    ULONG              AddressSpace;  // 0 is memory, 1 is I/O
    ULONG              Length;        // Length of section to map
} PHYSICAL_MEMORY_INFO, *PPHYSICAL_MEMORY_INFO;

```

```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     _int64 InputBufferLength,
5     _int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace, &TranslatedAddress);
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddress);
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;

```

```

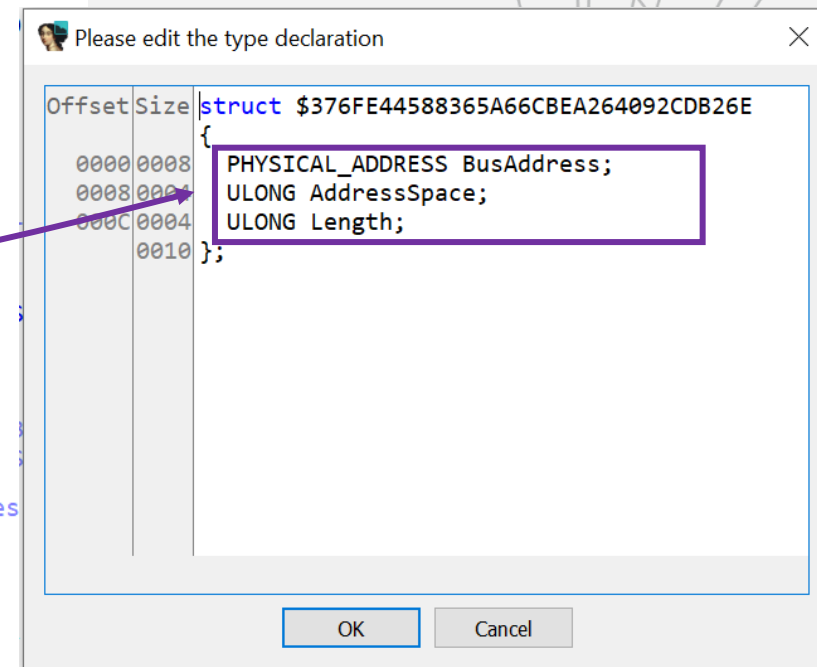
//
// Our user mode app will pass an initialized structure like this
// down to the kernel mode driver
//

```

```

typedef struct
{
    INTERFACE_TYPE    InterfaceType; // Isa, Eisa, etc....
    ULONG             BusNumber;      // Bus number
    PHYSICAL_ADDRESS  BusAddress;     // Bus-relative address
    ULONG             AddressSpace;   // 0 is memory, 1 is I/O
    ULONG             Length;         // Length of section to map
} PHYSICAL_MEMORY_INFO *PPHYSICAL_MEMORY_INFO;

```



```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace, &TranslatedAddress);
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddress);
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
            ViewSize = SizeView;

```



```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Obj
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Addre
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddr
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;

```

```

RtlInitUnicodeString (&physicalMemoryUnicodeString,
    L"\\Device\\PhysicalMemory");

InitializeObjectAttributes (&objectAttributes,
    &physicalMemoryUnicodeString,
    OBJ_CASE_INSENSITIVE,
    (HANDLE) NULL,
    (PSECURITY_DESCRIPTOR) NULL);

ntStatus = ZwOpenSection (&physicalMemoryHandle,
    SECTION_ALL_ACCESS,
    &objectAttributes);

if (!NT_SUCCESS(ntStatus))
{
    MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));

    goto done;
}

ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
    SECTION_ALL_ACCESS,
    (POBJECT_TYPE) NULL,
    KernelMode,
    &PhysicalMemorySection,
    (POBJECT_HANDLE_INFORMATION) NULL);

if (!NT_SUCCESS(ntStatus))
{
    MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle failed\n"));

    goto close_handle;
}

//
// Initialize the physical addresses that will be translated
//

physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,
    RtlConvertUlongToLargeInteger (
    length));

```

its reserved - baptiste



```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Obj
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Addre
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddr
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;
48
49            RtlInitUnicodeString (&physicalMemoryUnicodeString,
50                                 L"\\Device\\PhysicalMemory");
51
52            InitializeObjectAttributes (&objectAttributes,
53                                       &physicalMemoryUnicodeString,
54                                       OBJ_CASE_INSENSITIVE,
55                                       (HANDLE) NULL,
56                                       (PSECURITY_DESCRIPTOR) NULL);
57
58            ntStatus = ZwOpenSection (&physicalMemoryHandle,
59                                     SECTION_ALL_ACCESS,
60                                     &objectAttributes);
61
62            if (!NT_SUCCESS(ntStatus))
63            {
64                MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));
65
66                goto done;
67            }
68
69            ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
70                                                 SECTION_ALL_ACCESS,
71                                                 (POBJECT_TYPE) NULL,
72                                                 KernelMode,
73                                                 &PhysicalMemorySection,
74                                                 (POBJECT_HANDLE_INFORMATION) NULL);
75
76            if (!NT_SUCCESS(ntStatus))
77            {
78                MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle failed\n"));
79
80                goto close_handle;
81            }
82
83            //
84            // Initialize the physical addresses that will be translated
85            //
86
87            physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,
88                                                    RtlConvertUlongToLargeInteger (
89                                                        length));
90
91            //
92            //
93
94            //
95            //
96            //
97            //
98            //
99            //
100           //
101           //
102           //
103           //
104           //
105           //
106           //
107           //
108           //
109           //
110           //
111           //
112           //
113           //
114           //
115           //
116           //
117           //
118           //
119           //
120           //
121           //
122           //
123           //
124           //
125           //
126           //
127           //
128           //
129           //
130           //
131           //
132           //
133           //
134           //
135           //
136           //
137           //
138           //
139           //
140           //
141           //
142           //
143           //
144           //
145           //
146           //
147           //
148           //
149           //
150           //
151           //
152           //
153           //
154           //
155           //
156           //
157           //
158           //
159           //
160           //
161           //
162           //
163           //
164           //
165           //
166           //
167           //
168           //
169           //
170           //
171           //
172           //
173           //
174           //
175           //
176           //
177           //
178           //
179           //
180           //
181           //
182           //
183           //
184           //
185           //
186           //
187           //
188           //
189           //
190           //
191           //
192           //
193           //
194           //
195           //
196           //
197           //
198           //
199           //
200           //
201           //
202           //
203           //
204           //
205           //
206           //
207           //
208           //
209           //
210           //
211           //
212           //
213           //
214           //
215           //
216           //
217           //
218           //
219           //
220           //
221           //
222           //
223           //
224           //
225           //
226           //
227           //
228           //
229           //
230           //
231           //
232           //
233           //
234           //
235           //
236           //
237           //
238           //
239           //
240           //
241           //
242           //
243           //
244           //
245           //
246           //
247           //
248           //
249           //
250           //
251           //
252           //
253           //
254           //
255           //
256           //
257           //
258           //
259           //
260           //
261           //
262           //
263           //
264           //
265           //
266           //
267           //
268           //
269           //
270           //
271           //
272           //
273           //
274           //
275           //
276           //
277           //
278           //
279           //
280           //
281           //
282           //
283           //
284           //
285           //
286           //
287           //
288           //
289           //
290           //
291           //
292           //
293           //
294           //
295           //
296           //
297           //
298           //
299           //
300           //
301           //
302           //
303           //
304           //
305           //
306           //
307           //
308           //
309           //
310           //
311           //
312           //
313           //
314           //
315           //
316           //
317           //
318           //
319           //
320           //
321           //
322           //
323           //
324           //
325           //
326           //
327           //
328           //
329           //
330           //
331           //
332           //
333           //
334           //
335           //
336           //
337           //
338           //
339           //
340           //
341           //
342           //
343           //
344           //
345           //
346           //
347           //
348           //
349           //
350           //
351           //
352           //
353           //
354           //
355           //
356           //
357           //
358           //
359           //
360           //
361           //
362           //
363           //
364           //
365           //
366           //
367           //
368           //
369           //
370           //
371           //
372           //
373           //
374           //
375           //
376           //
377           //
378           //
379           //
380           //
381           //
382           //
383           //
384           //
385           //
386           //
387           //
388           //
389           //
390           //
391           //
392           //
393           //
394           //
395           //
396           //
397           //
398           //
399           //
400           //
401           //
402           //
403           //
404           //
405           //
406           //
407           //
408           //
409           //
410           //
411           //
412           //
413           //
414           //
415           //
416           //
417           //
418           //
419           //
420           //
421           //
422           //
423           //
424           //
425           //
426           //
427           //
428           //
429           //
430           //
431           //
432           //
433           //
434           //
435           //
436           //
437           //
438           //
439           //
440           //
441           //
442           //
443           //
444           //
445           //
446           //
447           //
448           //
449           //
450           //
451           //
452           //
453           //
454           //
455           //
456           //
457           //
458           //
459           //
460           //
461           //
462           //
463           //
464           //
465           //
466           //
467           //
468           //
469           //
470           //
471           //
472           //
473           //
474           //
475           //
476           //
477           //
478           //
479           //
480           //
481           //
482           //
483           //
484           //
485           //
486           //
487           //
488           //
489           //
490           //
491           //
492           //
493           //
494           //
495           //
496           //
497           //
498           //
499           //
500           //
501           //
502           //
503           //
504           //
505           //
506           //
507           //
508           //
509           //
510           //
511           //
512           //
513           //
514           //
515           //
516           //
517           //
518           //
519           //
520           //
521           //
522           //
523           //
524           //
525           //
526           //
527           //
528           //
529           //
530           //
531           //
532           //
533           //
534           //
535           //
536           //
537           //
538           //
539           //
540           //
541           //
542           //
543           //
544           //
545           //
546           //
547           //
548           //
549           //
550           //
551           //
552           //
553           //
554           //
555           //
556           //
557           //
558           //
559           //
560           //
561           //
562           //
563           //
564           //
565           //
566           //
567           //
568           //
569           //
570           //
571           //
572           //
573           //
574           //
575           //
576           //
577           //
578           //
579           //
580           //
581           //
582           //
583           //
584           //
585           //
586           //
587           //
588           //
589           //
590           //
591           //
592           //
593           //
594           //
595           //
596           //
597           //
598           //
599           //
600           //
601           //
602           //
603           //
604           //
605           //
606           //
607           //
608           //
609           //
610           //
611           //
612           //
613           //
614           //
615           //
616           //
617           //
618           //
619           //
620           //
621           //
622           //
623           //
624           //
625           //
626           //
627           //
628           //
629           //
630           //
631           //
632           //
633           //
634           //
635           //
636           //
637           //
638           //
639           //
640           //
641           //
642           //
643           //
644           //
645           //
646           //
647           //
648           //
649           //
650           //
651           //
652           //
653           //
654           //
655           //
656           //
657           //
658           //
659           //
660           //
661           //
662           //
663           //
664           //
665           //
666           //
667           //
668           //
669           //
670           //
671           //
672           //
673           //
674           //
675           //
676           //
677           //
678           //
679           //
680           //
681           //
682           //
683           //
684           //
685           //
686           //
687           //
688           //
689           //
690           //
691           //
692           //
693           //
694           //
695           //
696           //
697           //
698           //
699           //
700           //
701           //
702           //
703           //
704           //
705           //
706           //
707           //
708           //
709           //
710           //
711           //
712           //
713           //
714           //
715           //
716           //
717           //
718           //
719           //
720           //
721           //
722           //
723           //
724           //
725           //
726           //
727           //
728           //
729           //
730           //
731           //
732           //
733           //
734           //
735           //
736           //
737           //
738           //
739           //
740           //
741           //
742           //
743           //
744           //
745           //
746           //
747           //
748           //
749           //
750           //
751           //
752           //
753           //
754           //
755           //
756           //
757           //
758           //
759           //
760           //
761           //
762           //
763           //
764           //
765           //
766           //
767           //
768           //
769           //
770           //
771           //
772           //
773           //
774           //
775           //
776           //
777           //
778           //
779           //
780           //
781           //
782           //
783           //
784           //
785           //
786           //
787           //
788           //
789           //
790           //
791           //
792           //
793           //
794           //
795           //
796           //
797           //
798           //
799           //
800           //
801           //
802           //
803           //
804           //
805           //
806           //
807           //
808           //
809           //
810           //
811           //
812           //
813           //
814           //
815           //
816           //
817           //
818           //
819           //
820           //
821           //
822           //
823           //
824           //
825           //
826           //
827           //
828           //
829           //
830           //
831           //
832           //
833           //
834           //
835           //
836           //
837           //
838           //
839           //
840           //
841           //
842           //
843           //
844           //
845           //
846           //
847           //
848           //
849           //
850           //
851           //
852           //
853           //
854           //
855           //
856           //
857           //
858           //
859           //
860           //
861           //
862           //
863           //
864           //
865           //
866           //
867           //
868           //
869           //
870           //
871           //
872           //
873           //
874           //
875           //
876           //
877           //
878           //
879           //
880           //
881           //
882           //
883           //
884           //
885           //
886           //
887           //
888           //
889           //
890           //
891           //
892           //
893           //
894           //
895           //
896           //
897           //
898           //
899           //
900           //
901           //
902           //
903           //
904           //
905           //
906           //
907           //
908           //
909           //
910           //
911           //
912           //
913           //
914           //
915           //
916           //
917           //
918           //
919           //
920           //
921           //
922           //
923           //
924           //
925           //
926           //
927           //
928           //
929           //
930           //
931           //
932           //
933           //
934           //
935           //
936           //
937           //
938           //
939           //
940           //
941           //
942           //
943           //
944           //
945           //
946           //
947           //
948           //
949           //
950           //
951           //
952           //
953           //
954           //
955           //
956           //
957           //
958           //
959           //
960           //
961           //
962           //
963           //
964           //
965           //
966           //
967           //
968           //
969           //
970           //
971           //
972           //
973           //
974           //
975           //
976           //
977           //
978           //
979           //
980           //
981           //
982           //
983           //
984           //
985           //
986           //
987           //
988           //
989           //
990           //
991           //
992           //
993           //
994           //
995           //
996           //
997           //
998           //
999           //
1000          //

```

