REVERSE ENGINEERING ANALYSIS OF THE NDIS 6.* STACK

DÉSIRÉE SACHER

PRESENTATION DIPLOMA THESIS, ZHAW

AGENDA

OVERVIEW

- Thesis Objectives
- Common Ground
- Thesis Approach
- Analysis Results
- Reflection
- Outlook

GOAL

THESIS OBJEKTIVES

- Analysing network stack of Windows operating systems that use NDIS 6.*
- Identify network connection structures in memory and document state changes

MEMORY FORENSICS

COMMON GROUND (I)

- Reverse Engineering → «Extracting knowledge or design information from anything manmade» (Wikipedia)
- Memory Forensics → Used to analyse what was running on machines at moment of image creation
- Kernel Debugger Datablock (KDBG) → Contains information about current processor, contains KPRCB
- Processor Control Block (KPRCB) → Contains CPU step information and pointer to thread object of current thread

MICROSOFT DEBUGGING

COMMON GROUND (II)

- PDB Files → Microsoft «program database» files used for debugging, includes function addresses, global variable names and addresses, parameters, and local variable names and offset addresses, etc.
- Pool Tag → Four byte character associated with dynamically allocated part of memory pool in Windows
- Memory Descriptor Lists (MDL) → List to describe physical page for a virtual memory buffer, when pages are bigger than a physical page, or pages are discontinuous

TOOLS

COMMON GROUND (III)

- Volatility → Most used open source forensic tool, relies on «static» profiles
- Rekall → Spin off of Volatility, relies on PDB file information
- SIFT Workstation → Virtual machine based on Ubuntu, preinstalled with forensic tool suites

TOOLS – WHY?

COMMON GROUND (IV)

- Rekall
  - Interactive Python shell
  - PDB profile information
  - Same amount of features as Volatility
  - Assistance for struct analysis with specific plugins
- SIFT Workstation
  - Installed virtual machine with all analysis tools

IDEA FOR THEORETICAL APPROACH

THESIS APPROACH

- Analyse UNIX socket structs to compare them to structures found in NDIS stack
- Answer general questions to understand memory management better
  - Identify how IP addresses are stored in memory
  - Find information about pool tags also used in UNIX
  - Compare netstat in Windows and Linux to find similarities

COMPARISON SOCKET FUNCTION

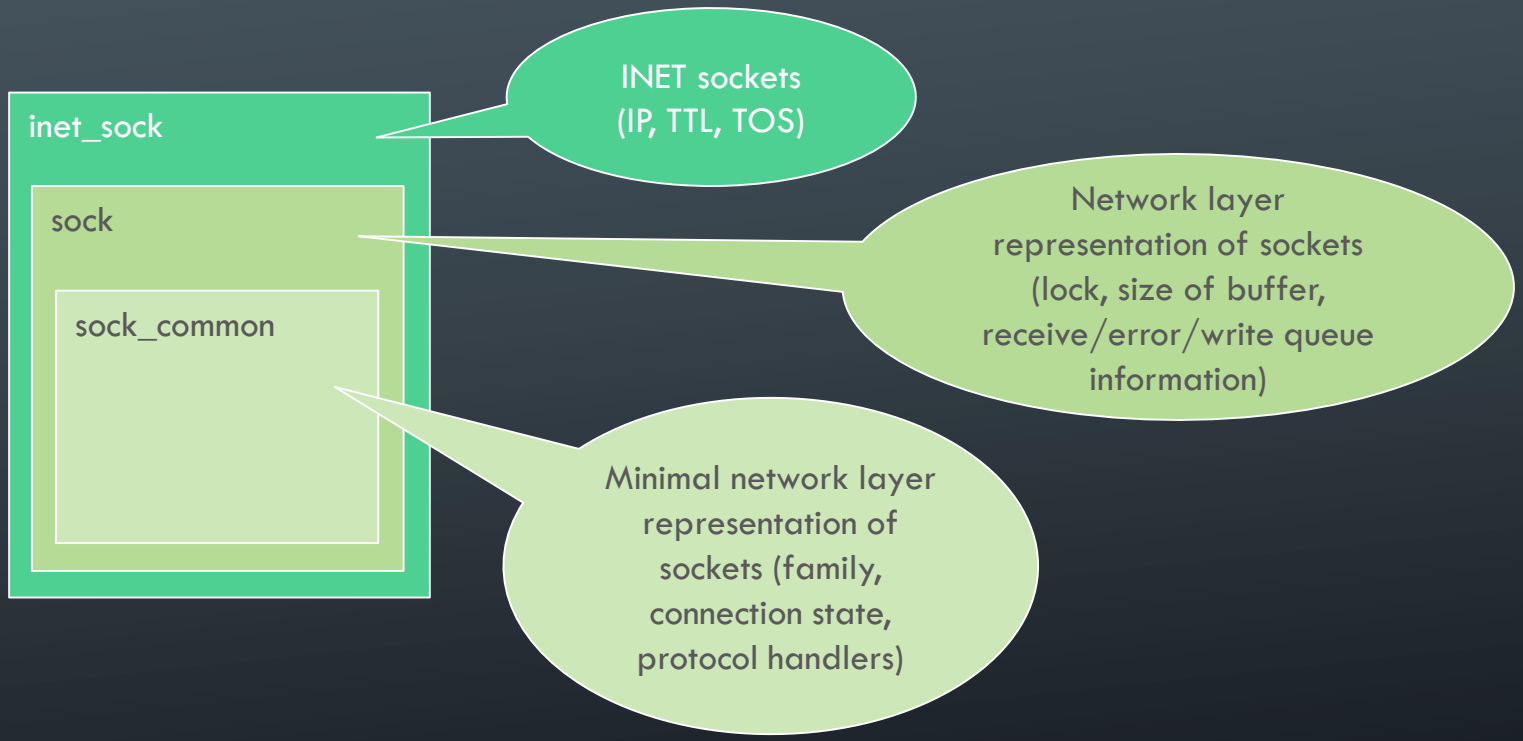
ANALYSIS RESULTS (I)

