Automating Windows Kernel Pool Overflow/Corruption Exploits Development

Nikita Tarakanov

Seoul, South Korea

POC 2018

8th of November 2018

Agenda

- Introduction
- Pool super basics
- Recap of known attacks + some new
- Framework
- Q&A

Who is Nikita Tarakanov

- Independent security researcher from USSR/Russian Federation
- Speaker (present research at various conferences since 2009)
- Trainer/Lecturer/Professor
- Funny dude ☺️

Introduction

- Ring3(IE, Adobe Reader, Flash player, MS Office etc) applications as first attack vector
- Not privileged level
- Sandboxes (IE EPM, Reader sandbox, Chrome sandbox etc)
- Need to get Ring0 to have ability to make fancy stuff
- So, Elevation of Privileges (R3->R0) Exploits/Vulnerabilities are critical
- Good examples: pwn2own 2013/2014 IE EPM sandbox escapes via kernel exploit

Pool basics

- Following 5 slides are copy-paste from work of mighty Tarjei Mandt

Pool Header 32-bits

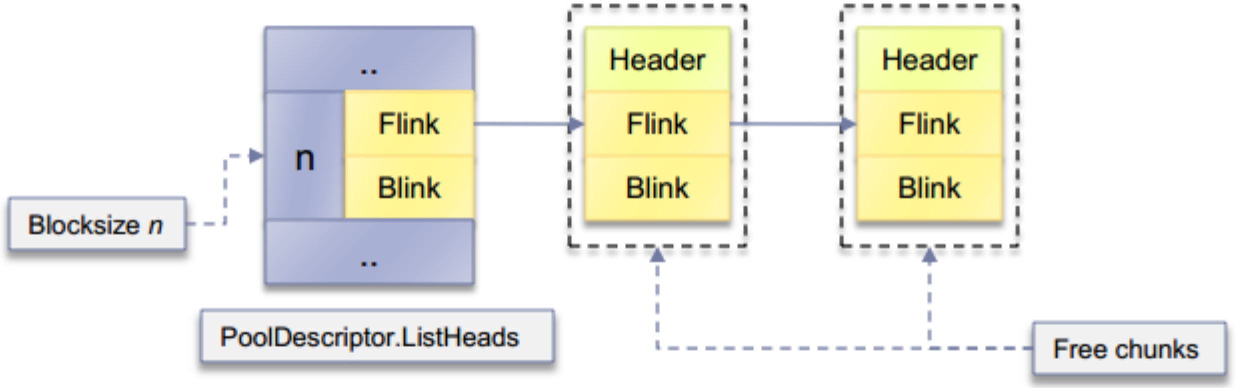
- kd> **dt nt!_POOL_HEADER**
- +0x000 PreviousSize : Pos 0, 9 Bits
- +0x000 PoolIndex : Pos 9, 7 Bits
- +0x002 BlockSize : Pos 0, 9 Bits
- +0x002 PoolType : Pos 9, 7 Bits
- +0x004 PoolTag : Uint4B
- PreviousSize: BlockSize of the preceding chunk
- PoolIndex: Index into the associated pool descriptor array
- BlockSize: (NumberOfBytes+0xF) >> 3
- PoolType: Free=0, Allocated=(PoolType|2)
- PoolTag: 4 printable characters identifying the code responsible for the allocation

Pool Header 64-bits

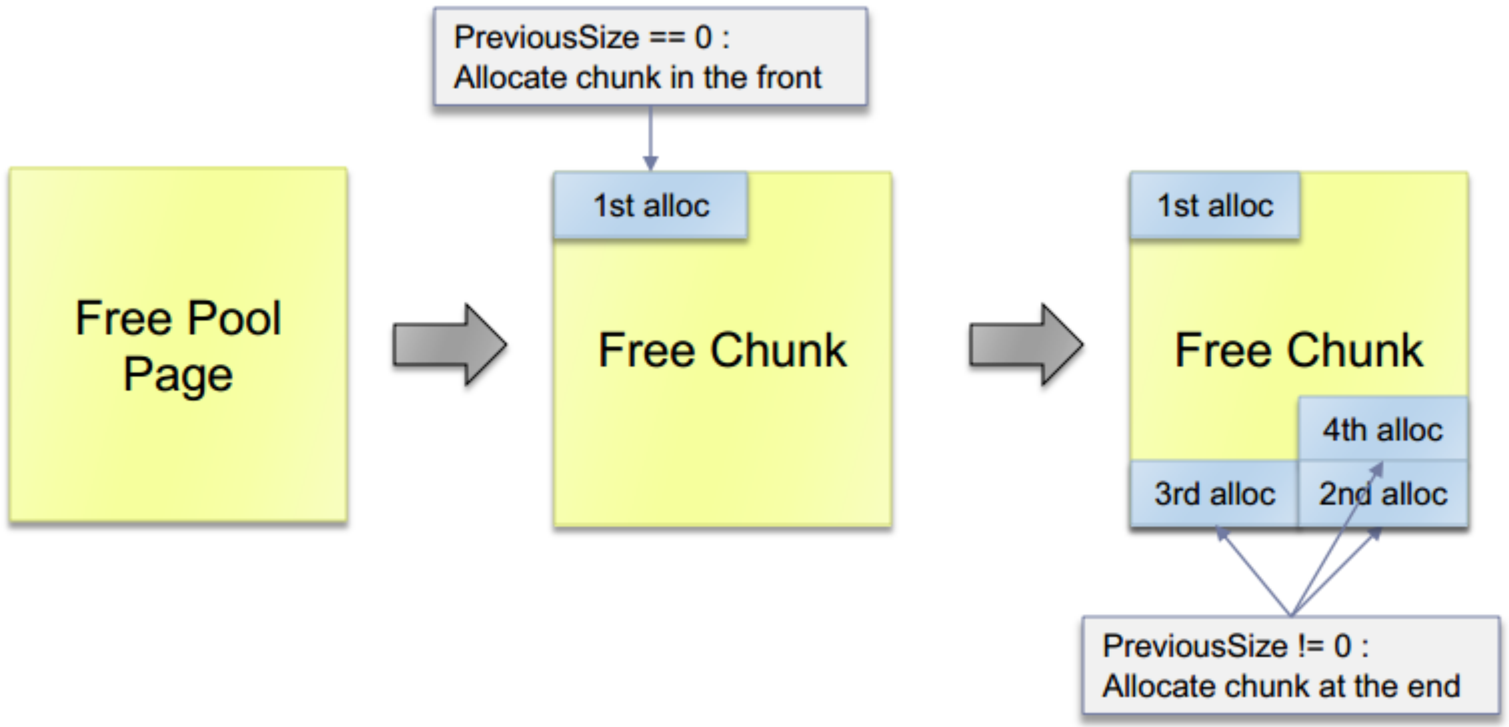
- kd> **dt nt!_POOL_HEADER**
- +0x000 PreviousSize : Pos 0, 8 Bits
- +0x000 PoolIndex : Pos 8, 8 Bits
- +0x000 BlockSize : Pos 16, 8 Bits
- +0x000 PoolType : Pos 24, 8 Bits
- +0x004 PoolTag : Uint4B
- +0x008 ProcessBilled : Ptr64 _EPROCESS
- BlockSize: (NumberOfBytes+0x1F) >> 4 (256 ListHeads entries due to 16 byte block size)
- ProcessBilled: Pointer to process object charged for the pool allocation (used in quota management)