```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Obj
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Addre
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddr
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;
48
49            RtlInitUnicodeString (&physicalMemoryUnicodeString,
50                                 L"\\Device\\PhysicalMemory");
51
52            InitializeObjectAttributes (&objectAttributes,
53                                       &physicalMemoryUnicodeString,
54                                       OBJ_CASE_INSENSITIVE,
55                                       (HANDLE) NULL,
56                                       (PSECURITY_DESCRIPTOR) NULL);
57
58            ntStatus = ZwOpenSection (&physicalMemoryHandle,
59                                     SECTION_ALL_ACCESS,
60                                     &objectAttributes);
61
62            if (!NT_SUCCESS(ntStatus))
63            {
64                MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));
65
66                goto done;
67            }
68
69            ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
70                                                 SECTION_ALL_ACCESS,
71                                                 (POBJECT_TYPE) NULL,
72                                                 KernelMode,
73                                                 &PhysicalMemorySection,
74                                                 (POBJECT_HANDLE_INFORMATION) NULL);
75
76            if (!NT_SUCCESS(ntStatus))
77            {
78                MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle failed\n"));
79
80                goto close_handle;
81            }
82
83            //
84            // Initialize the physical addresses that will be translated
85            //
86
87            physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,
88                                                    RtlConvertUlongToLargeInteger (
89                                                        length));
90
91            //
92            //
93
94            //
95            //
96
97            //
98            //
99
100           //
101           //
102
103           //
104           //
105
106           //
107           //
108
109           //
110           //
111
112           //
113           //
114
115           //
116           //
117
118           //
119           //
120
121           //
122           //
123
124           //
125           //
126
127           //
128           //
129
130           //
131           //
132
133           //
134           //
135
136           //
137           //
138
139           //
140           //
141
142           //
143           //
144
145           //
146           //
147
148           //
149           //
150
151           //
152           //
153
154           //
155           //
156
157           //
158           //
159
160           //
161           //
162
163           //
164           //
165
166           //
167           //
168
169           //
170           //
171
172           //
173           //
174
175           //
176           //
177
178           //
179           //
180
181           //
182           //
183
184           //
185           //
186
187           //
188           //
189
190           //
191           //
192
193           //
194           //
195
196           //
197           //
198
199           //
200           //
201
202           //
203           //
204
205           //
206           //
207
208           //
209           //
210
211           //
212           //
213
214           //
215           //
216
217           //
218           //
219
220           //
221           //
222
223           //
224           //
225
226           //
227           //
228
229           //
230           //
231
232           //
233           //
234
235           //
236           //
237
238           //
239           //
240
241           //
242           //
243
244           //
245           //
246
247           //
248           //
249
250           //
251           //
252
253           //
254           //
255
256           //
257           //
258
259           //
260           //
261
262           //
263           //
264
265           //
266           //
267
268           //
269           //
270
271           //
272           //
273
274           //
275           //
276
277           //
278           //
279
280           //
281           //
282
283           //
284           //
285
286           //
287           //
288
289           //
290           //
291
292           //
293           //
294
295           //
296           //
297
298           //
299           //
300
301           //
302           //
303
304           //
305           //
306
307           //
308           //
309
310           //
311           //
312
313           //
314           //
315
316           //
317           //
318
319           //
320           //
321
322           //
323           //
324
325           //
326           //
327
328           //
329           //
330
331           //
332           //
333
334           //
335           //
336
337           //
338           //
339
340           //
341           //
342
343           //
344           //
345
346           //
347           //
348
349           //
350           //
351
352           //
353           //
354
355           //
356           //
357
358           //
359           //
360
361           //
362           //
363
364           //
365           //
366
367           //
368           //
369
370           //
371           //
372
373           //
374           //
375
376           //
377           //
378
379           //
380           //
381
382           //
383           //
384
385           //
386           //
387
388           //
389           //
390
391           //
392           //
393
394           //
395           //
396
397           //
398           //
399
400           //
401           //
402
403           //
404           //
405
406           //
407           //
408
409           //
410           //
411
412           //
413           //
414
415           //
416           //
417
418           //
419           //
420
421           //
422           //
423
424           //
425           //
426
427           //
428           //
429
430           //
431           //
432
433           //
434           //
435
436           //
437           //
438
439           //
440           //
441
442           //
443           //
444
445           //
446           //
447
448           //
449           //
450
451           //
452           //
453
454           //
455           //
456
457           //
458           //
459
460           //
461           //
462
463           //
464           //
465
466           //
467           //
468
469           //
470           //
471
472           //
473           //
474
475           //
476           //
477
478           //
479           //
480
481           //
482           //
483
484           //
485           //
486
487           //
488           //
489
490           //
491           //
492
493           //
494           //
495
496           //
497           //
498
499           //
500           //
501
502           //
503           //
504
505           //
506           //
507
508           //
509           //
510
511           //
512           //
513
514           //
515           //
516
517           //
518           //
519
520           //
521           //
522
523           //
524           //
525
526           //
527           //
528
529           //
530           //
531
532           //
533           //
534
535           //
536           //
537
538           //
539           //
540
541           //
542           //
543
544           //
545           //
546
547           //
548           //
549
550           //
551           //
552
553           //
554           //
555
556           //
557           //
558
559           //
560           //
561
562           //
563           //
564
565           //
566           //
567
568           //
569           //
570
571           //
572           //
573
574           //
575           //
576
577           //
578           //
579
580           //
581           //
582
583           //
584           //
585
586           //
587           //
588
589           //
590           //
591
592           //
593           //
594
595           //
596           //
597
598           //
599           //
600
601           //
602           //
603
604           //
605           //
606
607           //
608           //
609
610           //
611           //
612
613           //
614           //
615
616           //
617           //
618
619           //
620           //
621
622           //
623           //
624
625           //
626           //
627
628           //
629           //
630
631           //
632           //
633
634           //
635           //
636
637           //
638           //
639
640           //
641           //
642
643           //
644           //
645
646           //
647           //
648
649           //
650           //
651
652           //
653           //
654
655           //
656           //
657
658           //
659           //
660
661           //
662           //
663
664           //
665           //
666
667           //
668           //
669
670           //
671           //
672
673           //
674           //
675
676           //
677           //
678
679           //
680           //
681
682           //
683           //
684
685           //
686           //
687
688           //
689           //
690
691           //
692           //
693
694           //
695           //
696
697           //
698           //
699
700           //
701           //
702
703           //
704           //
705
706           //
707           //
708
709           //
710           //
711
712           //
713           //
714
715           //
716           //
717
718           //
719           //
720
721           //
722           //
723
724           //
725           //
726
727           //
728           //
729
730           //
731           //
732
733           //
734           //
735
736           //
737           //
738
739           //
740           //
741
742           //
743           //
744
745           //
746           //
747
748           //
749           //
750
751           //
752           //
753
754           //
755           //
756
757           //
758           //
759
760           //
761           //
762
763           //
764           //
765
766           //
767           //
768
769           //
770           //
771
772           //
773           //
774
775           //
776           //
777
778           //
779           //
780
781           //
782           //
783
784           //
785           //
786
787           //
788           //
789
790           //
791           //
792
793           //
794           //
795
796           //
797           //
798
799           //
800           //
801
802           //
803           //
804
805           //
806           //
807
808           //
809           //
810
811           //
812           //
813
814           //
815           //
816
817           //
818           //
819
820           //
821           //
822
823           //
824           //
825
826           //
827           //
828
829           //
830           //
831
832           //
833           //
834
835           //
836           //
837
838           //
839           //
840
841           //
842           //
843
844           //
845           //
846
847           //
848           //
849
850           //
851           //
852
853           //
854           //
855
856           //
857           //
858
859           //
860           //
861
862           //
863           //
864
865           //
866           //
867
868           //
869           //
870
871           //
872           //
873
874           //
875           //
876
877           //
878           //
879
880           //
881           //
882
883           //
884           //
885
886           //
887           //
888
889           //
890           //
891
892           //
893           //
894
895           //
896           //
897
898           //
899           //
900
901           //
902           //
903
904           //
905           //
906
907           //
908           //
909
910           //
911           //
912
913           //
914           //
915
916           //
917           //
918
919           //
920           //
921
922           //
923           //
924
925           //
926           //
927
928           //
929           //
930
931           //
932           //
933
934           //
935           //
936
937           //
938           //
939
940           //
941           //
942
943           //
944           //
945
946           //
947           //
948
949           //
950           //
951
952           //
953           //
954
955           //
956           //
957
958           //
959           //
960
961           //
962           //
963
964           //
965           //
966
967           //
968           //
969
970           //
971           //
972
973           //
974           //
975
976           //
977           //
978
979           //
980           //
981
982           //
983           //
984
985           //
986           //
987
988           //
989           //
990
991           //
992           //
993
994           //
995           //
996
997           //
998           //
999
1000          //

```

```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Obj
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Addre
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddr
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;
48
49            RtlInitUnicodeString (&physicalMemoryUnicodeString,
50                                 L"\\Device\\PhysicalMemory");
51
52            InitializeObjectAttributes (&objectAttributes,
53                                       &physicalMemoryUnicodeString,
54                                       OBJ_CASE_INSENSITIVE,
55                                       (HANDLE) NULL,
56                                       (PSECURITY_DESCRIPTOR) NULL);
57
58            ntStatus = ZwOpenSection (&physicalMemoryHandle,
59                                    SECTION_ALL_ACCESS,
60                                    &objectAttributes);
61
62            if (!NT_SUCCESS(ntStatus))
63            {
64                MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));
65
66                goto done;
67            }
68
69            ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
70                                                SECTION_ALL_ACCESS,
71                                                (POBJECT_TYPE) NULL,
72                                                KernelMode,
73                                                &PhysicalMemorySection,
74                                                (POBJECT_HANDLE_INFORMATION) NULL);
75
76            if (!NT_SUCCESS(ntStatus))
77            {
78                MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle failed\n"));
79
80                goto close_handle;
81            }
82
83            //
84            // Initialize the physical addresses that will be translated
85            //
86
87            physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,
88                                                    RtlConvertUlongToLargeInteger (
89                                                        length));
90
91            //
92            //
93            //
94            //
95            //
96            //
97            //
98            //
99            //
100           //
101           //
102           //
103           //
104           //
105           //
106           //
107           //
108           //
109           //
110           //
111           //
112           //
113           //
114           //
115           //
116           //
117           //
118           //
119           //
120           //
121           //
122           //
123           //
124           //
125           //
126           //
127           //
128           //
129           //
130           //
131           //
132           //
133           //
134           //
135           //
136           //
137           //
138           //
139           //
140           //
141           //
142           //
143           //
144           //
145           //
146           //
147           //
148           //
149           //
150           //
151           //
152           //
153           //
154           //
155           //
156           //
157           //
158           //
159           //
160           //
161           //
162           //
163           //
164           //
165           //
166           //
167           //
168           //
169           //
170           //
171           //
172           //
173           //
174           //
175           //
176           //
177           //
178           //
179           //
180           //
181           //
182           //
183           //
184           //
185           //
186           //
187           //
188           //
189           //
190           //
191           //
192           //
193           //
194           //
195           //
196           //
197           //
198           //
199           //
200           //
201           //
202           //
203           //
204           //
205           //
206           //
207           //
208           //
209           //
210           //
211           //
212           //
213           //
214           //
215           //
216           //
217           //
218           //
219           //
220           //
221           //
222           //
223           //
224           //
225           //
226           //
227           //
228           //
229           //
230           //
231           //
232           //
233           //
234           //
235           //
236           //
237           //
238           //
239           //
240           //
241           //
242           //
243           //
244           //
245           //
246           //
247           //
248           //
249           //
250           //
251           //
252           //
253           //
254           //
255           //
256           //
257           //
258           //
259           //
260           //
261           //
262           //
263           //
264           //
265           //
266           //
267           //
268           //
269           //
270           //
271           //
272           //
273           //
274           //
275           //
276           //
277           //
278           //
279           //
280           //
281           //
282           //
283           //
284           //
285           //
286           //
287           //
288           //
289           //
290           //
291           //
292           //
293           //
294           //
295           //
296           //
297           //
298           //
299           //
300           //
301           //
302           //
303           //
304           //
305           //
306           //
307           //
308           //
309           //
310           //
311           //
312           //
313           //
314           //
315           //
316           //
317           //
318           //
319           //
320           //
321           //
322           //
323           //
324           //
325           //
326           //
327           //
328           //
329           //
330           //
331           //
332           //
333           //
334           //
335           //
336           //
337           //
338           //
339           //
340           //
341           //
342           //
343           //
344           //
345           //
346           //
347           //
348           //
349           //
350           //
351           //
352           //
353           //
354           //
355           //
356           //
357           //
358           //
359           //
360           //
361           //
362           //
363           //
364           //
365           //
366           //
367           //
368           //
369           //
370           //
371           //
372           //
373           //
374           //
375           //
376           //
377           //
378           //
379           //
380           //
381           //
382           //
383           //
384           //
385           //
386           //
387           //
388           //
389           //
390           //
391           //
392           //
393           //
394           //
395           //
396           //
397           //
398           //
399           //
400           //
401           //
402           //
403           //
404           //
405           //
406           //
407           //
408           //
409           //
410           //
411           //
412           //
413           //
414           //
415           //
416           //
417           //
418           //
419           //
420           //
421           //
422           //
423           //
424           //
425           //
426           //
427           //
428           //
429           //
430           //
431           //
432           //
433           //
434           //
435           //
436           //
437           //
438           //
439           //
440           //
441           //
442           //
443           //
444           //
445           //
446           //
447           //
448           //
449           //
450           //
451           //
452           //
453           //
454           //
455           //
456           //
457           //
458           //
459           //
460           //
461           //
462           //
463           //
464           //
465           //
466           //
467           //
468           //
469           //
470           //
471           //
472           //
473           //
474           //
475           //
476           //
477           //
478           //
479           //
480           //
481           //
482           //
483           //
484           //
485           //
486           //
487           //
488           //
489           //
490           //
491           //
492           //
493           //
494           //
495           //
496           //
497           //
498           //
499           //
500           //
501           //
502           //
503           //
504           //
505           //
506           //
507           //
508           //
509           //
510           //
511           //
512           //
513           //
514           //
515           //
516           //
517           //
518           //
519           //
520           //
521           //
522           //
523           //
524           //
525           //
526           //
527           //
528           //
529           //
530           //
531           //
532           //
533           //
534           //
535           //
536           //
537           //
538           //
539           //
540           //
541           //
542           //
543           //
544           //
545           //
546           //
547           //
548           //
549           //
550           //
551           //
552           //
553           //
554           //
555           //
556           //
557           //
558           //
559           //
560           //
561           //
562           //
563           //
564           //
565           //
566           //
567           //
568           //
569           //
570           //
571           //
572           //
573           //
574           //
575           //
576           //
577           //
578           //
579           //
580           //
581           //
582           //
583           //
584           //
585           //
586           //
587           //
588           //
589           //
590           //
591           //
592           //
593           //
594           //
595           //
596           //
597           //
598           //
599           //
600           //
601           //
602           //
603           //
604           //
605           //
606           //
607           //
608           //
609           //
610           //
611           //
612           //
613           //
614           //
615           //
616           //
617           //
618           //
619           //
620           //
621           //
622           //
623           //
624           //
625           //
626           //
627           //
628           //
629           //
630           //
631           //
632           //
633           //
634           //
635           //
636           //
637           //
638           //
639           //
640           //
641           //
642           //
643           //
644           //
645           //
646           //
647           //
648           //
649           //
650           //
651           //
652           //
653           //
654           //
655           //
656           //
657           //
658           //
659           //
660           //
661           //
662           //
663           //
664           //
665           //
666           //
667           //
668           //
669           //
670           //
671           //
672           //
673           //
674           //
675           //
676           //
677           //
678           //
679           //
680           //
681           //
682           //
683           //
684           //
685           //
686           //
687           //
688           //
689           //
690           //
691           //
692           //
693           //
694           //
695           //
696           //
697           //
698           //
699           //
700           //
701           //
702           //
703           //
704           //
705           //
706           //
707           //
708           //
709           //
710           //
711           //
712           //
713           //
714           //
715           //
716           //
717           //
718           //
719           //
720           //
721           //
722           //
723           //
724           //
725           //
726           //
727           //
728           //
729           //
730           //
731           //
732           //
733           //
734           //
735           //
736           //
737           //
738           //
739           //
740           //
741           //
742           //
743           //
744           //
745           //
746           //
747           //
748           //
749           //
750           //
751           //
752           //
753           //
754           //
755           //
756           //
757           //
758           //
759           //
760           //
761           //
762           //
763           //
764           //
765           //
766           //
767           //
768           //
769           //
770           //
771           //
772           //
773           //
774           //
775           //
776           //
777           //
778           //
779           //
780           //
781           //
782           //
783           //
784           //
785           //
786           //
787           //
788           //
789           //
790           //
791           //
792           //
793           //
794           //
795           //
796           //
797           //
798           //
799           //
800           //
801           //
802           //
803           //
804           //
805           //
806           //
807           //
808           //
809           //
810           //
811           //
812           //
813           //
814           //
815           //
816           //
817           //
818           //
819           //
820           //
821           //
822           //
823           //
824           //
825           //
826           //
827           //
828           //
829           //
830           //
831           //
832           //
833           //
834           //
835           //
836           //
837           //
838           //
839           //
840           //
841           //
842           //
843           //
844           //
845           //
846           //
847           //
848           //
849           //
850           //
851           //
852           //
853           //
854           //
855           //
856           //
857           //
858           //
859           //
860           //
861           //
862           //
863           //
864           //
865           //
866           //
867           //
868           //
869           //
870           //
871           //
872           //
873           //
874           //
875           //
876           //
877           //
878           //
879           //
880           //
881           //
882           //
883           //
884           //
885           //
886           //
887           //
888           //
889           //
890           //
891           //
892           //
893           //
894           //
895           //
896           //
897           //
898           //
899           //
900           //
901           //
902           //
903           //
904           //
905           //
906           //
907           //
908           //
909           //
910           //
911           //
912           //
913           //
914           //
915           //
916           //
917           //
918           //
919           //
920           //
921           //
922           //
923           //
924           //
925           //
926           //
927           //
928           //
929           //
930           //
931           //
932           //
933           //
934           //
935           //
936           //
937           //
938           //
939           //
940           //
941           //
942           //
943           //
944           //
945           //
946           //
947           //
948           //
949           //
950           //
951           //
952           //
953           //
954           //
955           //
956           //
957           //
958           //
959           //
960           //
961           //
962           //
963           //
964           //
965           //
966           //
967           //
968           //
969           //
970           //
971           //
972           //
973           //
974           //
975           //
976           //
977           //
978           //
979           //
980           //
981           //
982           //
983           //
984           //
985           //
986           //
987           //
988           //
989           //
990           //
991           //
992           //
993           //
994           //
995           //
996           //
997           //
998           //
999           //
1000          //

```