UNIX Argument	UNIX Argument Description [1]	Windows Argument	Windows Argument Description
int domain	Nature of communication including address format *Example: PF_INET(IPv4) or PF_INET6(IPv6)*	int af	Address family *Example: AF_INET(IPv4), AF_NETBIOS, AF_INET6(IPv6)*
int type	Type of socket including communication characteristics *Example: SOCK_STREAM(stream) or SOCK_DGRAM(datagram)*	int type	Type of socket *Example: SOCK_STREAM(stream), SOCK_DGRAM(datagram), RAW*
int protocol	Protocol type *Example: 0 (any), getprotobyname()(TCP/UDP)*	int protocol	Protocol type *Example: ICMP, TCP, UDP*

[1] Source: "Advanced Programming in the UNIX Environment", Section 16.2 Socket Descriptors
Thesis source: Table 3: Comparison socket UNIX/Windows arguments

INET_SOCK, SOCK, AND SOCK_COMMON RELATION

ANALYSIS RESULTS (II)



COMPARISON SOCK_COMMON MEMBERS TO NDIS

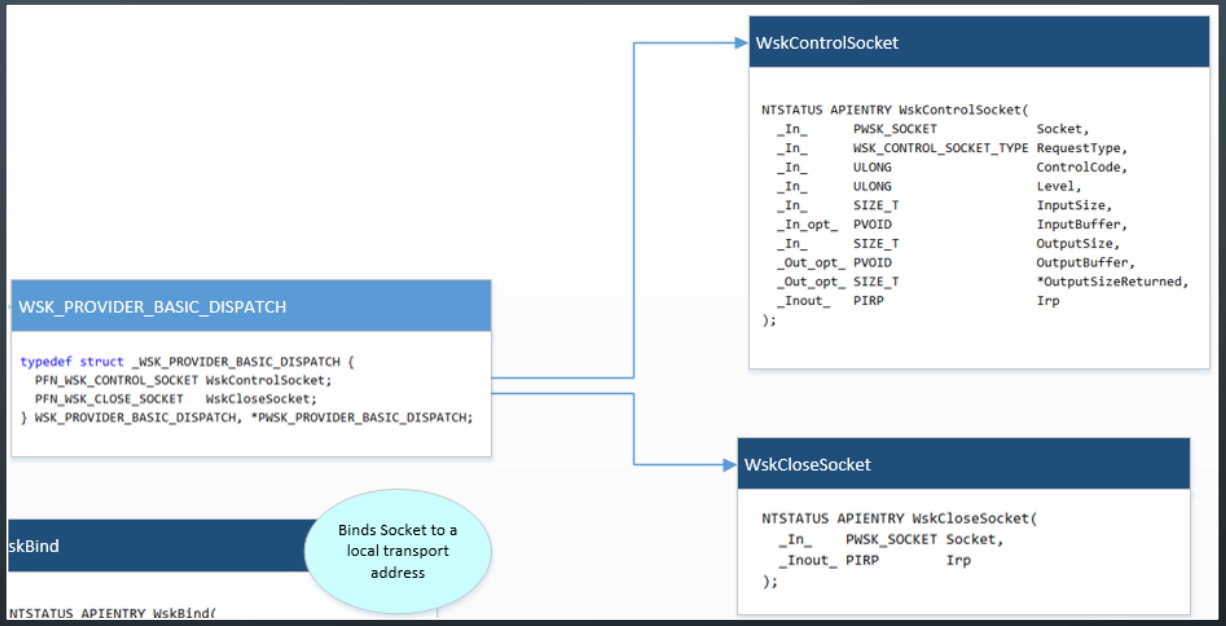
ANALYSIS RESULTS (III)

Socket Member	Description socket	NDIS Member
skc_daddr	Foreign IPv4 address	WskSocketConnect.RemoteAddress
skc_rcv_saddr	Bound local IPv4 address	WskSocketConnect.LocalAddress
skc_dport	Placeholder for inet_dport/tw_dport (destination port)	MiniportSendNetBufferLists.PortNumber
skc_num	Placeholder for inet_num/tw_num (local port)	NdisSendNetBufferLists.PortNumber
skc_state	Connection state	Wsk*.Irp

Thesis source: Table 4: sock_common member comparison with suitable representations in NDIS

PROVIDER DISPATCH TABLE

ANALYSIS RESULTS (IV)



Thesis source: Figure 18: Extract of NDIS Struct Relation Map - WSK_PROVIDER_BASIC_DISPATCH

SK_BUFF STRUCTURE

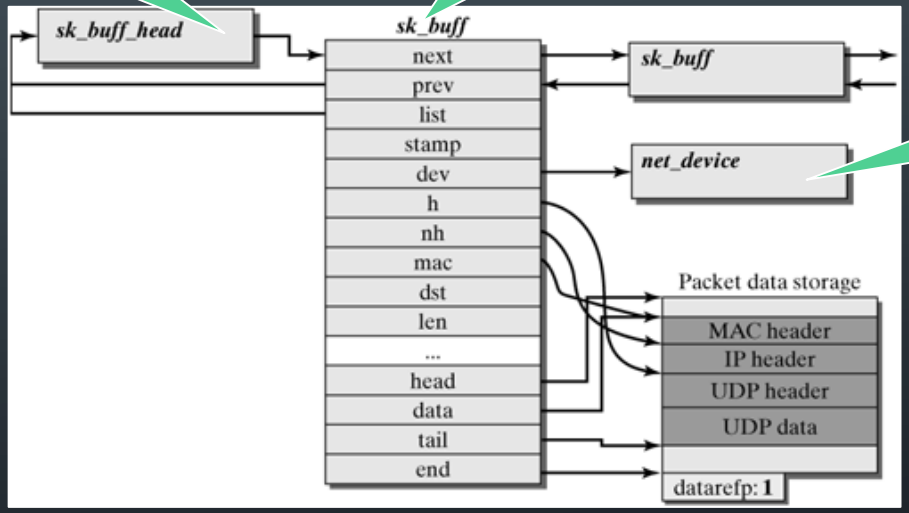
ANALYSIS RESULTS (V)