Free Chunks

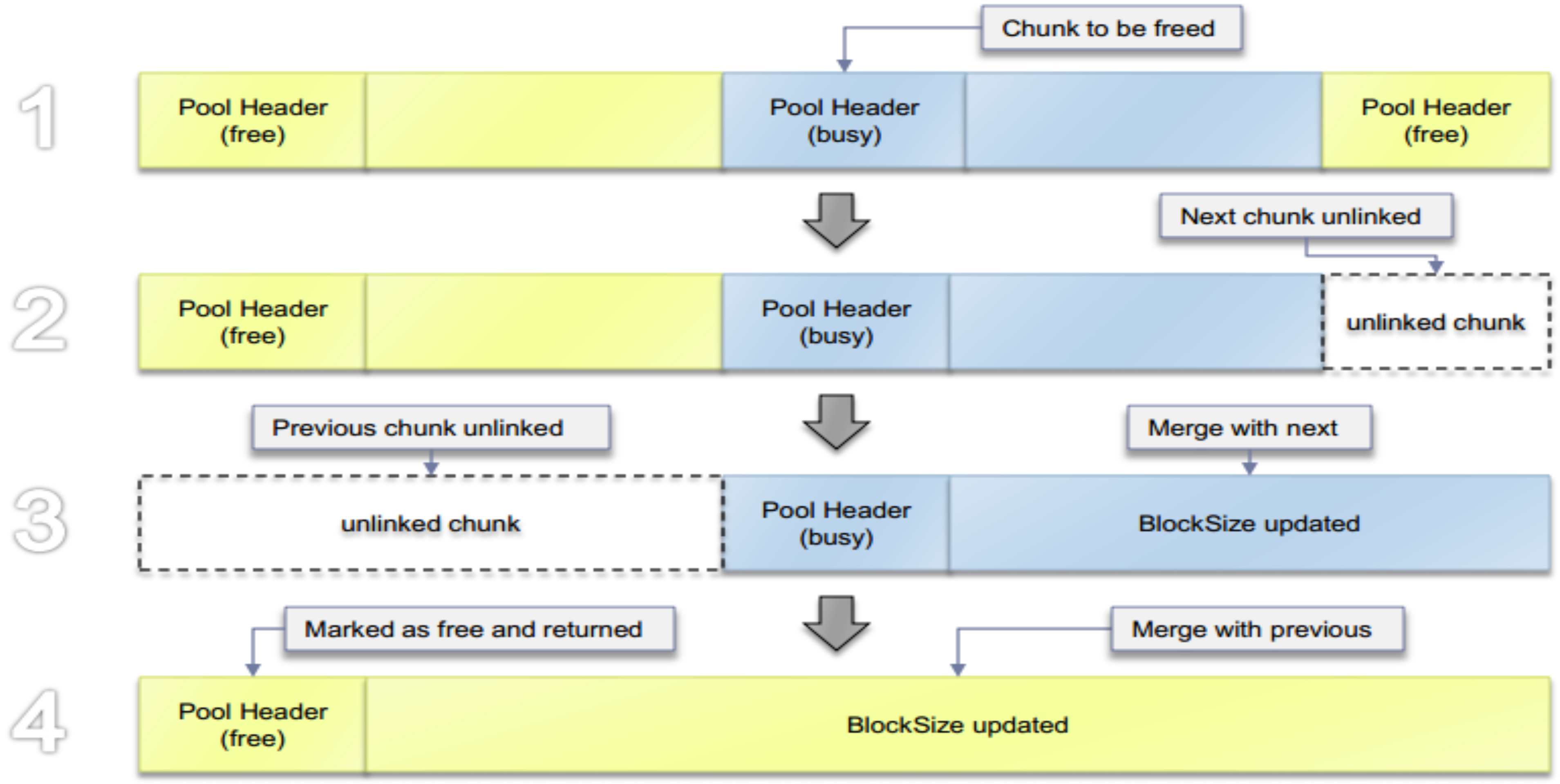
- If a pool chunk is freed to a pool descriptor ListHeads list, the header is followed by a **LINK_ENTRY** structure
- Pointed to by the ListHeads doubly-linked list
- kd> **dt nt!_LIST_ENTRY**
- +0x000 Flink : Ptr32 _LIST_ENTRY
- +0x004 Blink : Ptr32 _LIST_ENTRY



Allocation order



Merging Pool Chunks



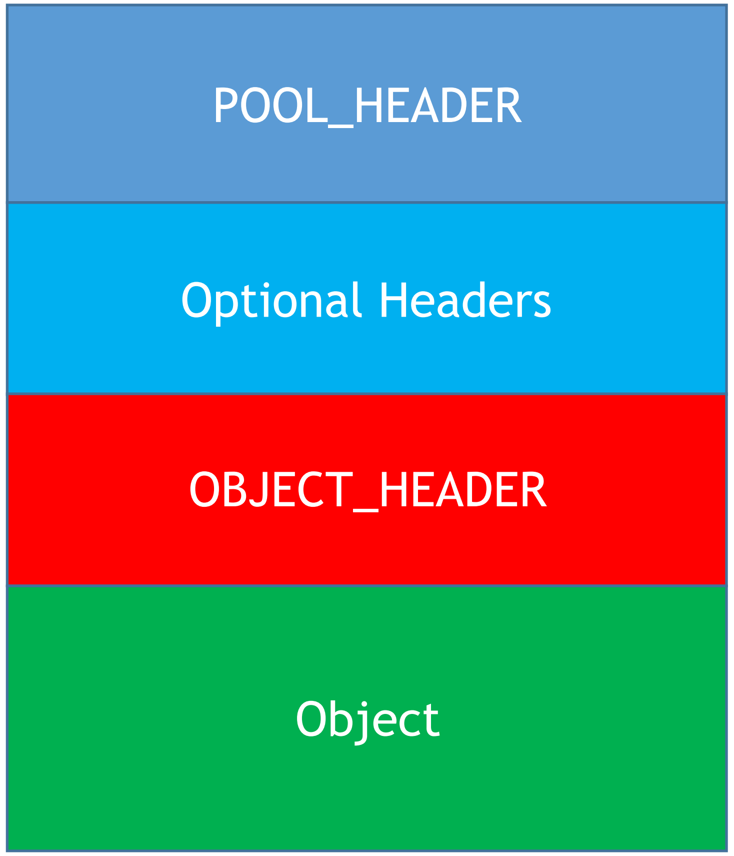
Recap of current attacks

- Pool metadata corruption - out of scope
- Object metadata corruption (DKOHM)
- DKOHM + DKOM

Object Metadata

- OBJECT_HEADER

- Optional headers



OBJECT_HEADER

- • kd> dt nt!_OBJECT_HEADER
- • +0x000 PointerCount : Int4B
- • +0x004 HandleCount : Int4B
- • +0x004 NextToFree : Ptr32 Void
- • +0x008 Lock : _EX_PUSH_LOCK
- **• +0x00c TypeIndex : UChar <- Index of pointer to OBJECT_TYPE structure in ObTypeIndexTable**
- • +0x00d TraceFlags : UChar
- • +0x00d DbgRefTrace : Pos 0, 1 Bit
- • +0x00d DbgTracePermanent : Pos 1, 1 Bit
- • +0x00e InfoMask : UChar
- • +0x00f Flags : UChar
- • +0x010 ObjectCreateInfo : Ptr32 _OBJECT_CREATE_INFORMATION
- • +0x010 QuotaBlockCharged : Ptr32 Void
- • +0x014 SecurityDescriptor : Ptr32 Void
- • +0x018 Body : _QUAD

ObTypeIndexTable

```
kd> dd nt!ObTypeIndexTable L40
81a3edc0 00000000 bad0b0b0 8499c040 849aa390
81a3edd0 84964f70 8499b4c0 84979500 84999618
81a3ede0 84974868 849783c8 8499bf70 84970b40
81a3edf0 849a8888 84979340 849aaf70 849a6a38
81a3ee00 8496df70 8495b040 8498cf70 84930a50
81a3ee10 8495af70 8497ff70 84985040 84999e78
81a3ee20 84997f70 8496c040 849646e0 84978f70
81a3ee30 8497aec0 84972608 849a0040 849a9750
81a3ee40 849586d8 84984f70 8499d578 849ab040
81a3ee50 84958938 84974a58 84967168 84967098
81a3ee60 8496ddd0 849a5140 8497ce40 849aa138
81a3ee70 84a6c058 84969c58 8497e720 85c62a28
81a3ee80 85c625f0 00000000 00000000 00000000
```