```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object);
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace_2, &BusAddress);
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddress);
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;
48        }
49    }
50    RtlInitUnicodeString (&physicalMemoryUnicodeString,
51                           L"\\Device\\PhysicalMemory");
52    InitializeObjectAttributes (&objectAttributes,
53                                &physicalMemoryUnicodeString,
54                                OBJ_CASE_INSENSITIVE,
55                                (HANDLE) NULL,
56                                (PSECURITY_DESCRIPTOR) NULL);
57    ntStatus = ZwOpenSection (&physicalMemoryHandle,
58                              SECTION_ALL_ACCESS,
59                              &objectAttributes);
60    if (!NT_SUCCESS(ntStatus))
61    {
62        MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));
63        goto done;
64    }
65    ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
66                                          SECTION_ALL_ACCESS,
67                                          (POBJECT_TYPE) NULL,
68                                          KernelMode,
69                                          &PhysicalMemorySection,
70                                          (POBJECT_HANDLE_INFORMATION) NULL);
71    if (!NT_SUCCESS(ntStatus))
72    {
73        MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle failed\n"));
74        goto close_handle;
75    }
76    //
77    // Initialize the physical addresses that will be translated
78    //
79    physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,
80                                              RtlConvertUlongToLargeInteger (length));
81    //
82    //
83    //
84    //
85    //
86    //
87    //
88    //
89    //
90    //
91    //
92    //
93    //
94    //
95    //
96    //
97    //
98    //
99    //
100   //
101   //
102   //
103   //
104   //
105   //
106   //
107   //
108   //
109   //
110   //
111   //
112   //
113   //
114   //
115   //
116   //
117   //
118   //
119   //
120   //
121   //
122   //
123   //
124   //
125   //
126   //
127   //
128   //
129   //
130   //
131   //
132   //
133   //
134   //
135   //
136   //
137   //
138   //
139   //
140   //
141   //
142   //
143   //
144   //
145   //
146   //
147   //
148   //
149   //
150   //
151   //
152   //
153   //
154   //
155   //
156   //
157   //
158   //
159   //
160   //
161   //
162   //
163   //
164   //
165   //
166   //
167   //
168   //
169   //
170   //
171   //
172   //
173   //
174   //
175   //
176   //
177   //
178   //
179   //
180   //
181   //
182   //
183   //
184   //
185   //
186   //
187   //
188   //
189   //
190   //
191   //
192   //
193   //
194   //
195   //
196   //
197   //
198   //
199   //
200   //
201   //
202   //
203   //
204   //
205   //
206   //
207   //
208   //
209   //
210   //
211   //
212   //
213   //
214   //
215   //
216   //
217   //
218   //
219   //
220   //
221   //
222   //
223   //
224   //
225   //
226   //
227   //
228   //
229   //
230   //
231   //
232   //
233   //
234   //
235   //
236   //
237   //
238   //
239   //
240   //
241   //
242   //
243   //
244   //
245   //
246   //
247   //
248   //
249   //
250   //
251   //
252   //
253   //
254   //
255   //
256   //
257   //
258   //
259   //
260   //
261   //
262   //
263   //
264   //
265   //
266   //
267   //
268   //
269   //
270   //
271   //
272   //
273   //
274   //
275   //
276   //
277   //
278   //
279   //
280   //
281   //
282   //
283   //
284   //
285   //
286   //
287   //
288   //
289   //
290   //
291   //
292   //
293   //
294   //
295   //
296   //
297   //
298   //
299   //
300   //
301   //
302   //
303   //
304   //
305   //
306   //
307   //
308   //
309   //
310   //
311   //
312   //
313   //
314   //
315   //
316   //
317   //
318   //
319   //
320   //
321   //
322   //
323   //
324   //
325   //
326   //
327   //
328   //
329   //
330   //
331   //
332   //
333   //
334   //
335   //
336   //
337   //
338   //
339   //
340   //
341   //
342   //
343   //
344   //
345   //
346   //
347   //
348   //
349   //
350   //
351   //
352   //
353   //
354   //
355   //
356   //
357   //
358   //
359   //
360   //
361   //
362   //
363   //
364   //
365   //
366   //
367   //
368   //
369   //
370   //
371   //
372   //
373   //
374   //
375   //
376   //
377   //
378   //
379   //
380   //
381   //
382   //
383   //
384   //
385   //
386   //
387   //
388   //
389   //
390   //
391   //
392   //
393   //
394   //
395   //
396   //
397   //
398   //
399   //
400   //
401   //
402   //
403   //
404   //
405   //
406   //
407   //
408   //
409   //
410   //
411   //
412   //
413   //
414   //
415   //
416   //
417   //
418   //
419   //
420   //
421   //
422   //
423   //
424   //
425   //
426   //
427   //
428   //
429   //
430   //
431   //
432   //
433   //
434   //
435   //
436   //
437   //
438   //
439   //
440   //
441   //
442   //
443   //
444   //
445   //
446   //
447   //
448   //
449   //
450   //
451   //
452   //
453   //
454   //
455   //
456   //
457   //
458   //
459   //
460   //
461   //
462   //
463   //
464   //
465   //
466   //
467   //
468   //
469   //
470   //
471   //
472   //
473   //
474   //
475   //
476   //
477   //
478   //
479   //
480   //
481   //
482   //
483   //
484   //
485   //
486   //
487   //
488   //
489   //
490   //
491   //
492   //
493   //
494   //
495   //
496   //
497   //
498   //
499   //
500   //
501   //
502   //
503   //
504   //
505   //
506   //
507   //
508   //
509   //
510   //
511   //
512   //
513   //
514   //
515   //
516   //
517   //
518   //
519   //
520   //
521   //
522   //
523   //
524   //
525   //
526   //
527   //
528   //
529   //
530   //
531   //
532   //
533   //
534   //
535   //
536   //
537   //
538   //
539   //
540   //
541   //
542   //
543   //
544   //
545   //
546   //
547   //
548   //
549   //
550   //
551   //
552   //
553   //
554   //
555   //
556   //
557   //
558   //
559   //
560   //
561   //
562   //
563   //
564   //
565   //
566   //
567   //
568   //
569   //
570   //
571   //
572   //
573   //
574   //
575   //
576   //
577   //
578   //
579   //
580   //
581   //
582   //
583   //
584   //
585   //
586   //
587   //
588   //
589   //
590   //
591   //
592   //
593   //
594   //
595   //
596   //
597   //
598   //
599   //
600   //
601   //
602   //
603   //
604   //
605   //
606   //
607   //
608   //
609   //
610   //
611   //
612   //
613   //
614   //
615   //
616   //
617   //
618   //
619   //
620   //
621   //
622   //
623   //
624   //
625   //
626   //
627   //
628   //
629   //
630   //
631   //
632   //
633   //
634   //
635   //
636   //
637   //
638   //
639   //
640   //
641   //
642   //
643   //
644   //
645   //
646   //
647   //
648   //
649   //
650   //
651   //
652   //
653   //
654   //
655   //
656   //
657   //
658   //
659   //
660   //
661   //
662   //
663   //
664   //
665   //
666   //
667   //
668   //
669   //
670   //
671   //
672   //
673   //
674   //
675   //
676   //
677   //
678   //
679   //
680   //
681   //
682   //
683   //
684   //
685   //
686   //
687   //
688   //
689   //
690   //
691   //
692   //
693   //
694   //
695   //
696   //
697   //
698   //
699   //
700   //
701   //
702   //
703   //
704   //
705   //
706   //
707   //
708   //
709   //
710   //
711   //
712   //
713   //
714   //
715   //
716   //
717   //
718   //
719   //
720   //
721   //
722   //
723   //
724   //
725   //
726   //
727   //
728   //
729   //
730   //
731   //
732   //
733   //
734   //
735   //
736   //
737   //
738   //
739   //
740   //
741   //
742   //
743   //
744   //
745   //
746   //
747   //
748   //
749   //
750   //
751   //
752   //
753   //
754   //
755   //
756   //
757   //
758   //
759   //
760   //
761   //
762   //
763   //
764   //
765   //
766   //
767   //
768   //
769   //
770   //
771   //
772   //
773   //
774   //
775   //
776   //
777   //
778   //
779   //
780   //
781   //
782   //
783   //
784   //
785   //
786   //
787   //
788   //
789   //
790   //
791   //
792   //
793   //
794   //
795   //
796   //
797   //
798   //
799   //
800   //
801   //
802   //
803   //
804   //
805   //
806   //
807   //
808   //
809   //
810   //
811   //
812   //
813   //
814   //
815   //
816   //
817   //
818   //
819   //
820   //
821   //
822   //
823   //
824   //
825   //
826   //
827   //
828   //
829   //
830   //
831   //
832   //
833   //
834   //
835   //
836   //
837   //
838   //
839   //
840   //
841   //
842   //
843   //
844   //
845   //
846   //
847   //
848   //
849   //
850   //
851   //
852   //
853   //
854   //
855   //
856   //
857   //
858   //
859   //
860   //
861   //
862   //
863   //
864   //
865   //
866   //
867   //
868   //
869   //
870   //
871   //
872   //
873   //
874   //
875   //
876   //
877   //
878   //
879   //
880   //
881   //
882   //
883   //
884   //
885   //
886   //
887   //
888   //
889   //
890   //
891   //
892   //
893   //
894   //
895   //
896   //
897   //
898   //
899   //
900   //
901   //
902   //
903   //
904   //
905   //
906   //
907   //
908   //
909   //
910   //
911   //
912   //
913   //
914   //
915   //
916   //
917   //
918   //
919   //
920   //
921   //
922   //
923   //
924   //
925   //
926   //
927   //
928   //
929   //
930   //
931   //
932   //
933   //
934   //
935   //
936   //
937   //
938   //
939   //
940   //
941   //
942   //
943   //
944   //
945   //
946   //
947   //
948   //
949   //
950   //
951   //
952   //
953   //
954   //
955   //
956   //
957   //
958   //
959   //
960   //
961   //
962   //
963   //
964   //
965   //
966   //
967   //
968   //
969   //
970   //
971   //
972   //
973   //
974   //
975   //
976   //
977   //
978   //
979   //
980   //
981   //
982   //
983   //
984   //
985   //
986   //
987   //
988   //
989   //
990   //
991   //
992   //
993   //
994   //
995   //
996   //
997   //
998   //
999   //
1000  //

```