Represents head of queue

Structure for buffered packets

Also used in functions sk_receive_queue, sk_write_queue, sk_error_queue

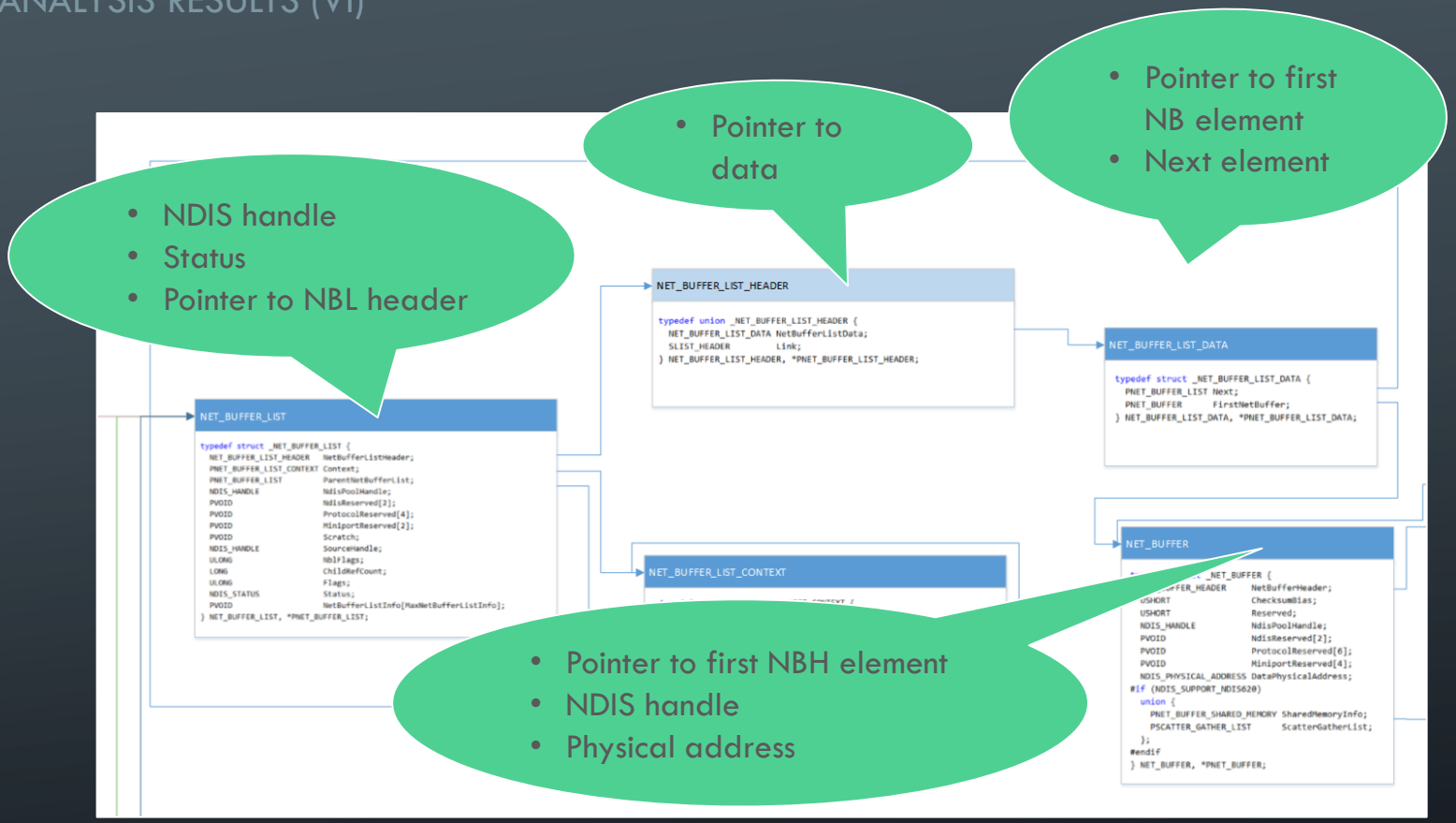
Device Configuration



Source: http://flylib.com/books/en/3.475.1.29/1/