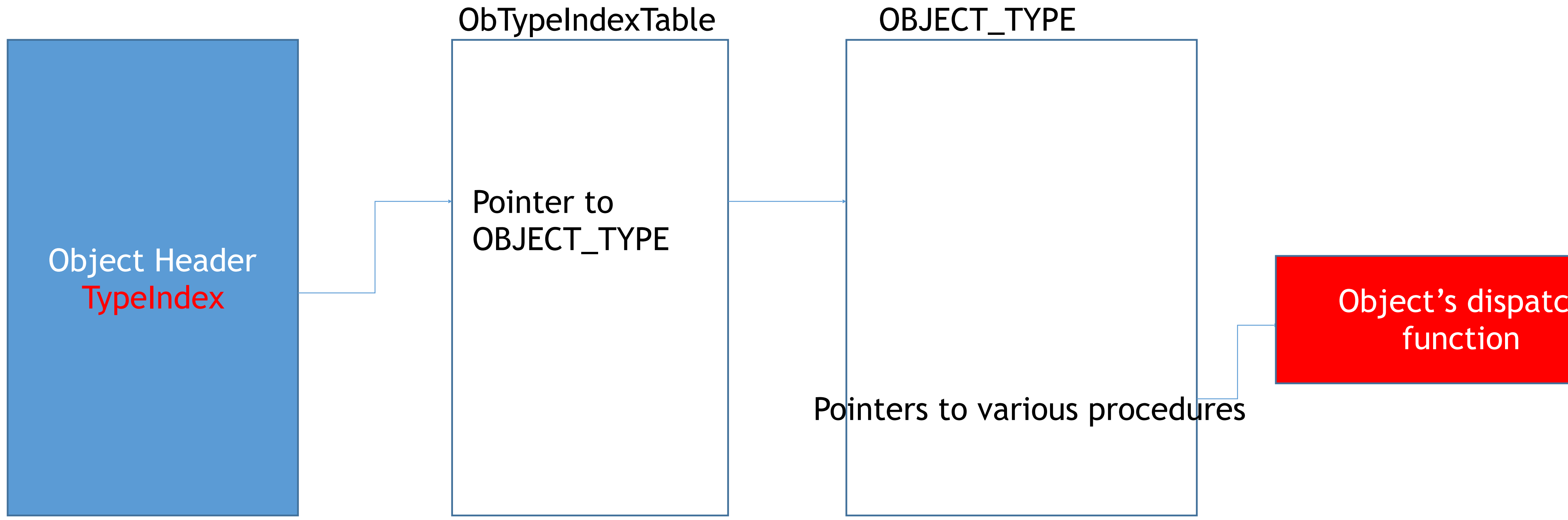
OBJECT_TYPE

- kd> dt nt!_OBJECT_TYPE
- +0x000 TypeList : _LIST_ENTRY
- +0x008 Name : _UNICODE_STRING
- +0x010 DefaultObject : Ptr32 Void
- +0x014 Index : UChar
- +0x018 TotalNumberOfObjects : Uint4B
- +0x01c TotalNumberOfHandles : Uint4B
- +0x020 HighWaterNumberOfObjects : Uint4B
- +0x024 HighWaterNumberOfHandles : Uint4B
- **+0x028 TypeInfo : _OBJECT_TYPE_INITIALIZER**
- +0x080 TypeLock : _EX_PUSH_LOCK
- +0x084 Key : Uint4B
- +0x088 CallbackList : _LIST_ENTRY

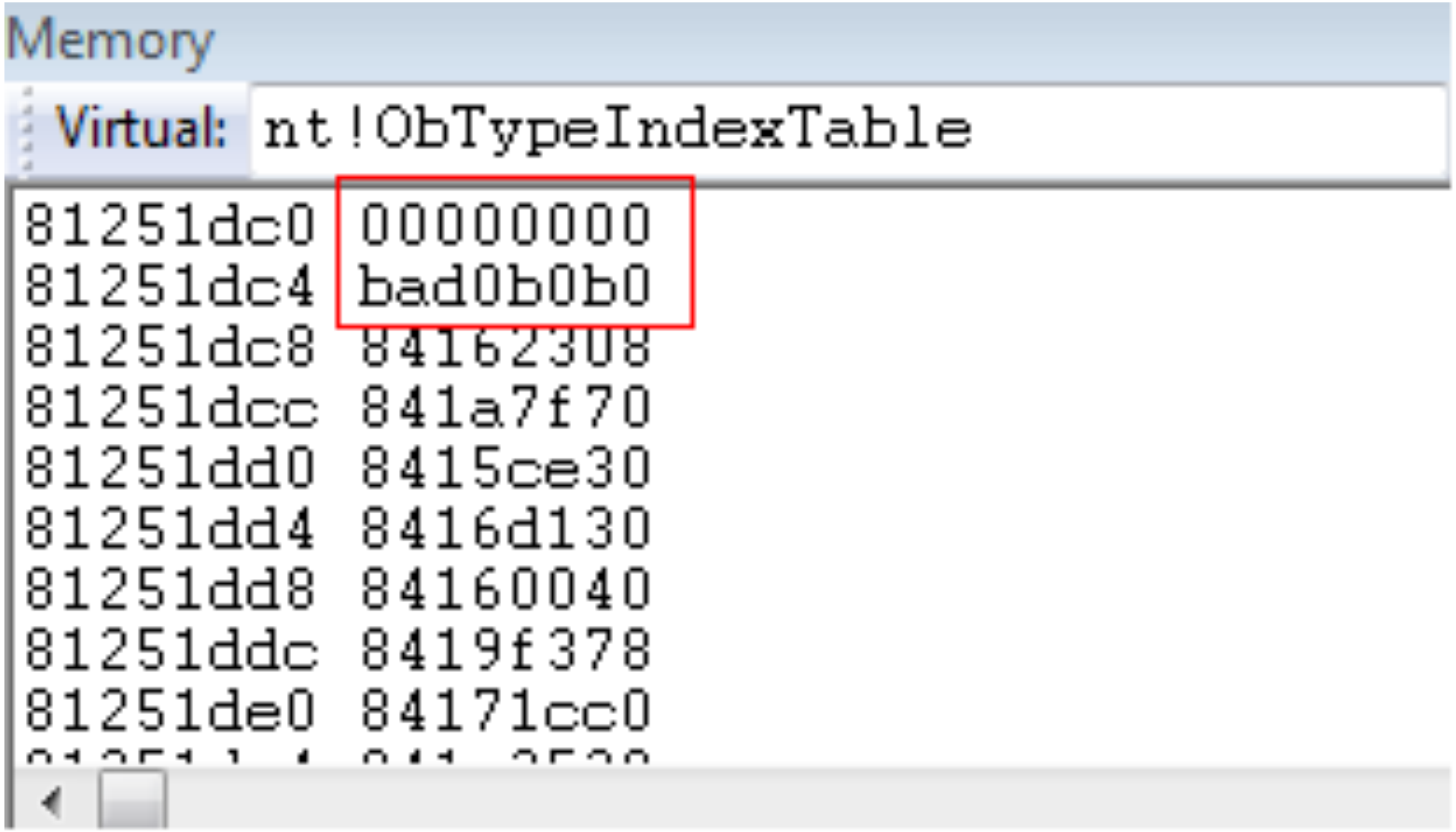
Procedures

- kd> dt nt!_OBJECT_TYPE_INITIALIZER
- [..]
- **+0x030 DumpProcedure   : Ptr32   void**
- **+0x034 OpenProcedure   : Ptr32   long**
- **+0x038 CloseProcedure   : Ptr32   void**
- **+0x03c DeleteProcedure : Ptr32   void**
- **+0x040 ParseProcedure   : Ptr32   long**
- **+0x044 SecurityProcedure : Ptr32   long**
- **+0x048 QueryNameProcedure : Ptr32   long**
- **+0x04c OkayToCloseProcedure : Ptr32   unsigned char**