```

1 NTSTATUS __fastcall WinIo(
2     struct _DEVICE_OBJECT *DeviceObject,
3     PPHYSICAL_MEMORY_INFO IoBuffer,
4     __int64 InputBufferLength,
5     __int64 OutputBufferLength)
6 {
7     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
8
9     OutputBufferLength_1 = OutputBufferLength;
10    InputBufferLength_1 = InputBufferLength;
11    SectionHandle = 0i64;
12    Object = 0i64;
13    if ( !CheckAddress(IoBuffer->BusAddress.QuadPart, IoBuffer->AddressSpace) )
14        return STATUS_UNSUCCESSFUL;
15    if ( InputBufferLength_1 < 0x20 || OutputBufferLength_1 < 8 )
16        return STATUS_INSUFFICIENT_RESOURCES;
17    *(_QWORD *)&BusNumber = IoBuffer->BusAddress.QuadPart;
18    AddressSpace_2 = 0;
19    AddressSpace = 0;
20    AddressSpace_1 = IoBuffer->AddressSpace;
21    RtlInitUnicodeString(&DestinationString, L"\\Device\\PhysicalMemory");
22    ObjectAttributes.Length = 48;
23    ObjectAttributes.RootDirectory = 0i64;
24    ObjectAttributes.Attributes = 576;
25    ObjectAttributes.ObjectName = &DestinationString;
26    ObjectAttributes.SecurityDescriptor = 0i64;
27    ObjectAttributes.SecurityQualityOfService = 0i64;
28    result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29    if ( result >= 0 )
30    {
31        result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object);
32        if ( result >= 0 )
33        {
34            BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35            fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace_2, &AddressSpace_1);
36            fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &AddressSpace_1);
37            if ( !fSuccess )
38                return STATUS_UNSUCCESSFUL;
39            if ( !fSuccess_1 )
40                return STATUS_UNSUCCESSFUL;
41            SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42            SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43            if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44                return STATUS_UNSUCCESSFUL;
45            SectionOffset = TranslatedAddress;
46            BaseAddress = 0i64;
47            ViewSize = SizeView;
48        }
49    }
50    RtlInitUnicodeString (&physicalMemoryUnicodeString,
51                           L"\\Device\\PhysicalMemory");
52    InitializeObjectAttributes (&objectAttributes,
53                                &physicalMemoryUnicodeString,
54                                OBJ_CASE_INSENSITIVE,
55                                (HANDLE) NULL,
56                                (PSECURITY_DESCRIPTOR) NULL);
57    ntStatus = ZwOpenSection (&physicalMemoryHandle,
58                              SECTION_ALL_ACCESS,
59                              &objectAttributes);
60    if (!NT_SUCCESS(ntStatus))
61    {
62        MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));
63        goto done;
64    }
65    ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
66                                          SECTION_ALL_ACCESS,
67                                          (OBJECT_TYPE) NULL,
68                                          KernelMode,
69                                          &PhysicalMemorySection,
70                                          (POBJECT_HANDLE_INFORMATION) NULL);
71    if (!NT_SUCCESS(ntStatus))
72    {
73        MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle failed\n"));
74        goto close_handle;
75    }
76    // Initialize the physical addresses that will be translated
77    physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,
78                                              RtlConvertUlongToLargeInteger (length));
79    // reserved - baptiste

```



```

28 result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29 if ( result >= 0 )
30 {
31     result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32     if ( result >= 0 )
33     {
34         BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35         fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &AddressSpace, &TranslatedAddress);
36         fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &BusAddress);
37         if ( !fSuccess )
38             return STATUS_UNSUCCESSFUL;
39         if ( !fSuccess_1 )
40             return STATUS_UNSUCCESSFUL;
41         SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42         SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43         if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44             return STATUS_UNSUCCESSFUL;
45         SectionOffset = TranslatedAddress;
46         BaseAddress = 0i64;
47         ViewSize = SizeView;
48         v13 = ZwMapViewOfSection(
49             SectionHandle,
50             (HANDLE)0xFFFFFFFFFFFFFFFFi64,
51             &BaseAddress,
52             0i64,
53             SizeView,
54             &SectionOffset,
55             &ViewSize,
56             ViewShare,
57             0,
58             0x204u);
59         if ( v13 >= 0 )
60         {
61             BaseAddress = (char *)BaseAddress + TranslatedAddress.LowPart - (unsigned __int64)SectionOffset.LowPart;
62             IoBuffer->BusAddress.QuadPart = (LONGLONG)BaseAddress;
63             ZwClose(SectionHandle);
64             return 0;
65         }
66         else
67         {
68             ZwClose(SectionHandle);
69             return v13;
70         }
71     }
72 }

```

```

28 result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29 if ( result >= 0 )
30 {
31     result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32     if ( result >= 0 )
33     {
34         BusAddress.QuadPart = *(_QMWORD *)BusNumber + AddressSpace_1;
35         fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, //
36         fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, // Translate the physical addresses.
37         if ( !fSuccess ) //
38             return STATUS_UNSUCCESSFUL;
39         if ( !fSuccess_1 )
40             return STATUS_UNSUCCESSFUL;
41         SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42         SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43         if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44             return STATUS_UNSUCCESSFUL;
45         SectionOffset = TranslatedAddress;
46         BaseAddress = 0i64;
47         ViewSize = SizeView;
48         v13 = ZwMapViewOfSection(
49             SectionHandle,
50             (HANDLE)0xFFFFFFFFFFFFFFFFi64,
51             &BaseAddress,
52             0i64,
53             SizeView,
54             &SectionOffset,
55             &ViewSize,
56             ViewShare,
57             0,
58             0x204u);
59         if ( v13 >= 0 )
60         {
61             BaseAddress = (char *)BaseAddress + TranslatedAddress.LowPart - (unsigned __int64)SectionOffset.LowPart;
62             IoBuffer->BusAddress.QuadPart = (LONGLONG)BaseAddress;
63             ZwClose(SectionHandle);
64             return 0;
65         }
66         else
67         {
68             ZwClose(SectionHandle);
69             return v13;
70         }
71     }
72 }

```

```

translateBaseAddress =
    HalTranslateBusAddress (interfaceType,
                           busNumber,
                           physicalAddress,
                           &inIoSpace,
                           &physicalAddressBase);

```

```

translateEndAddress =
    HalTranslateBusAddress (interfaceType,
                           busNumber,
                           physicalAddressEnd,
                           &inIoSpace2,
                           &physicalAddressEnd);

```

d – Baptiste

29

```

28 result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29 if ( result >= 0 )
30 {
31     result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &Object, 0i64);
32     if ( result >= 0 )
33     {
34         BusAddress.QuadPart = *(_QMWORD *)BusNumber + AddressSpace_1;
35         fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, //
36         fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, // Translate the physical addresses.
37         if ( !fSuccess ) //
38             return STATUS_UNSUCCESSFUL;
39         if ( !fSuccess_1 )
40             return STATUS_UNSUCCESSFUL;
41         SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42         SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43         if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44             return STATUS_UNSUCCESSFUL;
45         SectionOffset = TranslatedAddress;
46         BaseAddress = 0i64;
47         ViewSize = SizeView;
48         v13 = ZwMapViewOfSection(
49             SectionHandle,
50             (HANDLE)0xFFFFFFFFFFFFFFFFi64,
51             &BaseAddress,
52             0i64,
53             SizeView,
54             &SectionOffset,
55             &ViewSize,
56             ViewShare,
57             0,
58             0x204u);
59         if ( v13 >= 0 )
60         {
61             BaseAddress = (char *)BaseAddress + TranslatedAddress.LowPart - (unsigned __int64)SectionOffset.LowPart;
62             IoBuffer->BusAddress.QuadPart = (LONGLONG)BaseAddress;
63             ZwClose(SectionHandle);
64             return 0;
65         }
66         else
67         {
68             ZwClose(SectionHandle);
69             return v13;
70         }
71     }
72 }

```

```

translateBaseAddress =
    HalTranslateBusAddress (interfaceType,
                           busNumber,
                           physicalAddress,
                           &inIoSpace,
                           &physicalAddressBase);

```

```

translateEndAddress =
    HalTranslateBusAddress (interfaceType,
                           busNumber,
                           physicalAddressEnd,
                           &inIoSpace2,
                           &physicalAddressEnd);

```

```

28 result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29 if ( result >= 0 )
30 {
31     result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &
32     if ( result >= 0 )
33     {
34         BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35         fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Ac
36         fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &Bus
37         if ( !fSuccess )
38             return STATUS_UNSUCCESSFUL;
39         if ( !fSuccess_1 )
40             return STATUS_UNSUCCESSFUL;
41         SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42         SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43         if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44             return STATUS_UNSUCCESSFUL;
45         SectionOffset = TranslatedAddress;
46         BaseAddress = 0i64;
47         ViewSize = SizeView;
48         v13 = ZwMapViewOfSection(
49             SectionHandle,
50             (HANDLE)0xFFFFFFFFFFFFFFFFi64,
51             &BaseAddress,
52             0i64,
53             SizeView,
54             &SectionOffset,
55             &ViewSize,
56             ViewShare,
57             0,
58             0x204u);
59         if ( v13 >= 0 )
60         {
61             BaseAddress = (char *)BaseAddress + TranslatedAddress.LowPart - (unsigned
62             IoBuffer->BusAddress.QuadPart = (LONGLONG)BaseAddress;
63             ZwClose(SectionHandle);
64             return 0;
65         }
66         else
67         {
68             ZwClose(SectionHandle);
69             return v13;
70         }
71     }
72 }

```

```

//
// Map the section
//
ntStatus = ZwMapViewOfSection (physicalMemoryHandle,
                                (HANDLE) -1,
                                &virtualAddress,
                                0L,
                                length,
                                &viewBase,
                                &length,
                                ViewShare,
                                0,
                                PAGE_READWRITE | PAGE_NOCACHE);

if (!NT_SUCCESS(ntStatus))
{
    MapMemKdPrint (("MAPMEM.SYS: ZwMapViewOfSection failed\n"));

    goto close_handle;
}

//
// Mapping the section above rounded the physical address down to the
// nearest 64 K boundary. Now return a virtual address that sits where
// we want by adding in the offset from the beginning of the section.
//

(ULONG) virtualAddress += (ULONG)physicalAddressBase.LowPart -
                          (ULONG)viewBase.LowPart;

*((PVOID *) IoBuffer) = virtualAddress;
}

ntStatus = STATUS_SUCCESS;

close_handle:

    ZwClose (physicalMemoryHandle);

done:

    return ntStatus;

```

```

28 result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29 if ( result >= 0 )
30 {
31     result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &
32     if ( result >= 0 )
33     {
34         BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35         fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Ac
36         fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &Bus
37         if ( !fSuccess )
38             return STATUS_UNSUCCESSFUL;
39         if ( !fSuccess_1 )
40             return STATUS_UNSUCCESSFUL;
41         SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42         SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43         if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44             return STATUS_UNSUCCESSFUL;
45         SectionOffset = TranslatedAddress;
46         BaseAddress = 0i64;
47         ViewSize = SizeView;
48         v13 = ZwMapViewOfSection(
49             SectionHandle,
50             (HANDLE)0xFFFFFFFFFFFFFFFFi64,
51             &BaseAddress,
52             0i64,
53             SizeView,
54             &SectionOffset,
55             &ViewSize,
56             ViewShare,
57             0,
58             0x204u);
59         if ( v13 >= 0 )
60         {
61             BaseAddress = (char *)BaseAddress + TranslatedAddress.LowPart - (unsigned
62             IoBuffer->BusAddress.QuadPart = (LONGLONG)BaseAddress;
63             ZwClose(SectionHandle);
64             return 0;
65         }
66         else
67         {
68             ZwClose(SectionHandle);
69             return v13;
70         }
71     }
72 }

```

```

//
// Map the section
//
ntStatus = ZwMapViewOfSection (physicalMemoryHandle,
                                (HANDLE) -1,
                                &virtualAddress,
                                0L,
                                length,
                                &viewBase,
                                &length,
                                ViewShare,
                                0,
                                PAGE_READWRITE | PAGE_NOCACHE);

if (!NT_SUCCESS(ntStatus))
{
    MapMemKdPrint (("MAPMEM.SYS: ZwMapViewOfSection failed\n"));

    goto close_handle;
}

//
// Mapping the section above rounded the physical address down to the
// nearest 64 K boundary. Now return a virtual address that sits where
// we want by adding in the offset from the beginning of the section.
//

(ULONG) virtualAddress += (ULONG)physicalAddressBase.LowPart -
(ULONG)viewBase.LowPart;

*((PVOID *) IoBuffer) = virtualAddress;

ntStatus = STATUS_SUCCESS;

close_handle:

    ZwClose (physicalMemoryHandle);

done:

    return ntStatus;

```



```

28 result = ZwOpenSection(&SectionHandle, SECTION_ALL_ACCESS, &ObjectAttributes);
29 if ( result >= 0 )
30 {
31     result = ObReferenceObjectByHandle(SectionHandle, SECTION_ALL_ACCESS, 0i64, 0, &
32     if ( result >= 0 )
33     {
34         BusAddress.QuadPart = *(_QWORD *)&BusNumber + AddressSpace_1;
35         fSuccess = HalTranslateBusAddress(Isa, 0, *(PHYSICAL_ADDRESS *)&BusNumber, &Ac
36         fSuccess_1 = HalTranslateBusAddress(Isa, 0, BusAddress, &AddressSpace_2, &Bus
37         if ( !fSuccess )
38             return STATUS_UNSUCCESSFUL;
39         if ( !fSuccess_1 )
40             return STATUS_UNSUCCESSFUL;
41         SizeView = BusAddress.LowPart - TranslatedAddress.LowPart;
42         SectionOffset.QuadPart = BusAddress.QuadPart - TranslatedAddress.QuadPart;
43         if ( BusAddress.LowPart == TranslatedAddress.LowPart )
44             return STATUS_UNSUCCESSFUL;
45         SectionOffset = TranslatedAddress;
46         BaseAddress = 0i64;
47         ViewSize = SizeView;
48         v13 = ZwMapViewOfSection(
49             SectionHandle,
50             (HANDLE)0xFFFFFFFFFFFFFFFFi64,
51             &BaseAddress,
52             0i64,
53             SizeView,
54             &SectionOffset,
55             &ViewSize,
56             ViewShare,
57             0,
58             0x204u);
59         if ( v13 >= 0 )
60         {
61             BaseAddress = (char *)BaseAddress + TranslatedAddress.LowPart - (unsigned
62             IoBuffer->BusAddress.QuadPart = (LONGLONG)BaseAddress;
63             ZwClose(SectionHandle);
64             return 0;
65         }
66         else
67         {
68             ZwClose(SectionHandle);
69             return v13;
70         }
71     }
72 }

```

```

//
// Map the section
//
ntStatus = ZwMapViewOfSection (physicalMemoryHandle,
                                (HANDLE) -1,
                                &virtualAddress,
                                0L,
                                length,
                                &viewBase,
                                &length,
                                ViewShare,
                                0,
                                PAGE_READWRITE | PAGE_NOCACHE);

if (!NT_SUCCESS(ntStatus))
{
    MapMemKdPrint (("MAPMEM.SYS: ZwMapViewOfSection failed\n"));

    goto close_handle;
}

//
// Mapping the section above rounded the physical address down to the
// nearest 64 K boundary. Now return a virtual address that sits where
// we want by adding in the offset from the beginning of the section.
//

(ULONG) virtualAddress += (ULONG)physicalAddressBase.LowPart -
(ULONG)viewBase.LowPart;

*((PVOID *) IoBuffer) = virtualAddress;

ntStatus = STATUS_SUCCESS;

close_handle:
ZwClose (physicalMemoryHandle);

done:
return ntStatus;

```

```
case 0x8000202C:
    if ( (_DWORD)InputBufferLength == 8 )
    {
        ContentSystem = GlobalStorage;
        if ( *(_DWORD *)SystemBuffer != 0x80000000 )
            ContentSystem = *(_DWORD *)SystemBuffer;
        GlobalStorage = ContentSystem;
        GlobalStorage = *((_DWORD *)SystemBuffer + 1) + ContentSystem;
        *(_DWORD *)SystemBuffer = GlobalStorage;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 8i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;
```

Write where you want



```
case 0x80002048:
```

```
if ( (_DWORD)InputBufferLength == 0x30 )
```

```
{
```

```
    offset = *((_QWORD *)SystemBuffer + 1); // Read
```

```
    if ( offset )
```

```
    {
```

```
        switch ( *((_DWORD *)SystemBuffer + 6) )
```

```
        {
```

```
            case 1:
```

```
                *((_DWORD *)SystemBuffer + 7) = *(unsigned __int8 *)((*((unsigned int *)SystemBuffer + 5) + offset));
```