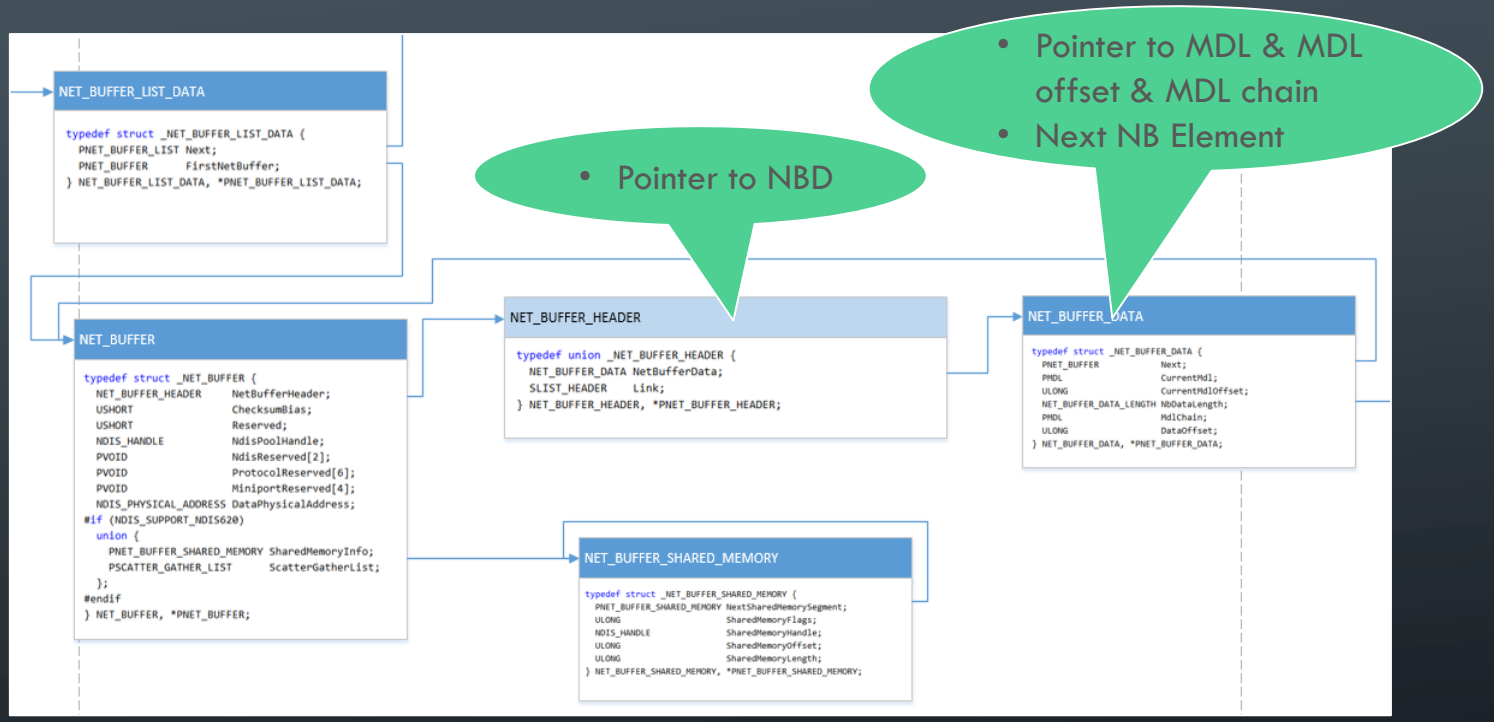
NET_BUFFER STRUCTURE I

ANALYSIS RESULTS (VI)



NET_BUFFER STRUCTURE II

ANALYSIS RESULTS (VII)



PHYSICAL DEVICE LINUX (NET_DEVICE)

ANALYSIS RESULTS (VIII)