ObTypeIndexTable & Object Type

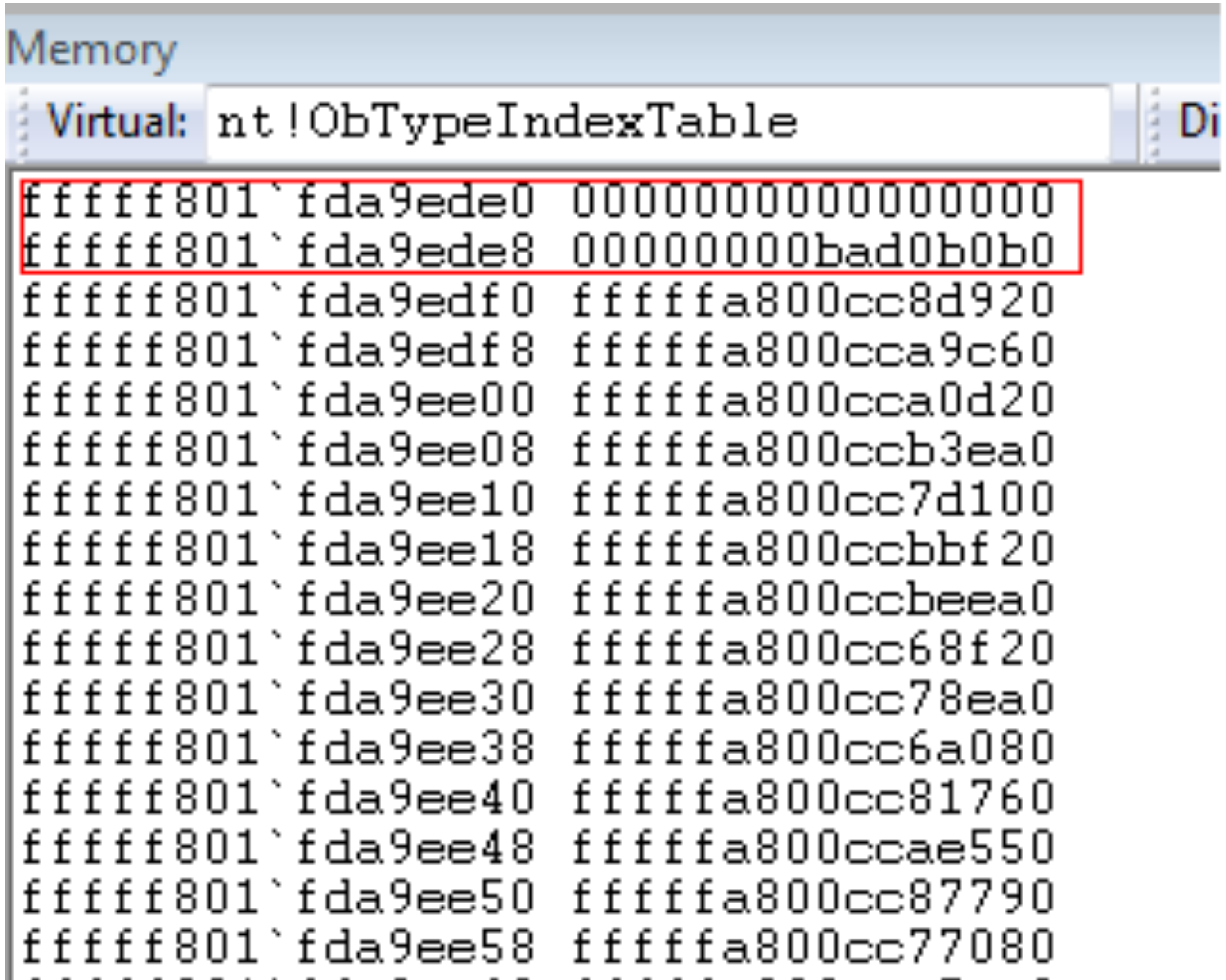


Object Type Index Table (x86)



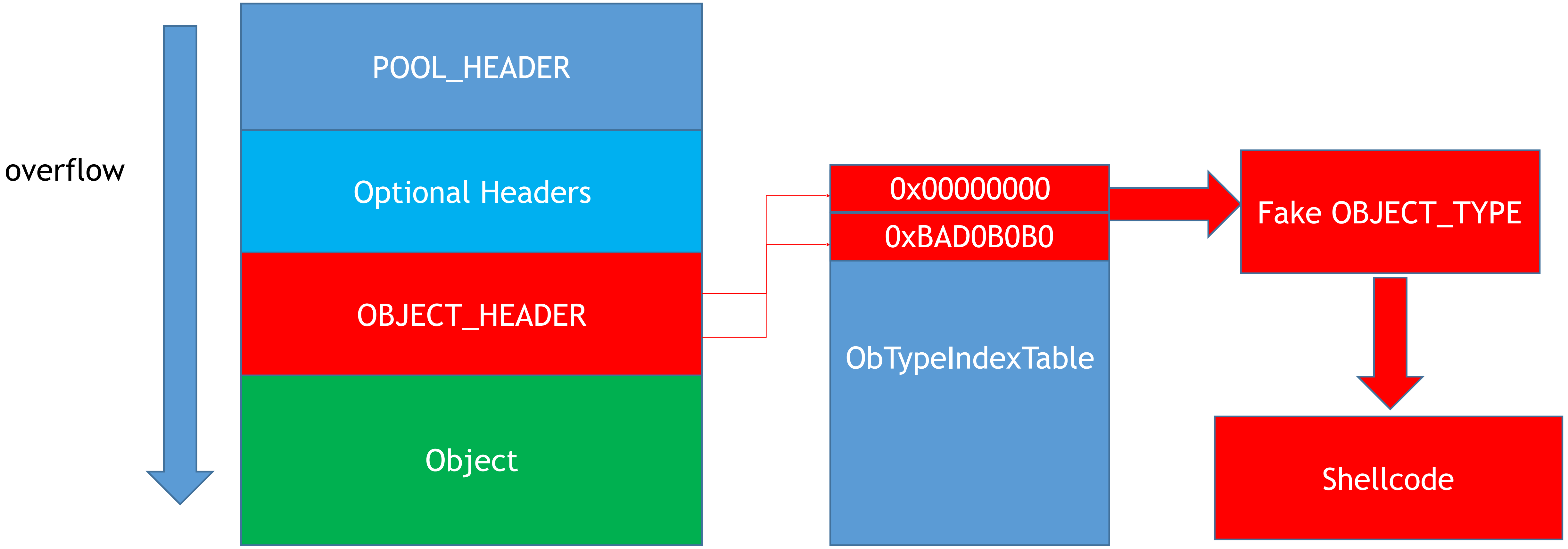
```
nt!ObTypeIndexTable
81251dc0  00000000
81251dc4  bad0b0b0
81251dc8  84162308
81251dcc  841a7f70
81251dd0  8415ce30
81251dd4  8416d130
81251dd8  84160040
81251ddc  8419f378
81251de0  84171cc0
```

Object Type Index Table (x64)