```
                break;
```

```
            case 2:
```

```
                *((_DWORD *)SystemBuffer + 7) = *(unsigned __int16 *)((*((unsigned int *)SystemBuffer + 5) + offset));
```

```
                break;
```

```
            case 4:
```

```
                *((_DWORD *)SystemBuffer + 7) = *(_DWORD *)((*((unsigned int *)SystemBuffer + 5) + offset));
```

```
                break;
```

```
        }
```

```
        Irp->IoStatus.Status = 0;
```

```
        Irp->IoStatus.Information = 48i64;
```

```
    }
```

```
else
```

```
{
```

```
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
```

```
}
```

```
typedef struct _READ_WHERE_YOU_WANT {
```

```
    PVOID Reserved_1;           // +0x00
```

```
    ULONGLONG Offset;          // +0x08
```

```
    PVOID BaseAddress;         // +0x10
```

```
    DWORD SizeSwitch;          // +0x18
```

```
    union {                     // +0x1C
```

```
        BYTE OutByte;
```

```
        WORD OutWord;
```

```
        DWORD OutDWord;
```

```
    }
```

```
    PVOID Reserved_3;          // +0x20
```

```
    PVOID Reserved_4;          // +0x28
```

```
} READ_WHERE_YOU_WANT, *PREAD_WHERE_YOU_WANT;
```

```

case 0x80002040:
    if ( (_DWORD)InputBufferLength == 48 )
    {
        status_1 = ((__int64 (__fastcall *)(PVOID, __int64, void *, __int64))MapIoSpace)(
            SystemBuffer,
            InputBufferLength,
            &loc_114B6,
            OutputBufferLength);
        Irp->IoStatus.Status = status_1;
        if ( status_1 < 0 )
            Irp->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
        else
            Irp->IoStatus.Information = 48i64;
    }
    else
    {
        Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
    }
    break;
case 0x80002044:
    if ( (_DWORD)InputBufferLength == 48 )
    {
        v10 = (void *)*((_QWORD *)SystemBuffer + 1);
        if ( v10 )
        {
            MmUnmapIoSpace(v10, *((unsigned int *)SystemBuffer + 4));
            Irp->IoStatus.Status = 0;
        }
        else
        {
            Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
        }
    }
    else
    {
        Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
    }
    break;

```

```

case 0x80002040:
if ( ( _DWORD)InputBufferLength == 48 )
{
    status_1 = ((__int64 (__fastcall *)(PVOID, __int64, void *, __int64))MapIoSpace)(
        SystemBuffer,
        InputBufferLength,
        &loc_114B6,
        OutputBufferLength);
    Irp->IoStatus.Status = status_1;
    if ( status_1 < 0 )
        Irp->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
    else
        Irp->IoStatus.Information = 48i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
case 0x80002044:
if ( ( _DWORD)InputBufferLength == 48 )
{
    v10 = (void *)*((_QWORD *)SystemBuffer + 1);
    if ( v10 )
    {
        MmUnmapIoSpace(v10, *((unsigned int *)SystemBuffer + 4));
        Irp->IoStatus.Status = 0;
    }
    else
    {
        Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
    }
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;

```

```

1  __int64 __fastcall sub_113A0(PHYSICAL_ADDRESS *a1)
2  {
3      PVOID v3; // rax
4      ULONG AddressSpace; // [rsp+30h] [rbp-18h] BYREF
5      LARGE_INTEGER PhysicalAddress; // [rsp+38h] [rbp-10h] BYREF
6
7      if ( !CheckAddress(a1->QuadPart, a1[2].LowPart) )
8          return 3221225473i64;
9      AddressSpace = 0;
10     if ( !HalTranslateBusAddress(Isa, 0, *a1, &AddressSpace, &PhysicalAddress) )
11         return 3221225626i64;
12     v3 = MmMapIoSpace(PhysicalAddress, a1[2].LowPart, MmNonCached);
13     if ( !v3 )
14         return 3221225626i64;
15     a1[1].QuadPart = (LONGLONG)v3;
16     return 0i64;
17 }

```



```

case 0x80002040:
if ( ( _DWORD)InputBufferLength == 48 )
{
    status_1 = ((__int64 (__fastcall *)(PVOID, __int64, void *, __int64))MapIoSpace)(
        SystemBuffer,
        InputBufferLength,
        &loc_114B6,
        OutputBufferLength);
    Irp->IoStatus.Status = status_1;
    if ( status_1 < 0 )
        Irp->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
    else
        Irp->IoStatus.Information = 48i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
case 0x80002044:
if ( ( _DWORD)InputBufferLength == 48 )
{
    v10 = (void *)*((_QWORD *)SystemBuffer + 1);
    if ( v10 )
    {
        MmUnmapIoSpace(v10, *((unsigned int *)SystemBuffer + 4));
        Irp->IoStatus.Status = 0;
    }
    else
    {
        Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
    }
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;

```

```


1  __int64 __fastcall sub_113A0(PHYSICAL_ADDRESS *a1)
2  {
3      PVOID v3; // rax
4      ULONG AddressSpace; // [rsp+30h] [rbp-18h] BYREF
5      LARGE_INTEGER PhysicalAddress; // [rsp+38h] [rbp-10h] BYREF
6
7      if ( !CheckAddress(a1->QuadPart, a1[2].LowPart) )
8          return 3221225473i64;
9      AddressSpace = 0;
10     if ( !HalTranslateBusAddress(Isa, 0, *a1, &AddressSpace, &PhysicalAddress) )
11         return 3221225626i64;
12     v3 = MmMapIoSpace(PhysicalAddress, a1[2].LowPart, MmNonCached);
13     if ( !v3 )
14         return 3221225626i64;
15     a1[1].QuadPart = (LONGLONG)v3;
16     return 0i64;
17 }

```

The **MmMapIoSpace** routine maps the given physical address range to nonpaged system space.

## Syntax

C++

 Copy

```
PVOID MmMapIoSpace(  
    [in] PHYSICAL_ADDRESS    PhysicalAddress,  
    [in] SIZE_T              NumberOfBytes,  
    [in] MEMORY_CACHING_TYPE CacheType  
);
```

## Parameters

[in] **PhysicalAddress**

Specifies the starting physical address of the I/O range to be mapped.

[in] **NumberOfBytes**

Specifies a value greater than zero, indicating the number of bytes to be mapped.

[in] **CacheType**

Specifies a **MEMORY\_CACHING\_TYPE** value, which indicates the cache attribute to use to map the physical address range.

## Return value

**MmMapIoSpace** returns the base virtual address that maps the base physical address for the range. If space for mapping the range is insufficient, it returns **NULL**.

```
case 0x80002030:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        v24 = __readmsr(*(_DWORD *)SystemBuffer); // MSR access :)
        *((_DWORD *)SystemBuffer + 1) = HIDWORD(v24);
        *((_DWORD *)SystemBuffer + 2) = v24;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;
case 0x80002034:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        // Idem, with write access.
        __writemsr(*(_DWORD *)SystemBuffer, __PAIR64__(*((_DWORD *)SystemBuffer + 1), *((_DWORD *)SystemBuffer + 2)));
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;
```

case 0x80002030:

```
if ( (_DWORD)InputBufferLength == 12 )
```

```
{
```

```
    v24 = __readmsr(*(_DWORD *)SystemBuffer); // MSR access :)
```

```
    *(_DWORD *)SystemBuffer + 1 = HIWORD(v24);
```

```
    *(_DWORD *)SystemBuffer + 2 = v24;
```

```
    Irp->IoStatus.Status = 0;
```

```
    Irp->IoStatus.Information = 12i64;
```

```
}
```

```
else
```

```
{
```

```
    Irp->IoStatus.Status = -1073741811;
```

```
}
```

```
break;
```

case 0x80002034:

```
if ( (_DWORD)InputBufferLength == 12 )
```

```
{
```

```
    // Idem, with write access.
```

```
    __writemsr(*(_DWORD *)SystemBuffer, __PAIR64__(*(_DWORD *)SystemBuffer + 1, *(_DWORD *)SystemBuffer + 2));
```

```
    Irp->IoStatus.Status = 0;
```

```
    Irp->IoStatus.Information = 12i64;
```

```
}
```

```
else
```

```
{
```

```
    Irp->IoStatus.Status = -1073741811;
```

```
}
```

```
break;
```

```

131         break;
132
133         case IOCTL_OLS_READ_MSR:
134             status = ReadMsr(
135                 pIrp->AssociatedIrp.SystemBuffer,
136                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137                 pIrp->AssociatedIrp.SystemBuffer,
138                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139                 (ULONG*)&pIrp->IoStatus.Information
140             );
141             break;
142         case IOCTL_OLS_WRITE_MSR:
143             status = WriteMsr(
144                 pIrp->AssociatedIrp.SystemBuffer,
145                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146                 pIrp->AssociatedIrp.SystemBuffer,
147                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148                 (ULONG*)&pIrp->IoStatus.Information
149             );
150             break;

```

```

case 0x80002030:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        v24 = __readmsr((__DWORD *)SystemBuffer); // MSR access :)
        *((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
        *((_DWORD *)SystemBuffer + 2) = v24;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;
case 0x80002034:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        // Idem, with write access.
        __writemsr((__DWORD *)SystemBuffer, __PAIR64__((_DWORD *)SystemBuffer + 1), *((_DWORD *)SystemBuffer + 2)));
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;

```



```

case 0x80002030:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        v24 = __readmsr(*(_DWORD *)SystemBuffer); // MSR access :)
        *((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
        *((_DWORD *)SystemBuffer + 2) = v24;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;

```

```

case 0x80002034:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        // Idem, with write access.
        __writemsr(*(_DWORD *)SystemBuffer, __PAIR64__((_DWORD *)SystemBuffer + 1), *((_DWORD *)SystemBuffer + 2)));
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;

```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```

131         break;
132
133         case IOCTL_OLS_READ_MSR:
134             status = ReadMsr(
135                 pIrp->AssociatedIrp.SystemBuffer,
136                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137                 pIrp->AssociatedIrp.SystemBuffer,
138                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139                 (ULONG*)&pIrp->IoStatus.Information
140             );
141             break;
142         case IOCTL_OLS_WRITE_MSR:
143             status = WriteMsr(
144                 pIrp->AssociatedIrp.SystemBuffer,
145                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146                 pIrp->AssociatedIrp.SystemBuffer,
147                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148                 (ULONG*)&pIrp->IoStatus.Information
149             );
150             break;

```

```
case 0x8002030:
if ( (_DWORD)InputBufferLength == 12 )
{
v24 = __readmsr((__DWORD *)SystemBuffer); // MSR access :)
*((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
*((_DWORD *)SystemBuffer + 2) = v24;
```

```
NTSTATUS
ReadMsr(void* lpInBuffer,
        ULONG nInBufferSize,
        void* lpOutBuffer,
        ULONG nOutBufferSize,
        ULONG* lpBytesReturned)
{
    __try
    {
        UNREFERENCED_PARAMETER(nInBufferSize);
        if (nOutBufferSize < 8)
        {
            *lpBytesReturned = 0;
            return STATUS_BUFFER_TOO_SMALL;
        }
        #ifdef _ARM64_
        ULONGLONG data = _ReadStatusReg((__ULONG*)lpInBuffer);
        #else
        ULONGLONG data = __readmsr((__ULONG*)lpInBuffer);
        #endif
        memcpy((PULONG)lpOutBuffer, &data, 8);
        *lpBytesReturned = 8;
        return STATUS_SUCCESS;
    }
    break;
```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```
131 break;
132
133 case IOCTL_OLS_READ_MSR:
134 status = ReadMsr(
135     pIrp->AssociatedIrp.SystemBuffer,
136     pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137     pIrp->AssociatedIrp.SystemBuffer,
138     pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139     (ULONG*)&pIrp->IoStatus.Information
140 );
141 break;
142 case IOCTL_OLS_WRITE_MSR:
143 status = WriteMsr(
144     pIrp->AssociatedIrp.SystemBuffer,
145     pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146     pIrp->AssociatedIrp.SystemBuffer,
147     pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148     (ULONG*)&pIrp->IoStatus.Information
149 );
150 break;
```

```
write access.
WORD *)SystemBuffer + 1), *((_DWORD *)SystemBuffer + 2)));
```

```
case 0x80002030:
if ( (_DWORD)InputBufferLength == 12 )
{
v24 = __readmsr((__DWORD *)SystemBuffer); // MSR access :)
*((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
*((_DWORD *)SystemBuffer + 2) = v24;
Irp->IoStatus = STATUS_SUCCESS;
break;
}
else
{
Irp->IoStatus = STATUS_INVALID_DEVICE_STATE;
break;
}
case 0x80002034:
if ( (_DWORD)InputBufferLength == 8 )
{
__writemsr((__DWORD *)SystemBuffer + 1, *((_DWORD *)SystemBuffer + 2));
Irp->IoStatus = STATUS_SUCCESS;
break;
}
else
{
Irp->IoStatus = STATUS_INVALID_DEVICE_STATE;
break;
}
```

```
NTSTATUS
ReadMsr(void* lpInBuffer,
        ULONG nInBufferSize,
        void* lpOutBuffer,
        ULONG nOutBufferSize,
        ULONG* lpBytesReturned)
{
    __try
    {
        UNREFERENCED_PARAMETER(nInBufferSize);
        if (nOutBufferSize < 8)
        {
            *lpBytesReturned = 0;
            return STATUS_BUFFER_TOO_SMALL;
        }

        #ifdef _ARM64_
        ULONGLONG data = _ReadStatusReg((__ULONG*)lpInBuffer);
        #else
        ULONGLONG data = __readmsr((__ULONG*)lpInBuffer);
        #endif

        memcpy((PULONG)lpOutBuffer, &data, 8);
        *lpBytesReturned = 8;
        return STATUS_SUCCESS;
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        return STATUS_INVALID_DEVICE_STATE;
    }
}
```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```
131 break;
132
133 case IOCTL_OLS_READ_MSR:
134 status = ReadMsr(
135     pIrp->AssociatedIrp.SystemBuffer,
136     pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137     pIrp->AssociatedIrp.SystemBuffer,
138     pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139     (ULONG*)&pIrp->IoStatus.Information
140 );
141 break;
142 case IOCTL_OLS_WRITE_MSR:
143 status = WriteMsr(
144     pIrp->AssociatedIrp.SystemBuffer,
145     pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146     pIrp->AssociatedIrp.SystemBuffer,
147     pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148     (ULONG*)&pIrp->IoStatus.Information
149 );
150 break;
```

```
write access.
*((_DWORD *)SystemBuffer + 1, *((_DWORD *)SystemBuffer + 2));
```

```

case 0x80002030:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        v24 = __readmsr((__DWORD *)SystemBuffer); // MSR access :)
        *((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
        *((_DWORD *)SystemBuffer + 2) = v24;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;
case 0x80002034:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        // Idem, with write access.
        __writemsr((__DWORD *)SystemBuffer, __PAIR64__((_DWORD *)SystemBuffer + 1), *((_DWORD *)SystemBuffer + 2)));
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;