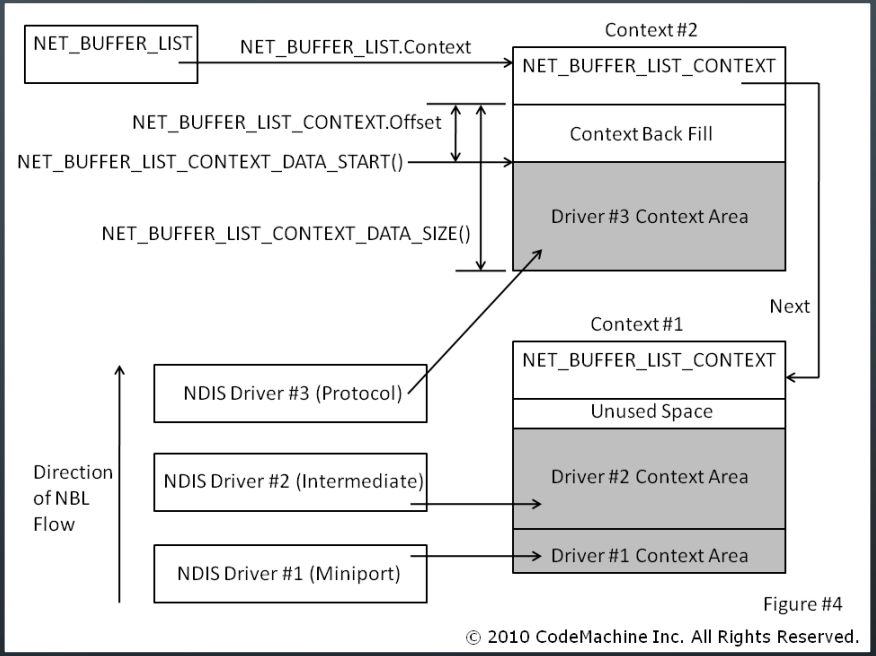
- Struct that contains
  - Physical device connection settings
  - Shared memory reference
  - Network statistic parameters
  - Interface flags
  - MTU value

```
1349 /**
1350  *      struct net_device - The DEVICE structure.
1351  *              Actually, this whole structure is a big mistake.  It mixes I/O
1352  *              data with strictly "high-level" data, and it has to know about
1353  *              almost every data structure used in the INET module.
1354  *
1355  *      @name:  This is the first field of the "visible" part of this structure
1356  *              (i.e. as seen by users in the "Space.c" file).  It is the name
1357  *              of the interface.
1358  *
1359  *      @name_hlist:    Device name hash chain, please keep it close to name[]
1360  *      @ifalias:       SNMP alias
1361  *      @mem_end:       Shared memory end
1362  *      @mem_start:     Shared memory start
1363  *      @base_addr:     Device I/O address
1364  *      @irq:           Device IRQ number
1365  *
1366  *      @carrier_changes:       Stats to monitor carrier on<->off transitions
1367  *
1368  *      @state:         Generic network queuing layer state, see netdev_state_t
1369  *      @dev_list:      The global list of network devices
1370  *      @napi_list:     List entry, that is used for polling napi devices
1371  *      @unreg_list:    List entry, that is used, when we are unregistering the
1372  *                      device, see the function unregister_netdev
1373  *      @close_list:    List entry, that is used, when we are closing the device
1374  *
1375  *      @adj_list:      Directly linked devices, like slaves for bonding
1376  *      @all_adj_list:  All linked devices, *including* neighbours
1377  *      @features:      Currently active device features
1378  *      @hw_features:   User-changeable features
1379  *
1380  *      @wanted_features:       User-requested features
1381  *      @vlan_features:         Mask of features inheritable by VLAN devices
1382  *
1383  *      @hw_enc_features:       Mask of features inherited by encapsulating devices
1384  *                              This field indicates what encapsulation
1385  *                              offloads the hardware is capable of doing,
1386  *                              and drivers will need to set them appropriately.
1387  *
1388  *      @mpls_features: Mask of features inheritable by MPLS
1389  *
1390  *      @ifindex:       interface index
1391  *      @group:         The group, that the device belongs to
1392  *
```

Source: http://lxr.free-electrons.com/source/include/linux/netdevice.h#L1560

PHYSICAL DEVICE CONFIGURATION I

ANALYSIS RESULTS (IX)



Thesis source: Figure 24: NDIS driver association within context areas when packets are received
Source: http://codemachine.com/article_ndis6nbls.html

PHYSICAL DEVICE CONFIGURATION II

ANALYSIS RESULTS (X)