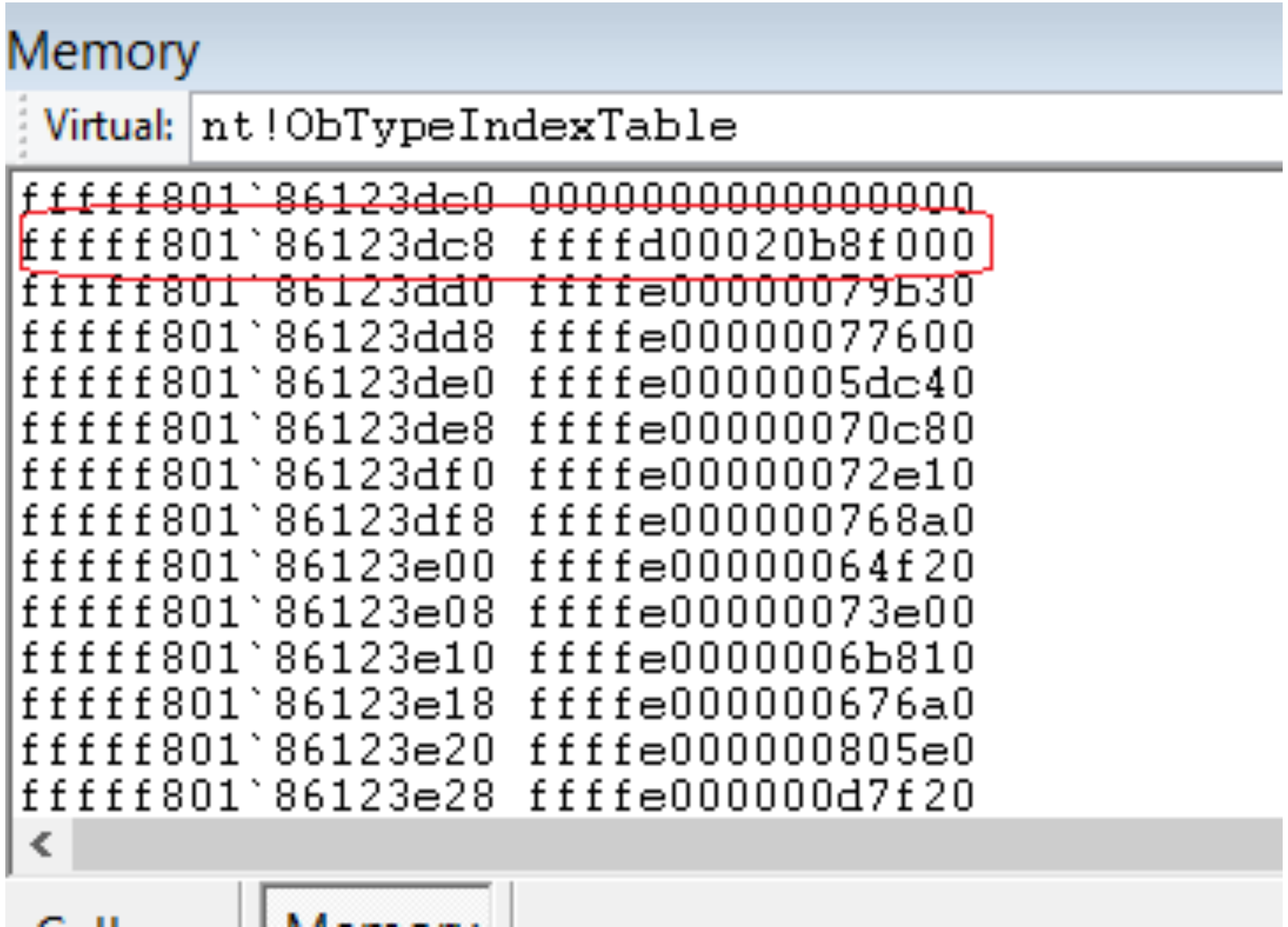
```
fffff801`fda9ede0  0000000000000000
fffff801`fda9ede8  00000000bad0b0b0
fffff801`fda9edf0  fffffa800cc8d920
fffff801`fda9edf8  fffffa800cca9c60
fffff801`fda9ee00  fffffa800cca0d20
fffff801`fda9ee08  fffffa800ccb3ea0
fffff801`fda9ee10  fffffa800cc7d100
fffff801`fda9ee18  fffffa800ccbbf20
fffff801`fda9ee20  fffffa800ccbeea0
fffff801`fda9ee28  fffffa800cc68f20
fffff801`fda9ee30  fffffa800cc78ea0
fffff801`fda9ee38  fffffa800cc6a080
fffff801`fda9ee40  fffffa800cc81760
fffff801`fda9ee48  fffffa800ccae550
fffff801`fda9ee50  fffffa800cc87790
fffff801`fda9ee58  fffffa800cc77080
```

Object metadata corruption (DKOHM): Win7



Windows 8.1 – DKOHM is dead

- 0xBAD0B0B0 has gone ☹️



```
nt!ObTypeIndexTable
fffff801`86123dc0 0000000000000000
fffff801`86123dc8 ffffd00020b8f000
fffff801`86123dd0 ffffe00000079b30
fffff801`86123dd8 ffffe00000077600
fffff801`86123de0 ffffe0000005dc40
fffff801`86123de8 ffffe00000070c80
fffff801`86123df0 ffffe00000072e10
fffff801`86123df8 ffffe000000768a0
fffff801`86123e00 ffffe00000064f20
fffff801`86123e08 ffffe00000073e00
fffff801`86123e10 ffffe0000006b810
fffff801`86123e18 ffffe000000676a0
fffff801`86123e20 ffffe000000805e0
fffff801`86123e28 ffffe000000d7f20
```