```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code

Blame

722 lines (614 loc) · 17.1 KB

```

131         break;
132
133     case IOCTL_OLS_READ_MSR:
134         status = ReadMsr(
135             pIrp->AssociatedIrp.SystemBuffer,
136             pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137             pIrp->AssociatedIrp.SystemBuffer,
138             pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139             (ULONG*)&pIrp->IoStatus.Information
140         );
141         break;
142     case IOCTL_OLS_WRITE_MSR:
143         status = WriteMsr(
144             pIrp->AssociatedIrp.SystemBuffer,
145             pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146             pIrp->AssociatedIrp.SystemBuffer,
147             pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148             (ULONG*)&pIrp->IoStatus.Information
149         );
150         break;

```



```

131         break;
132
133         case IOCTL_OLS_READ_MSR:
134             status = ReadMsr(
135                 pIrp->AssociatedIrp.SystemBuffer,
136                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137                 pIrp->AssociatedIrp.SystemBuffer,
138                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139                 (ULONG*)&pIrp->IoStatus.Information
140             );
141             break;
142         case IOCTL_OLS_WRITE_MSR:
143             status = WriteMsr(
144                 pIrp->AssociatedIrp.SystemBuffer,
145                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146                 pIrp->AssociatedIrp.SystemBuffer,
147                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148                 (ULONG*)&pIrp->IoStatus.Information
149             );
150             break;

```

```

case 0x80002030:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        v24 = __readmsr(*(_DWORD *)SystemBuffer); // MSR access :)
        *((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
        *((_DWORD *)SystemBuffer + 2) = v24;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;
case 0x80002034:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        // Idem, with write access.
        __writemsr(*(_DWORD *)SystemBuffer, __PAIR64__((_DWORD *)SystemBuffer + 1), *((_DWORD *)SystemBuffer + 2));
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = -1073741811;
    }
    break;

```





```
case 0x8002030:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        v24 = __readmsr((__DWORD *)SystemBuffer); // MSR access :)
        *((_DWORD *)SystemBuffer + 1) = HIWORD(v24);
        *((_DWORD *)SystemBuffer + 2) = v24;
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = 0;
    }
    break;
case 0x8002034:
    if ( (_DWORD)InputBufferLength == 12 )
    {
        __writemsr((__DWORD *)SystemBuffer + 1, *((_DWORD *)SystemBuffer + 2));
        Irp->IoStatus.Status = 0;
        Irp->IoStatus.Information = 12i64;
    }
    else
    {
        Irp->IoStatus.Status = 0;
    }
    break;
```

```
NTSTATUS
WriteMsr(void* lpInBuffer,
         ULONG nInBufferSize,
         void* lpOutBuffer,
         ULONG nOutBufferSize,
         ULONG* lpBytesReturned)
{
    __try
    {
        UNREFERENCED_PARAMETER(lpOutBuffer);
        if (BufferSizeCheck(nInBufferSize, nOutBufferSize, lpBytesReturned,
                           OLS_WRITE_MSR_INPUT* param = (OLS_WRITE_MSR_INPUT*)lpInBuffer;

#ifdef _ARM64_
        _WriteStatusReg(param->Register, param->Value.QuadPart);
#else
        __writemsr(param->Register, param->Value.QuadPart);
#endif

        *lpBytesReturned = 0;
        return STATUS_SUCCESS;
    }
}
```

WinRing0 / WinRing0Sys / OpenLibSys.c

Code Blame 722 lines (614 loc) · 17.1 KB

```
131         break;
132
133         case IOCTL_OLS_READ_MSR:
134             status = ReadMsr(
135                 pIrp->AssociatedIrp.SystemBuffer,
136                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
137                 pIrp->AssociatedIrp.SystemBuffer,
138                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
139                 (ULONG*)&pIrp->IoStatus.Information
140             );
141             break;
142         case IOCTL_OLS_WRITE_MSR:
143             status = WriteMsr(
144                 pIrp->AssociatedIrp.SystemBuffer,
145                 pIrpStack->Parameters.DeviceIoControl.InputBufferLength,
146                 pIrp->AssociatedIrp.SystemBuffer,
147                 pIrpStack->Parameters.DeviceIoControl.OutputBufferLength,
148                 (ULONG*)&pIrp->IoStatus.Information
149             );
150             break;
```

```
...ffer + 1), *((_DWORD *)SystemBuffer + 2));
```



```
case 0x80002008:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content = __inbyte(*(_WORD *)SystemBuffer); // IN (assembly)
    *((_DWORD *)SystemBuffer + 1) = Content;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
case 0x8000200C:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content_1 = __inword(*(_WORD *)SystemBuffer); // IN (assembly)
    *((_DWORD *)SystemBuffer + 1) = Content_1;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
case 0x80002010:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content_2 = __indword(*(_WORD *)SystemBuffer); // IN (assembly)
    *((_DWORD *)SystemBuffer + 1) = Content_2;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
```



```
case 0x80002008:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content = __inbyte(*(_WORD *)SystemBuffer); // IN (assembly)
    *((_DWORD *)SystemBuffer + 1) = Content;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAM
}
break;
case 0x8000200C:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content_1 = __inword(*(_WORD *)SystemBuffer
    *((_DWORD *)SystemBuffer + 1) = Content_1;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAM
}
break;
case 0x80002010:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content_2 = __indword(*(_WORD *)SystemBuffer); // IN (assembly)
    *((_DWORD *)SystemBuffer + 1) = Content_2;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
```

winio / Source / Drv / WinIo.c

Code

Blame

478 lines (346 loc) · 12 KB

```
188         case IOCTL_WINIO_WRITEPORT:
189
190             KdPrint(("IOCTL_WINIO_WRITEPORT"));
191
192             if (dwInputBufferLength)
193             {
194                 memcpy (&PortStruct, pvIOBuffer, dwInputBufferLength);
195
196                 switch (PortStruct.bSize)
197                 {
198                     case 1:
199                         WRITE_PORT_UCHAR((PUCHAR)(USHORT)PortStruct.wPortAddr, (UCHAR)PortStruct.dwPortVal);
200                         break;
201
202                     case 2:
203                         WRITE_PORT_USHORT((PUSHORT)(USHORT)PortStruct.wPortAddr, (USHORT)PortStruct.dwPortVal);
204                         break;
205
206                     case 4:
207                         WRITE_PORT_ULONG((PULONG)(USHORT)PortStruct.wPortAddr, PortStruct.dwPortVal);
208                         break;
209                 }
210             }
```



```
case 0x80002008:
if ( (_DWORD)InputBufferLength == 8 )
{
    Content = __inbyte(*(_WORD *)SystemBuffer); // IN (assembly)
    *((_DWORD *)SystemBuffer + 1) = Content;
    Irp->IoStatus.Status = 0;
    Irp->IoStatus.Information = 8i64;
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAM

```

winio / Source / Drv / WinIo.c

Code

Blame

478 lines (346 loc) · 12 KB

188

case IOCTL\_WINIO\_WRITEPORT:

189

190

KdPrint(("IOCTL\_WINIO\_WRITEPORT"));

Filter by title

WRITE\_PORT\_BUFFER\_ULONG function

WRITE\_PORT\_BUFFER\_USHORT function

WRITE\_PORT\_UCHAR function

WRITE\_PORT\_ULONG function

WRITE\_PORT\_USHORT function

WRITE\_REGISTER\_BUFFER\_UCHAR function

WRITE\_REGISTER\_BUFFER\_ULONG function

WRITE\_REGISTER\_BUFFER\_ULONG64 function

WRITE\_REGISTER\_BUFFER\_USHORT function

WRITE\_REGISTER\_UCHAR function

WRITE\_REGISTER\_ULONG function

WRITE\_REGISTER\_ULONG64 function

WRITE\_REGISTER\_USHORT function

WriteInt32NoFence function

WriteInt32Raw function

WriteInt32Release function

The WRITE\_PORT\_USHORT routine writes a USHORT value to the specified port address.

## Syntax

C++

Copy

```
NTHALAPI VOID WRITE_PORT_USHORT(
    [in] PUSHORT Port,
    [in] USHORT Value
);
```

## Parameters

[in] Port

Pointer to the port, which must be a mapped memory range in I/O space.

[in] Value

Specifies a USHORT value to be written to the port.

uffer, dwInputBufferLength);

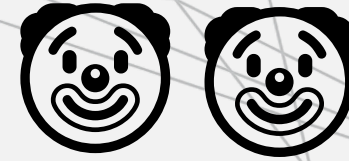
PUCHAR)(USHORT)PortStruct.wPortAddr, (UCHAR)PortStruct.dwPortVal);

(PUSHORT)(USHORT)PortStruct.wPortAddr, (USHORT)PortStruct.dwPortVal);

PULONG)(USHORT)PortStruct.wPortAddr, PortStruct.dwPortVal);

```
}
else
{
    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
}
break;
```





# MSI Afterburner

MSI Afterburner is the world's most recognized and widely used graphics card overclocking utility which gives you full control of your ~~graphics cards.~~ *computer*

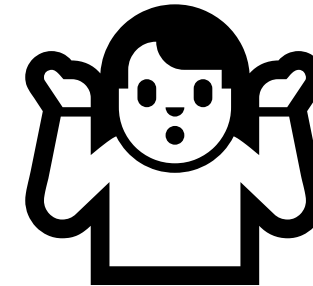


Micro-Star International

<https://www.msi.com> › Graphics-Card



GeForce® GT 1030 AERO ITX 4GD4 OC - MSI





# Applications in real world

---

Illustration of real cases ...

# Widely used
























- At Windows XP time:
  - The name of the game was to write you own rootkit driver.
    - Not so hard to do it with a decent tutorial.
    - 2000's: Find a new “trick” to hide something in the kernel.
    - Almost no serious security in the kernel regarding today.
- But came Windows Vista:
  - Driver must be signed on x64 CPU architecture.
  - Harder to write and deploy you own driver...
  - Good news:
    - Some former/past/legacy drivers are still there for you, full of vulnerabilities.
    - Thanks to backward compatibility, most of the API they use still work today.
    - Signed forever, if the target system is not up to date, it is Christmas 🎅.

# Is it really used?

<https://www.rapid7.com/blog/post/2021/12/13/driver-based-attacks-past-and-present/>

## Known usage in the wild






















BYOVD is a common technique used by advanced adversaries and opportunistic attackers alike. To illustrate this, the following table is a non-exhaustive list of well-known advisories/malware that use the BYOVD tactic, the associated vulnerable driver, and the associated vulnerability where applicable or known.

Year Published	Adversary/Malware	Driver Name	Driver Creator	CVE ID
2021	Candiru 	physmem.sys 	Hilscher	N/A
2021	Iron Tiger 	procexp152.sys 	Process Explorer 	N/A
2021	Iron Tiger	cpuz141.sys 	CPUID CPU-Z	CVE-2017-15303 
2021	GhostEmperor 	dbk64.sys	CheatEngine 	N/A
2021	ZINC 	viraglt64.sys 	Vir.IT eXplorer	CVE-2017-16238 
2021	Various Cryptominers using XMRig 	winring00x64.sys 	OpenLibSys 	N/A
2021	TunnelSnake 	vboxdrv.sys 	VirtualBox	CVE-2008-3431 
2020	RobbinHood 	gdrv.sys 	Gigabyte	CVE-2018-19320 
2020	Trickbot 	rwdrv.sys	RWEverything 	N/A

# Is it really used?

<https://www.rapid7.com/blog/post/2021/12/13/driver-based-attacks-past-and-present/>

## Known usage in the wild

2020	InvisiMole 	speedfan.sys 	Comparetti Speedfan	CVE-2007-5633 
2020	ZeroCleare 	vboxdrv.sys	VirtualBox	Unclear
2020	Winnti Group 	vboxdrv.sys	VirtualBox	CVE-2008-3431
2020	AcidBox 	vboxdrv.sys	VirtualBox	Unclear
2020	Dustman 	vboxdrv.sys	VirtualBox	CVE-2008-3431
2019	Doppelpaymer 	kprocesshacker.sys 	Process Hacker 	N/A
2018	LoJax 	rwdrv.sys	RWEverything	N/A
2018	Slingshot 	sandra.sys 	SiSoftware Sandra	CVE-2010-1592 
2018	Slingshot	elbycdio.sys	Elaborate Bytes	CVE-2009-0824 
2018	Slingshot	speedfan.sys	Alfredo Milani Comparetti Speedfan	CVE-2007-5633
2018	Slingshot	goad.sys	??	Unclear
2017	The Lamberts 	sandra.sys	SiSoftware Sandra	CVE-2010-1592
2016	Remsec 	aswsnx.sys	Avast!	Unclear
2016	Remsec	sandbox.sys	Agnitum Output	Unclear
2015	Equation Group 	elbycdio.sys	CloneCD	CVE-2009-0824 
2020	Trickbot 	rwdrv.sys	RWEverything 	N/A



# Is it really used?

## Known usage in the wild

2020	InvisiMole <a href="#">✗</a>	speedfan.sys <a href="#">✗</a>	Comparetti Speedfan	CVE-2007-5633 <a href="#">✗</a>
2020	ZeroCleare <a href="#">✗</a>	vboxdrv.sys	VirtualBox	Unclear
2020	Winnti Group <a href="#">✗</a>	vboxdrv.sys	VirtualBox	CVE-2008-3431
2020	AcidBox <a href="#">✗</a>	vboxdrv.sys	VirtualBox	Unclear
2020	Dustman <a href="#">✗</a>	vboxdrv.sys	VirtualBox	CVE-2008-3431
2019	Doppelpaymer <a href="#">✗</a>	kprocesshacker.sys <a href="#">✗</a>	Process Hacker <a href="#">✗</a>	N/A

2015	Derusbi <a href="#">✗</a>	nicm.sys <a href="#">✗</a> , nscm.sys <a href="#">✗</a> , ncpl.sys <a href="#">✗</a>	Novell	CVE-2013-3956 <a href="#">✗</a>
2014	Turla <a href="#">✗</a>	vboxdrv.sys	VirtualBox	CVE-2008-3431
2012	Shamoon <a href="#">✗</a>	elrawdsk.sys	Eldos Rawdisk	N/A

2018	Slingshot	goad.sys	??	Unclear
2017	The Lamberts <a href="#">✗</a>	sandra.sys	SiSoftware Sandra	CVE-2010-1592
2016	Remsec <a href="#">✗</a>	aswsnx.sys	Avast!	Unclear
2016	Remsec	sandbox.sys	Agnitum Output	Unclear
2015	Equation Group <a href="#">✗</a>	elbycdio.sys	CloneCD	CVE-2009-0824 <a href="#">✗</a>

2020	Trickbot <a href="#">✗</a>	rwdrv.sys	RWEverything <a href="#">✗</a>	N/A
------	----------------------------	-----------	--------------------------------	-----