C++

```
typedef union _NDIS_MINIPORT_ADAPTER_ATTRIBUTES {
  NDIS_OBJECT_HEADER                                Header;
  NDIS_MINIPORT_ADD_DEVICE_REGISTRATION_ATTRIBUTES AddDeviceRegistrationAttributes;
  NDIS_MINIPORT_ADAPTER_REGISTRATION_ATTRIBUTES    RegistrationAttributes;
  NDIS_MINIPORT_ADAPTER_GENERAL_ATTRIBUTES         GeneralAttributes;
  NDIS_MINIPORT_ADAPTER_OFFLOAD_ATTRIBUTES         OffloadAttributes;
  NDIS_MINIPORT_ADAPTER_NATIVE_802_11_ATTRIBUTES   Native_802_11_Attributes;
#if  (NDIS_SUPPORT_NDIS61)
  NDIS_MINIPORT_ADAPTER_HARDWARE_ASSIST_ATTRIBUTES HardwareAssistAttributes;
#endif
#if (NDIS_SUPPORT_NDIS630)
  NDIS_MINIPORT_ADAPTER_NDK_ATTRIBUTES             NDKAttributes;
#endif
} NDIS_MINIPORT_ADAPTER_ATTRIBUTES, *PNDIS_MINIPORT_ADAPTER_ATTRIBUTES;
```

Thesis source: Figure 27: NDIS_MINIPORT_ADAPTER_ATTRIBUTES union structure

NETSTAT DIFFERENCES

ANALYSIS RESULTS (XI)

Linux	Windows
Reads /proc/net/tcp	Stored in Hash Table, marked TcpE, TcpL, UdpA
netscan carves network connection structures	netscan builds several pool scanners and looks for pool tags & simple checks

DISPLAY OF IP ADDRESS IN MEMORY I

ANALYSIS RESULTS (XII)

Little endian pointer to offset address of AddressInfo struct

```
Rule: r1
Owner: (Unknown Kernel Memory)
  Offset                          Hex                                Data
---------- ------------------------ ------------------- ----------------
0x8547ed64 54 63 70 45 00 00 00 00 02 00 00 00 d0 60 ca 84  TcpE…….....`..
0x8547ed74 80 c4 b8 84 38 ca 4f 84 70 9e b7 84 70 9e b7 84  ….8.O.p…p…
0x8547ed84 b4 33 60 b9 60 66 b7 84 60 66 b7 84 b9 33 60 b9  .3`.`f..`f…3`.
0x8547ed94 28 ba 5a 87 58 9b 65 84 04 00 00 00 c0 9d 00 50  (.Z.X.e…….....P
```

Thesis source: Figure 44: Finding the IP address in hex in Rekall

Pointer to offset address of _IN_ADDR Remote struct

```
[1] memorydumpMSSite151002.dmp 21:27:03> analyse_struct 0x844fca38
--------------------------------------→ analyse_struct(0x844fca38)
2016-04-26 21:41:42,581:DEBUG:rekall.1:Running plugin (analyse_struct) with args
((2219821624,)) kwargs ({})
  Offset    Content
----------  -------
       0x0  Data:0x84b95230
       0x4  Data:0xe0000001
       0x8  Data:0x844fcac8
       0xc  Data:0x22
      0x7c Data:0x0

                                    Out<3> Plugin: analyse_struct
```

Thesis source: Figure 45: analyse_struct display of offset address of _ADDRINFO struct in Rekall

DISPLAY OF IP ADDRESS IN MEMORY II

ANALYSIS RESULTS (XIII)

```
[1] memorydumpMSSite151002.dmp 21:42:43> analyse_struct 0x844fcac8
-------------------------------------→ analyse_struct(0x844fcac8)
2016-04-26 21:46:33,434:DEBUG:rekall.1:Running plugin (analyse_struct) with args
((2219821768,)) kwargs ({})
  Offset   Content
---------- -------
       0x0 Data:0x2315a3d9
       0x4 Data:0x0
       0x8 Data:0x844fc7d8
                                  Out<4> Plugin: analyse_struct
```

Address stored in little endian

Thesis source: Figure 46: analyse_struct display of offset address of the Remote _IN_ADDR struct in Rekall

```
2315a3d9 => 0xd9 (=217).0xa3(=163).0x15(=21).0x23(=35)
```

Thesis Source: Figure 47: Calculation of IP address from hex value

POOL TAGS IN LINUX

ANALYSIS RESULTS (XIV)

- Is there anything like pool tags in Linux???