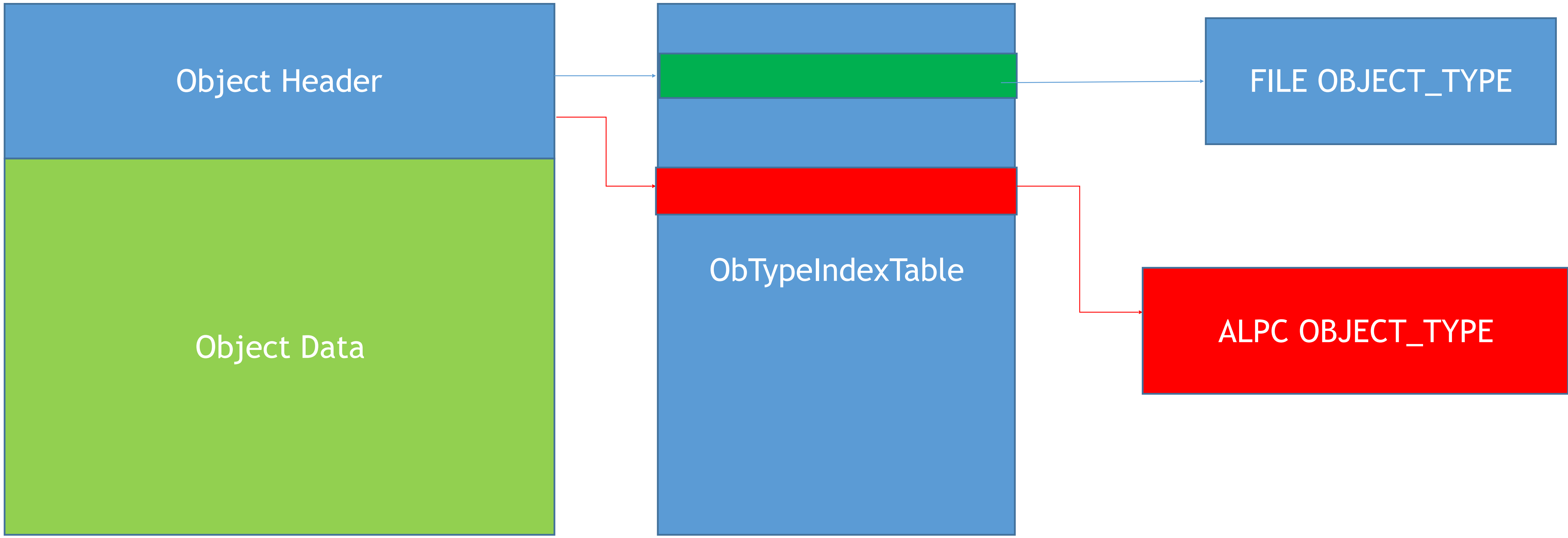
Type Confusion attack

- Object data corruption (DKOHM + DKOM)
- Object type confusion

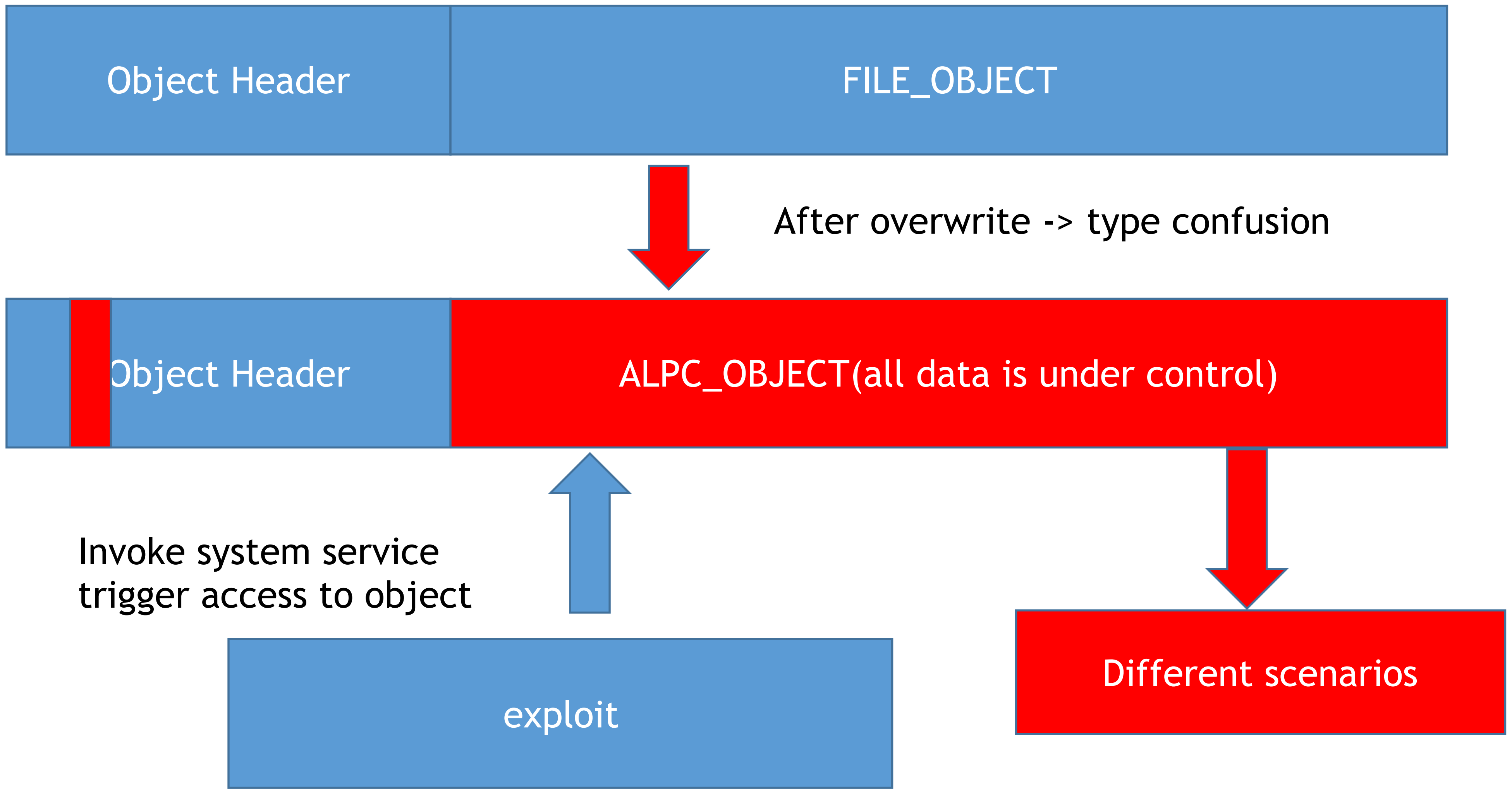
Object data corruption (DKOHM + DKOM)

- Set TypeIndex value to different object type (object type confusion)
- Object Manager is fooled (before it was Type A, not it’s Type B)
- Craft malicious object’s data (counters, pointers)
- Invoke system service(s) to trigger access to malicious object
- Profit

Object data corruption (DKOHM + DKOM)



Object data corruption (DKOHM+DKOM)



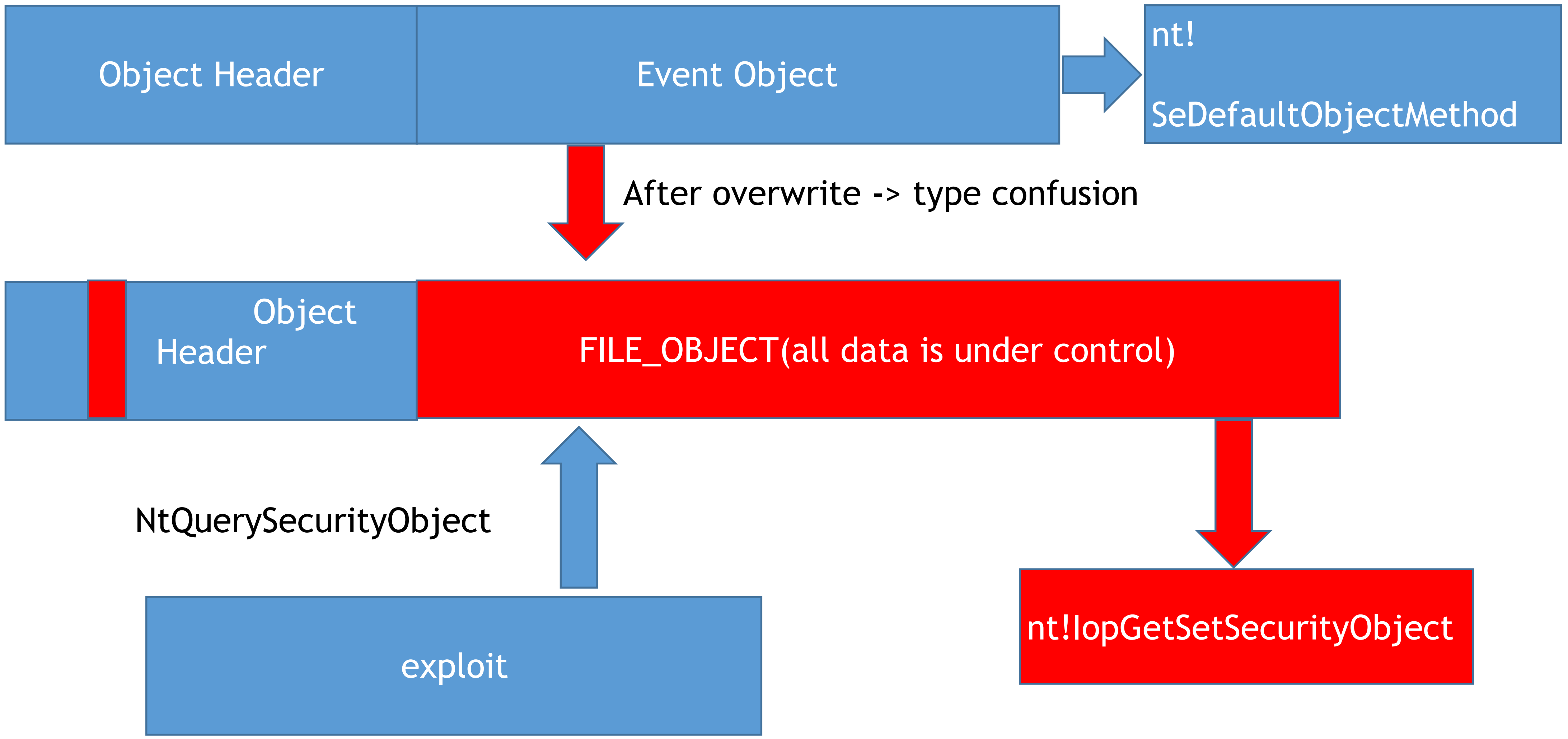
OBJECT_TYPE_INITIALIZER Procedures

- +0x030 DumpProcedure : (null)
- +0x038 OpenProcedure : (null)
- +0x040 CloseProcedure : 0xfffff801`5b913b44 void **nt!ObpCloseDirectoryObject**+0
- +0x048 DeleteProcedure : 0xfffff801`5b92743c void **nt!ObpDeleteDirectoryObject**+0
- +0x050 ParseProcedure : (null)
- +0x058 SecurityProcedure : 0xfffff801`5b848e54 long **nt!SeDefaultObjectMethod**+0
- +0x060 QueryNameProcedure : (null)
- +0x068 OkayToCloseProcedure : (null)