# I want more!!! Okay 😊

- Please visit : <https://www.loldrivers.io/>

Tag↕	SHA256↕	Category↕	Created↕
prokiller64.sys	0440ef40c46fdd2b5d86e7feef8577a8591de862cfd7928cdbcc8f47b8fa3ffc	Malicious	2023-05-07
amigendrv64.sys	09043c51719d4bf6405c9a7a292bb9bb3bcc782f639b708ddcc4eedb5e5c9ce9	Vulnerable driver	2023-05-06
eneio64.sys	38c18db050b0b2b07f657c03db1c9595febae0319c746c3eede677e21cd238b0	Vulnerable driver	2023-05-06
Se64a.sys	6cb51ae871fbd5d07c5aad6ff8eea43d34063089528603ca9ceb8b4f52f68ddc	Vulnerable driver	2023-01-09
iQVW64.SYS	19bf0d0f55d2ad33ef2d105520bde8fb4286f00e9d7a721e3c9587b9408a0775	Vulnerable driver	2023-05-06
d4.sys	823da894b2c73ffcd39e77366b6f1abf0ae9604d9b20140a54e6d55053aadeba	Vulnerable driver	2023-01-09
vmdrv.sys	5c0b429e5935814457934fa9c10ac7a88e19068fa1bd152879e4e9b89c103921	Vulnerable driver	2023-05-06
d3.sys	36875562e747136313ec5db58174e5fab870997a054ca8d3987d181599c7db6a	Vulnerable driver	2023-01-09
netfilter2.sys	f1718a005232d1261894b798a60c73d971416359b70d0e545d7e7a40ed742b71	Vulnerable drivers	2023-07-22
nvflash.sys	9368e51ec98e2ad20893a5fc21e6a8b20c5bee158d5c49ca58649cff84db9d68	Vulnerable drivers	2023-07-22
CITMDRV_IA64.sys	1c8dfa14888bb58848b4792fb1d8a921976a9463be8334cff45cc96f1276049a	Vulnerable driver	2023-01-09

## Description

Signed POORTRY Samples

- UUID: 6fe10a55-7fb8-4a9d-9ebc-1b27b6e5b833
- Created: 2023-05-07
- Author: Guus Verbeek
- Acknowledgement: |

Download

 This download link contains the malicious driver!

## Commands

```
sc.exe create prokiller64.sys binPath=C:\windows\temp\prokiller64.sys type=kernel && sc.exe start prokiller64.sys
```

NOT SET

## Use Case

Elevate privileges

## Privileges

kernel


## Operating System

Windows 10


## Detections

YARA 

► Expand

Sigma 

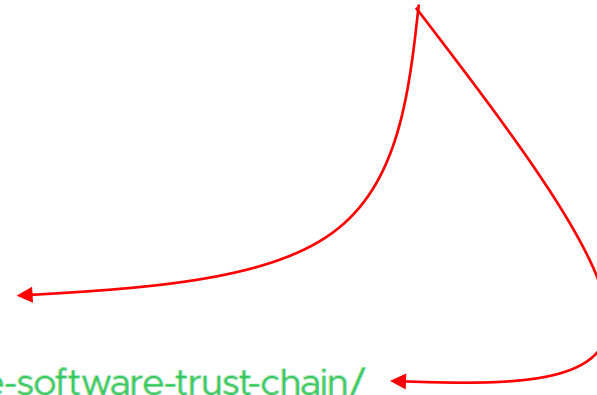
► Expand

Sysmon 

► Expand

## Resources

- <https://www.mandiant.com/resources/blog/hunting-attestation-signed-malware>
- <https://news.sophos.com/en-us/2022/12/13/signed-driver-malware-moves-up-the-software-trust-chain/>







# Lessons learned

Solutions and Mitigation



# What does it mean?

- The development of sensitive software:
  - Is sometime not considered with enough caution.
  - Is based on old technologies.
  - Is based on old designs ...
- But the most important is that the problem remains.
  - **Backward compatibility** for the **best** and for the **worst**.
  - There are plenty of device drivers **ignoring basics** about **security**.
    - Even worst, it is about providing ways to bypass the security.
    - Not because it is necessary, but because of incompetency.
      - They ignore most of the time that other (but more complex) solutions exist.

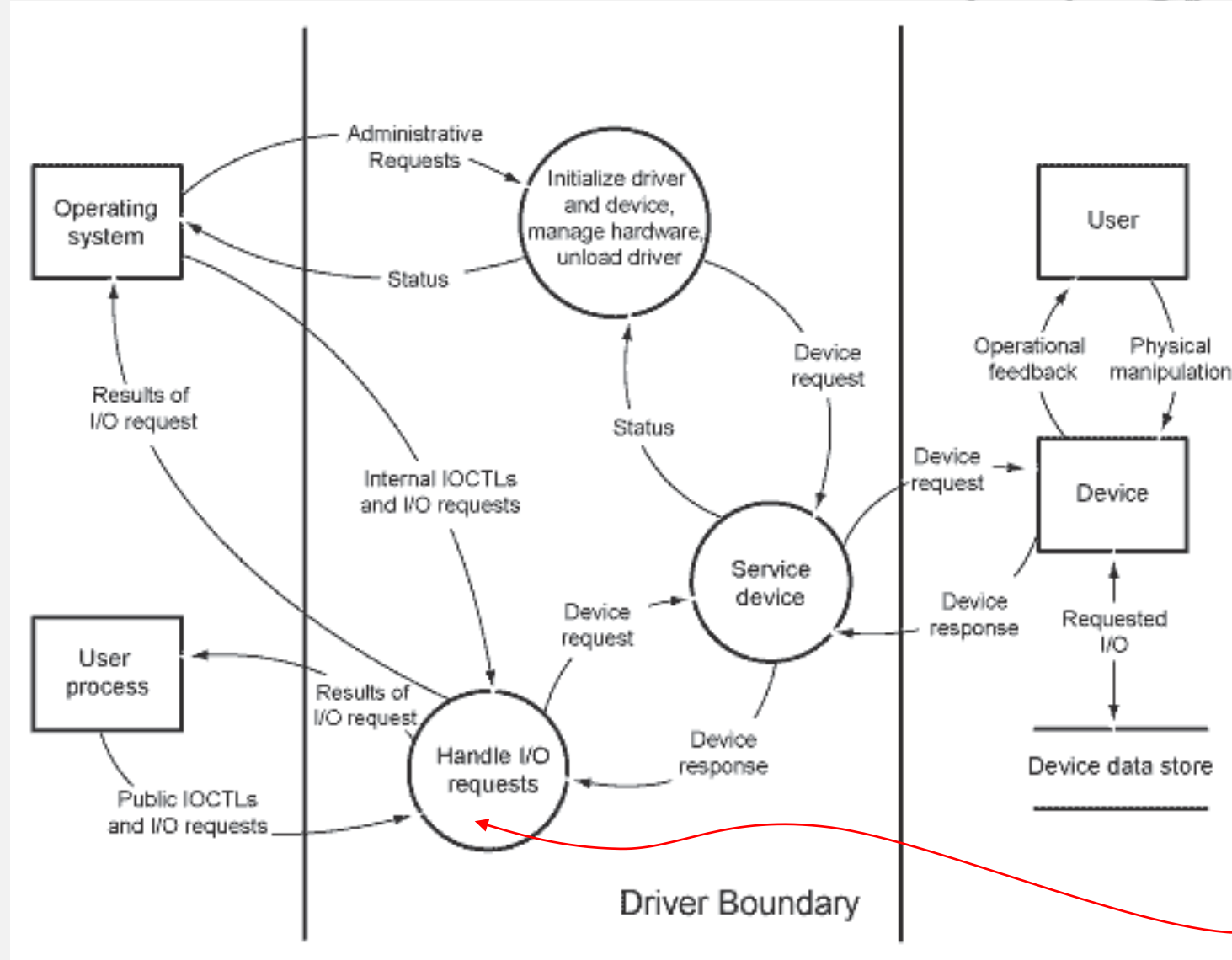
# What to do at technical level?

- Microsoft recommends to driver writers and architects to make threat modeling an integral part of the design process for any driver.
  - There is a [Driver Security Checklist](#).
- Any successful product is a target.
  - “(...) *assume that sometime, somewhere, someone will try to use your driver to compromise system security*”, [Microsoft](#).
  - It may involve to:
    - Remove your driver or use you driver to remove another driver/security.
    - Attack the full system by allowing elevation of privileges.
      - Abuse ring 0 reserved feature (loading driver, memory access, special operation reserved, ...)
      - Change the access token of a process to run with admin privileges.

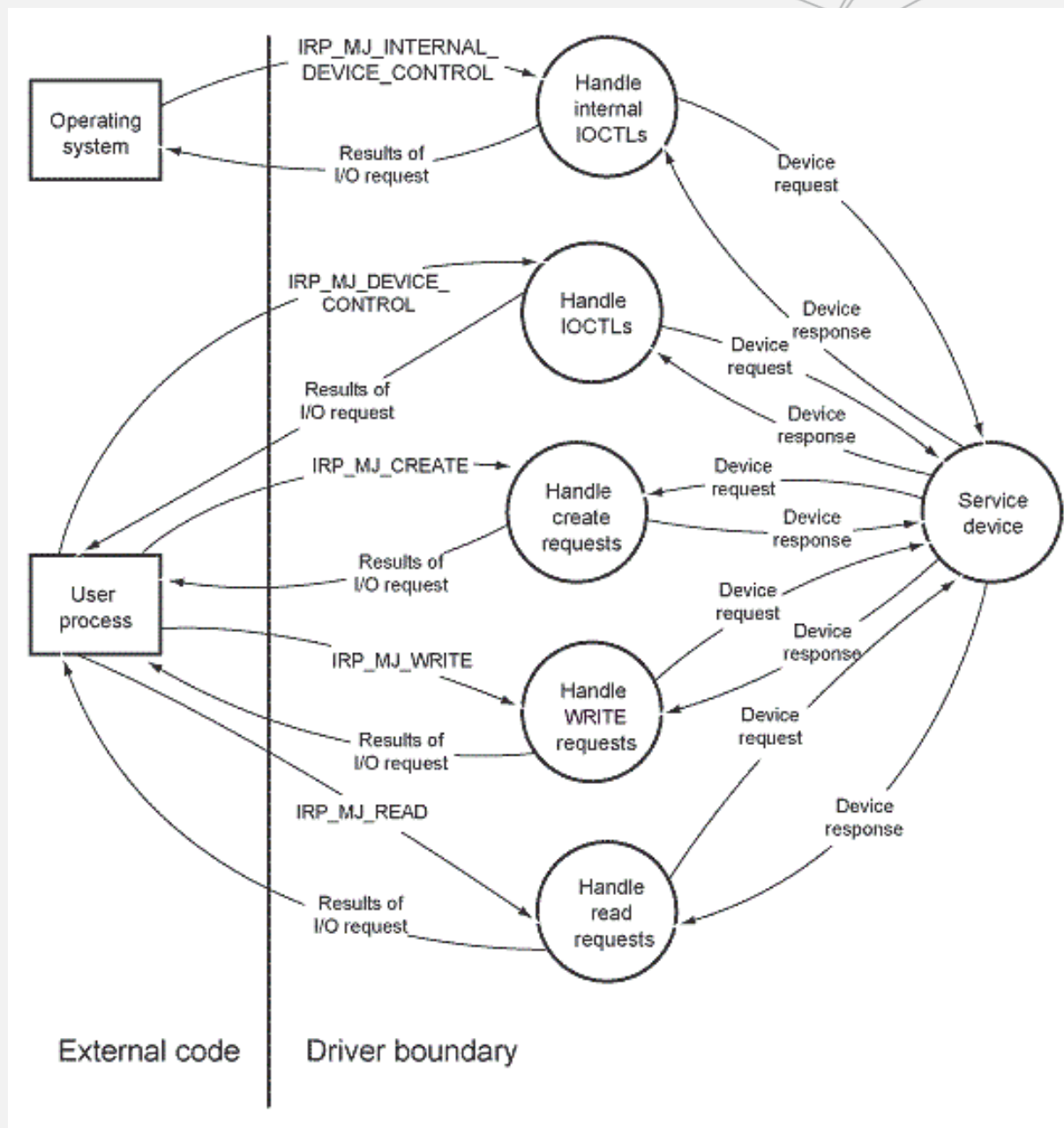
# Understanding the input of a driver

- Possible data sources include:
  - IRP\_MJ\_XXX requests that the driver handles
  - IOCTLs that the driver defines or handles
  - APIs that the driver calls
  - Callback routines
  - Any other interfaces that the driver exposes
    - Mini filter drivers, windows filtering platform (firewall), ...
  - Files that the driver reads or writes, including those used during installation
  - Registry keys that the driver reads or writes
  - Configuration property pages, and any other information provided by the user that the driver consumes

- For example (taken from [Microsoft](#)):



This circle could be extended in a more detailed way...





# STRIDE

- The acronym **STRIDE** describes six categories of threats to software.
  - **S**poofing
  - **T**ampering
  - **R**epudiation
  - **I**nformation disclosure
  - **D**enial of service
  - **E**levation of privilege
- For each potential vulnerable point of the driver:
  - Determine the types of attack that might match.
  - Consider a scenario for a possible attack.

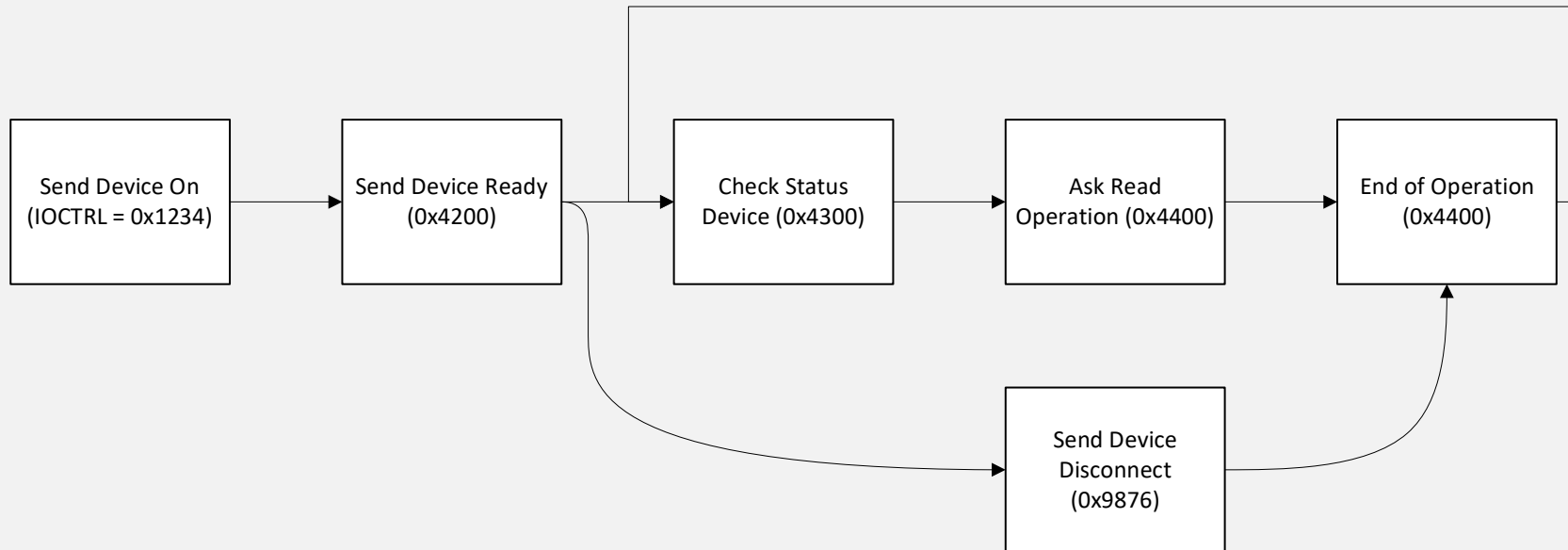
# STRIDE

- The goal is to create one or more attack scenarios for each plausible threat.