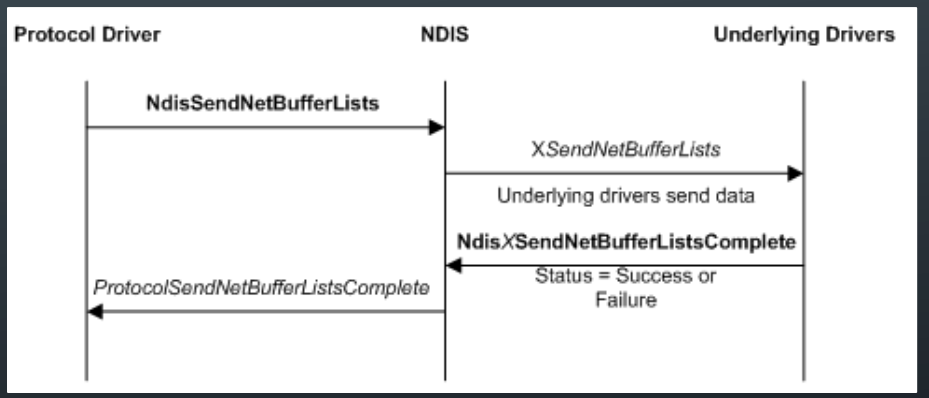
No!

→ Correct profile of machine is indispensable

PRACTICAL ANALYSIS I

ANALYSIS RESULTS (XV)

- Guessed pool tag TNbl (“TCP Send NetBufferLists”) to be associated with function NdisSendNetBufferLists



Thesis source: Figure 25: Sending data from a protocol driver
Source: https://msdn.microsoft.com/en-us/library/windows/hardware/ff570753%28v=vs.85%29.aspx

PRACTICAL ANALYSIS II

ANALYSIS RESULTS (VXI)

- Found values being pushed to stack in disassembly
- Possible matching of function

```
C++

VOID NdisSendNetBufferLists(
  _In_ NDIS_HANDLE      NdisBindingHandle,
  _In_ PNET_BUFFER_LIST NetBufferLists,
  _In_ NDIS_PORT_NUMBER PortNumber,
  _In_ ULONG            SendFlags
);
```

Source Thesis: Figure 51: NdisSendNetBufferLists function from MSDN

```
0x8722891c    0x106 ff35b43c3187      PUSH DWORD [0x87313cb4]     0x0 tcpip!Microsoft_Windows_TCPIPHandle + 0x4
0x87228922    0x10c ff35b03c3187      PUSH DWORD [0x87313cb0]     0x2f tcpip!Microsoft_Windows_TCPIPHandle
0x87228928    0x112 e8072a0900        CALL 0x872bb334             tcpip!TcpipTransferActivityIDToNBL + 0x37
0x8722892d    0x117 ff35e4a73187      PUSH DWORD [0x8731a7e4]     0x84385840 tcpip!TcpSendRequestPool
0x87228933    0x11d e8de000900        CALL 0x872b8a16             tcpip!InetInspectRemoteDisconnect + 0xc
0x87228938    0x122 e962ffffff        JMP 0x8722889f              tcpip!TcpStartSendModule + 0x89
0x8722893d    0x127 68544e626c        PUSH DWORD 0x6c624e54
0x87228942    0x12c 56                PUSH ESI
0x87228943    0x12d 6a08              PUSH 0x8
0x87228945    0x12f e89c000000        CALL 0x872289e6
```

Source Thesis: Figure 50: Disassembly of tcpip.sys TcpStartSendModule section

OWN CONTRIBUTION

REFLECTION (I)

- Creation of “NDIS Struct Relation Map”
  - Strategical exploration of NDIS stack possible
- Comparison and summarized documentation about NDIS 6.* stack

CONCLUSION

REFLECTION (II)

GOAL

THESIS OBJEKTIVES

- Analysing network stack of Windows operating systems that use NDIS 6.*
- Identify network connection structures in memory and document state changes

- NDIS 6.* stack was analysed and documented
- New possible network connection structures possible to be identified, state changes not documented yet
- Amount of work needed for creating understanding was underestimated

→ A lot is still possible to be analysed

POSSIBLE NEXT STEPS

OUTLOOK

- Further analysis of NDIS structs
- Check for possible non-public pool tags
- Attempting matching of pool tags with functions/structs
- Possible extension to forensic tools

QUESTIONS?

THANK YOU