OBJECT_TYPE_INITIALIZER Procedures

- +0x030 DumpProcedure   : (null)
- +0x038 OpenProcedure   : (null)
- +0x040 CloseProcedure   : (null)
- +0x048 DeleteProcedure : 0xfffff801`5b9250fc   void **nt!IopDeleteDevice**+0
- +0x050 ParseProcedure   : 0xfffff801`5b86dde0   long **nt!IopParseDevice**+0
- +0x058 SecurityProcedure : 0xfffff801`5b842028   long **nt!IopGetSetSecurityObject**+0
- +0x060 QueryNameProcedure : (null)
- +0x068 OkayToCloseProcedure : (null)

Type Confusion


Object Header
Event Object
nt!
SeDefaultObjectMethod
After overwrite -> type confusion
Object
Header
FILE_OBJECT(all data is under control)
NtQuerySecurityObject
exploit
nt!IopGetSetSecurityObject

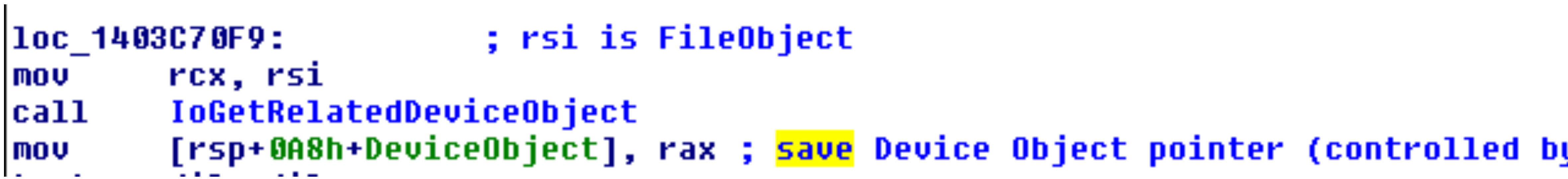
SecurityProcedure vector

- For most object types: nt!SeDefaultObjectMethod
- WmiGuid object type: nt!WmipSecurityMethod
- File/Device object type: nt!IopGetSetSecurityObject
- Key object type: nt!CmpSecurityMethod

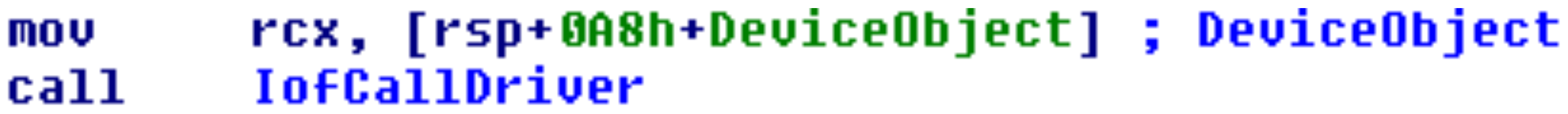
nt!IopGetSetSecurityObject

- FILE_OBJECT -> DEVICE_OBJECT -> DRIVER_OBJECT -> MAJOR_ROUTINE -> attacker’s shellcode
- Execution Hijack by three consequent dereferences!!!!

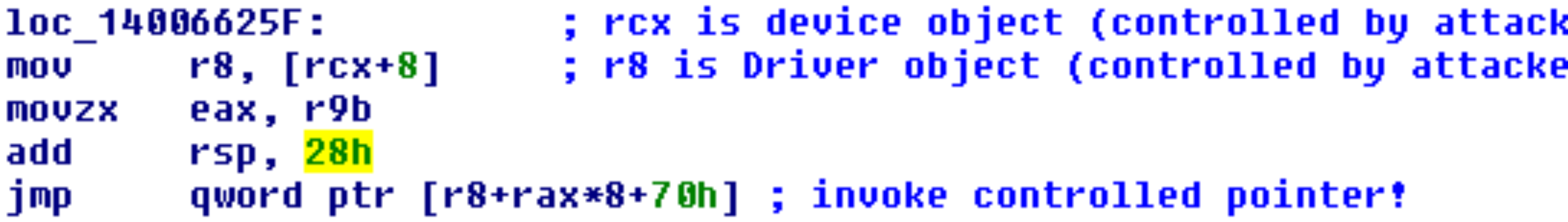
nt!IopGetSetSecurityObject



```
loc_1403C70F9:           ; rsi is FileObject
mov     rcx, rsi
call    IoGetRelatedDeviceObject
mov     [rsp+0A8h+DeviceObject], rax ; save Device Object pointer (controlled by
```



```
mov     rcx, [rsp+0A8h+DeviceObject] ; DeviceObject
call    IofCallDriver
```



```
loc_14006625F:           ; rcx is device object (controlled by attack
mov     r8, [rcx+8]      ; r8 is Driver object (controlled by attacke
movzx   eax, r9b
add     rsp, 28h
jmp     qword ptr [r8+rax*8+70h] ; invoke controlled pointer!
```

nt!IopGetSetSecurityObject chain

- 0: kd> dt nt!_FILE_OBJECT
- `   +0x000 Type             : Int2B`
- `   +0x002 Size             : Int2B`
- **`   +0x008 DeviceObject     : Ptr64 _DEVICE_OBJECT`**
- 0: kd> dt nt!_DEVICE_OBJECT
- `   +0x000 Type             : Int2B`
- `   +0x002 Size             : Uint2B`
- `   +0x004 ReferenceCount   : Int4B`
- **`   +0x008 DriverObject     : Ptr64 _DRIVER_OBJECT`**

nt!IopGetSetSecurityObject chain

- 0: kd> dt nt!_DRIVER_OBJECT
-    +0x000 Type   : Int2B
-    +0x002 Size   : Int2B
-    +0x008 DeviceObject   : Ptr64 _DEVICE_OBJECT
-    +0x010 Flags   : Uint4B
-    +0x018 DriverStart   : Ptr64 Void
-    +0x020 DriverSize   : Uint4B
-    +0x028 DriverSection   : Ptr64 Void
-    +0x030 DriverExtension : Ptr64 _DRIVER_EXTENSION
-    +0x038 DriverName   : _UNICODE_STRING
-    +0x048 HardwareDatabase : Ptr64 _UNICODE_STRING
-    +0x050 FastIoDispatch   : Ptr64 _FAST_IO_DISPATCH
-    +0x058 DriverInit   : Ptr64   long
-    +0x060 DriverStartIo   : Ptr64   void
-    +0x068 DriverUnload   : Ptr64   void
-    **+0x070 MajorFunction   : [28] Ptr64   long**

Close/Delete Procedure vector

- Huge amount of different execution flows: 56 functions
- Mostly arbitrary memory overwrite
- Some adjacent read/write
- Some hijack of execution flow

Other Procedures

- DumpProcedure, OpenProcedure, ParseProcedure,

  QueryNameProcedure, OkayToCloseProcedure

- Are rare – no interest in here

Object’s body vector (DKOM)

- There are several typical OOP interfaces
- Constructor – NtCreate* (NtCreateFile)
- Destructor – NtClose
- Getter – NtQueryInformation* (NtQueryInformationWorkerFactory)
- Setter – NtSetInformation* (NtSetInformationKey)
- Object Type specific: NtClearEvent, NtAlpcAcceptConnectPort, NtEnumerateValueKey, NtRecoverResourceManager etc