Vulnerable point	Potential threat (STRIDE)	Scenario
IOCTL 0xFFFF	DOS	Send IOCTL 0xFFFF with non allocated input buffer

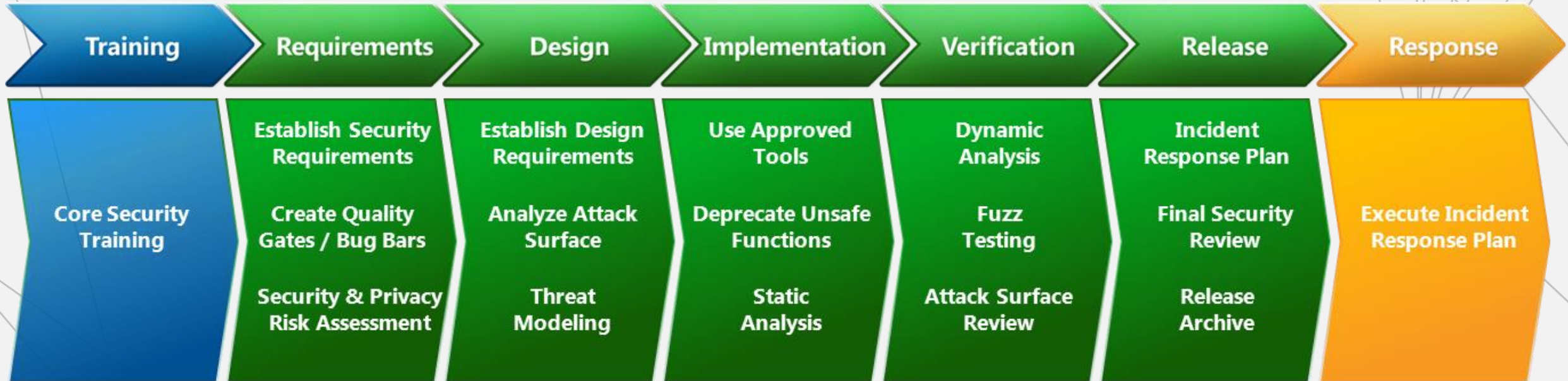
- Some types of attacks depend on a sequence of events.
  - It is possible combine different events.
  - For instance, send many IOCTLs with different information inside.

# STRIDE



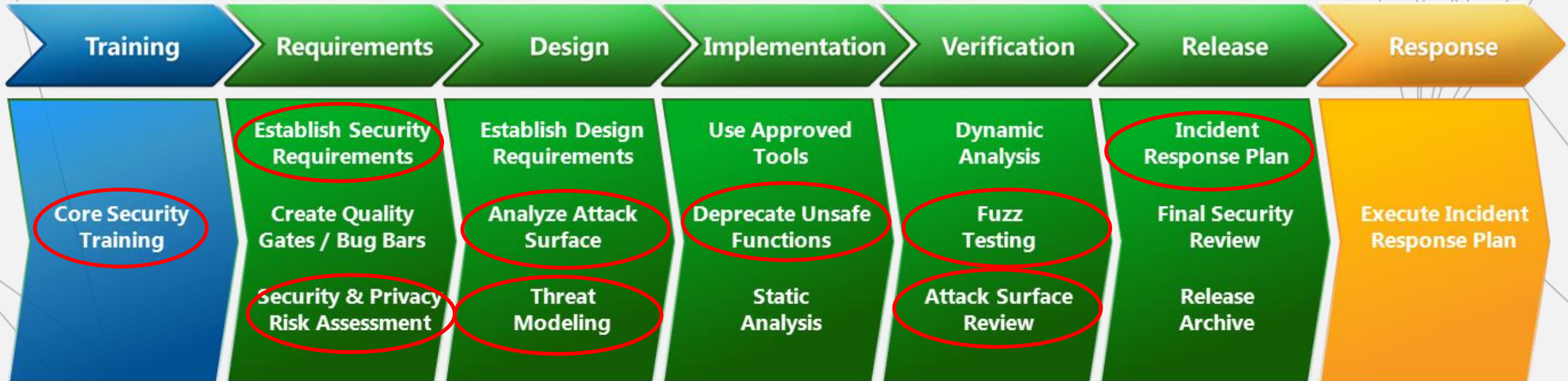
# What to do at a development level?

- [Microsoft Security Development Lifecycle \(SDL\)](#) [[White Paper](#)]



# What to do at a development level?

- [Microsoft Security Development Lifecycle \(SDL\) \[White Paper\]](#)





# What to do at an administrative level?

- We need to learn how to “live” with the threat.
- Microsoft aims to enhance the security of its system.
  - [Mitigate threats by using Windows 10 security features](#)
  - Especially:
    - Data Execution Prevention (DEP)
    - Structured Exception Handling Overwrite Protection (SEHOP)
    - Address Space Layout Randomization (ASLR)
    - Control Flow Guard (CFG)
    - Since Win10: mitigations to protect against memory exploits (among other stuffs)
      - Protected Processes
      - Kernel pool protections
      - ...

# What to do at an administrative level?

- Kernel pool protections
  - There are many mitigations that have been added over time, such as process quota pointer encoding, lookaside, delay free, pool page cookies, and PoolIndex bounds checks.
  - Since Windows 10, there are multiple other "pool hardening" protections.
    - Integrity checks (Patchguard / HyperGuard – HVCI).
    - Font parsing in [AppContainer](#).
      - Credential, Device, File, Network, Process, and Window isolation.
    - Disabling of NT Virtual DOS Machine (NTVDM) isolation.
      - Running 16-bit applications, avoid associated exploits especially the protection against Null dereference.
    - Supervisor Mode Execution Prevention (SMEP)
    - Memory reservations.
    - [Kernel Data Protection](#) (KDP).
      - Based on the [MmProtectDriverSection](#) function.

# What to do at an administrative level?

- **Implement Security Best Practices for the hardening of Windows 10/11** (and Windows Server)
- Relevant information can be found here (documents are in English despite the title 😊)
  - [SiSyPHuS Win 10: Empfehlung zur Härtung von Windows 10 mit Bordmitteln](#) (BSI)
  - [Security Baselines for current Windows 10 /11 and for Windows Server](#) (Microsoft)
  - [CIS Benchmark for Windows 10](#) (CIS)
  - [CIS Benchmark for Windows Server](#) (CIS)



Bundesamt  
für Sicherheit in der  
Informationstechnik



# What to do at an organizational level?

- Use WDAC
  - The list of drivers allowed or refused on the system.
  - Windows Defender Application Control (WDAC).
    - Allow organizations to control which drivers and applications are allowed to run.
  - [Microsoft recommended driver block rules](#) about drivers:
    - Known security vulnerabilities that can be exploited by attackers to elevate privileges in the Windows kernel
    - Malicious behaviors or certificates used to sign malware
    - The list could be extended 😊...
  - But only for a few number highly secured human managed devices.
    - Due to a lack of [administration/configuration/deployment] capabilities.
  - Hypervisor-protected code integrity (HVCI) & Smart App Control
    - Avoid to temper with the system and check what is runnable on the system.

```
<Signer ID="ID_SIGNER_VERISIGN_AMD" Name="VeriSign Class 3 Code Signing 2010 CA">
  <CertRoot Type="TBS" Value="4843A82ED3B1F2BFBEE9671960E1940C942F688D" />
  <CertPublisher Value="Advanced Micro Devices, Inc." />
  <FileAttribRef RuleID="ID_FILEATTRIB_AMD_RYZEN" />
</Signer>
<Signer ID="ID_SIGNER_VERISIGN_TG_SOFT" Name="VeriSign Class 3 Code Signing 2010 CA">
  <CertRoot Type="TBS" Value="4843A82ED3B1F2BFBEE9671960E1940C942F688D" />
  <CertPublisher Value="TG Soft S.a.s. Di Tonello Gianfranco e C." />
  <FileAttribRef RuleID="ID_FILEATTRIB_VIRAGT" />
  <FileAttribRef RuleID="ID_FILEATTRIB_VIRAGT64" />
</Signer>
<Signer ID="ID_SIGNER_GLOBALSIGN_TG_SOFT" Name="GlobalSign CodeSigning CA - G3">
  <CertRoot Type="TBS" Value="F478F0E790D5C8EC6056A3AB2567404A991D2837" />
  <CertPublisher Value="TG Soft di Tonello Gianfranco ed Enrico S.r.l." />
  <FileAttribRef RuleID="ID_FILEATTRIB_VIRAGT" />
  <FileAttribRef RuleID="ID_FILEATTRIB_VIRAGT64" />
</Signer>
<Signer ID="ID_SIGNER_HP" Name="DigiCert SHA2 Assured ID Code Signing CA">
  <CertRoot Type="TBS" Value="E767799478F64A34B3F53FF3BB9057FE1768F4AB178041B0DCC0FF1E210CBA65" />
  <CertPublisher Value="HP Inc." />
  <FileAttribRef RuleID="ID_FILEATTRIB_HPPORTIOX64" />
  <FileAttribRef RuleID="ID_FILEATTRIB_WINRING0" />
</Signer>
<Signer ID="ID_SIGNER_GETAC" Name="Symantec Class 3 Extended Validation Code Signing CA - G2">
  <CertRoot Type="TBS" Value="B3C925B4048C3F7C444D248A2B101186B57CBA39596EB5DCE0E17A4EE4B32F19" />
  <CertPublisher Value="Getac Technology Corp." />
  <FileAttribRef RuleID="ID_FILEATTRIB_MTCBSV64" />
</Signer>
<Signer ID="ID_SIGNER_GLOBALSIGN_CHEAT_ENGINE" Name="GlobalSign CA Cheat Engine Publisher">
  <CertRoot Type="TBS" Value="BD1765C56594221373893EF26D97F88C144FB0E5A0111215B45D7239C3444DF7" />
  <CertPublisher Value="Cheat Engine" />
</Signer>
<Signer ID="ID_SIGNER_GLOBALSIGN_G2_CHEAT_ENGINE" Name="GlobalSign CodeSigning CA - G2">
  <CertRoot Type="TBS" Value="589A7D4DF869395601BA7538A65AFAE8C4616385" />
  <CertPublisher Value="Cheat Engine" />
</Signer>
<Signer ID="ID_SIGNER_PHYSMEM" Name="GlobalSign CodeSigning CA - G2">
  <CertRoot Type="TBS" Value="589A7D4DF869395601BA7538A65AFAE8C4616385" />
  <CertPublisher Value="Hilscher Gesellschaft fuer Systemautomation mbH" />
  <FileAttribRef RuleID="ID_FILEATTRIB_PHYSMEM" />
</Signer>
```



# What to do at organizational level?

- Manage devices in your organization.
  - Not all **fancy devices** have a good reason to be there.
  - **Check** carefully which **device drivers** are **installed** on your system.
    - Even if it is not listed by Microsoft in the [recommended driver block rules](#), some driver may be vulnerable.
    - We are far to have check all device driver up to now.
  - Only **authorize** a **subset** of **certificates** to sign driver.
    - Trusted vendors & organizations.
    - Those you can trust because you checked the quality of their software.
  - **Default driver** shipped with **Windows** “**out of the box**” are more than enough.
    - Most of the time they support any kind of device.
    - Not always at 100% of their capabilities, but 90% is enough for day-to-day life.



# Going further



- With project **SiSyPHuS** the **Federal Office for Information Security** (BSI) analyzes several parts of Windows 10 which might have an impact on the overall system security.
  - The study is being conducted by **ERNW GmbH** on behalf of the BSI.
  - It is a reverse engineering study of Windows operating system.
    - Providing also a review of the security.
    - And how to configure / use Windows in a secure way.
  - Many topic are relevant in the context of device drivers.
    - Especially to understand how it does work behind the stage.
    - Especially to enhance the security of Windows.
    - Especially to get specific logs about device driver events.

# Going

- With pro (BSI) and impact c
- The st
- It is a
  - Pro
  - An
- Many
  - Esp
  - Esp
  - Esp

## Work Packages

- > Analysis of Windows 10 - General OS Structure
- > TPM Vulnerability CVE-2017-15361
- > Telemetry Service
- > TPM and "UEFI SecureBoot"
- > Virtualization Based Security
- > Device Guard
- > PowerShell and Windows Script Host
- > Logging Guideline
- > Hardening Guideline
- > GPOs for Guidelines
- > Monitoring System Modifications
- > Universal Windows Apps and Windows Information Protection
- > Secure Boot Configuration Policy
- > Telemetry Monitoring Framework
- > ETW Monitoring Methodology (AFUNKT)
- Windows Application Compatibility Infrastructure
- Driver Management



Security

ave an

BSI.

# Conclusion

Times up...

# Conclusion

- First of all, finding vulnerabilities in drivers is not such a hard job.
  - There are low hanging fruits.
  - Any driver which provides a “fancy” but “dubious” feature.
    - Overclock, bypass security, direct interface with ...
  - Any driver which exists since Windows XP (or before 😊) keeping similar features.
    - It is not a 100% chance of success ... but it is not low 😊.
    - Super power of the copy/past.
  - Any driver from hardware manufacturer, always good surprises...
    - But also some antivirus vendors ... 😊 ... from time to time ...
    - *Avast 21.5 release in June 2021* [\[REF\]](#).



# Conclusion

- Driver vulnerabilities are not close to disappear.
  - Old/legacy/deprecated drivers are still useable.
  - And they are still in used.
  - And they will be used in the future with a high probability
- Security of code based on legacy/deprecated solutions.
  - Always the same story ... always the same codes ...
  - SDL should be seriously considered by hardware vendors.
    - Especially on a training side (sharing knowledge, anticipating the issues).
    - And customers who should complain about such low quality of software.
- Necessity to remediate it at different levels
  - Technical, management, policies ...

# Thank you for your attention



b david@ernw.de



@ernw



www.ernw.de



www.insinuator.net