DKOHM+DKOM restrictions

- Unfortunately you cant freely use Getter/Setter/Specific when you change type of an object – caused by ValidAccessMask field ☹️
  - +0x010 Name   : _UNICODE_STRING "WindowStation"
  - +0x01c ValidAccessMask : **0xf037f**

  - +0x010 Name   : _UNICODE_STRING "Directory"
  - +0x01c ValidAccessMask : **0xf000f**

- But you can still smash object's data without changing object type

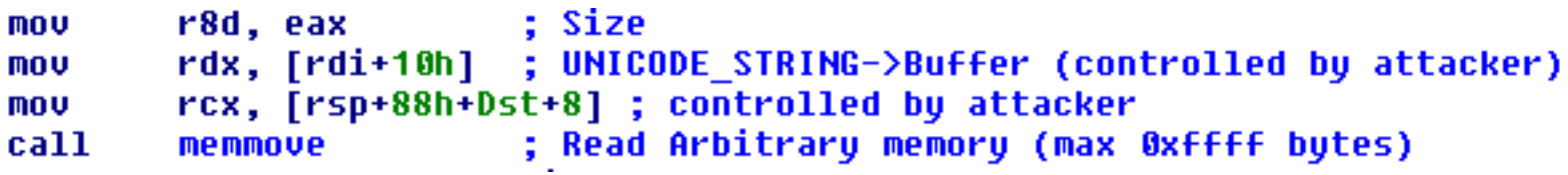
DKOHM+DKOM restrictions

- Some Object Types have same ValidAccessMask
  - +0x010 Name : _UNICODE_STRING "Section“
  - +0x01c ValidAccessMask : **0x1f001f**

  - +0x010 Name : _UNICODE_STRING "Job“
  - +0x01c ValidAccessMask **: 0x1f001f**

- So technique using Getter/Setter/Specific is possible, but limited

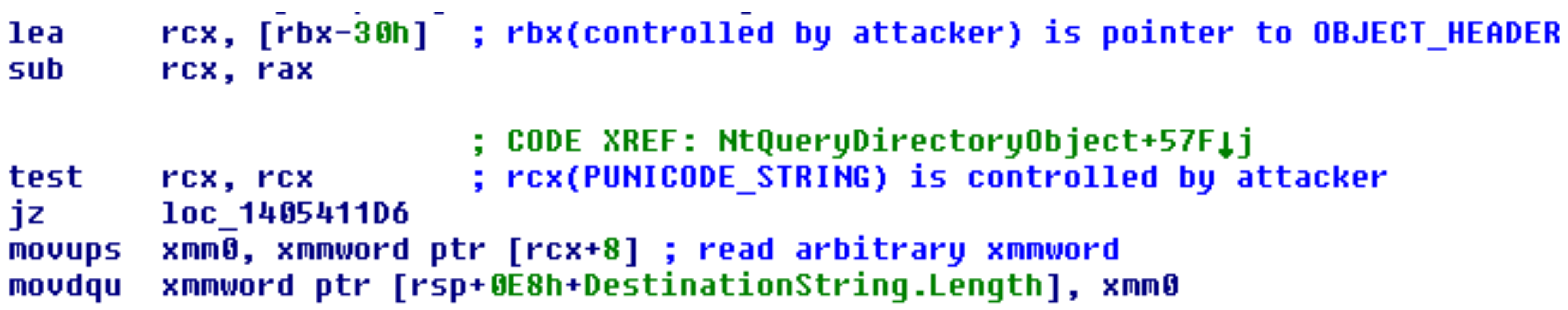
Symbolic Link: Getter vector NtQuerySymbolicLinkObject



```
mov     r8d, eax            ; Size
mov     rdx, [rdi+10h]      ; UNICODE_STRING->Buffer (controlled by attacker)
mov     rcx, [rsp+88h+Dst+8] ; controlled by attacker
call    memmove             ; Read Arbitrary memory (max 0xffff bytes)
```

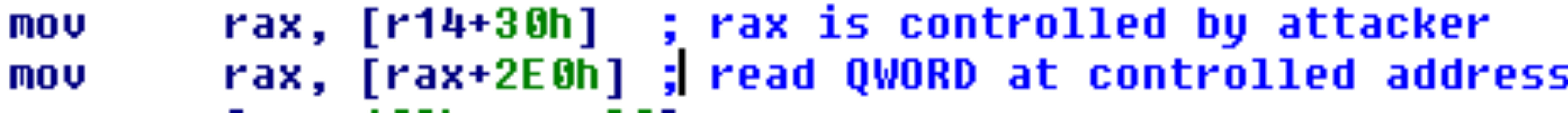
Directory Object: Getter vector NtQueryDirectoryObject

- Up-to 0x25 times of reading arbitrary memory



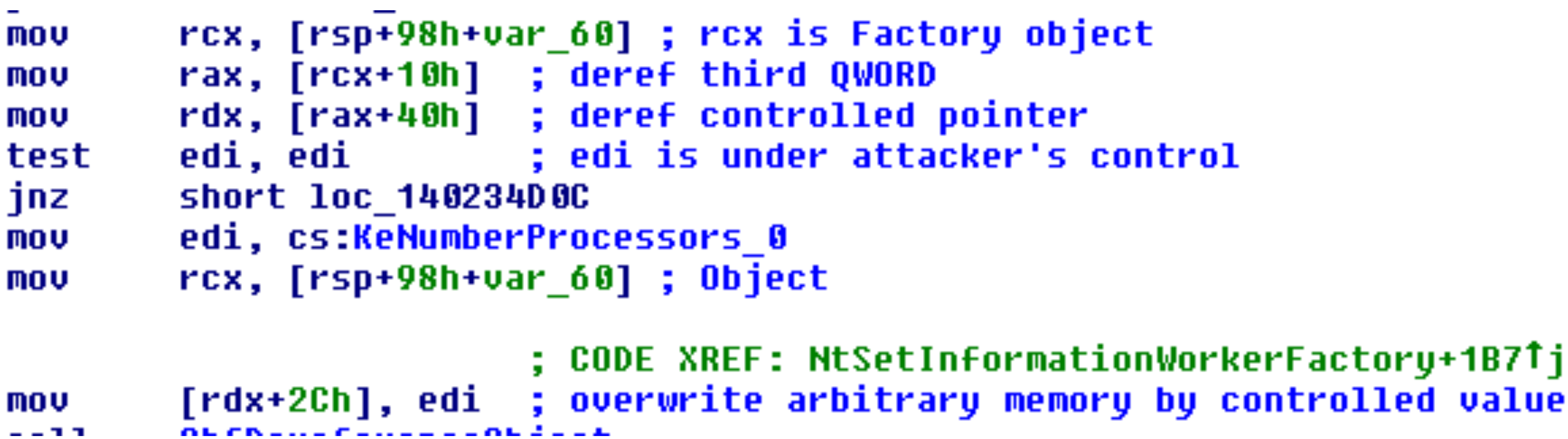
```
lea     rcx, [rbx-30h]  ; rbx(controlled by attacker) is pointer to OBJECT_HEADER
sub     rcx, rax

                        ; CODE XREF: NtQueryDirectoryObject+57F↓j
test    rcx, rcx        ; rcx(PUNICODE_STRING) is controlled by attacker
jz      loc_1405411D6
movups  xmm0, xmmword ptr [rcx+8] ; read arbitrary xmmword
movdqu  xmmword ptr [rsp+0E8h+DestinationString.Length], xmm0
```

WorkerFactory object Getter: NtQueryInformationWorkerFactory



```
mov     rax, [r14+30h]  ; rax is controlled by attacker
mov     rax, [rax+2E0h] ; read QWORD at controlled addres
```

WorkerFactory object Setter: NtSetInformationWorkerFactory

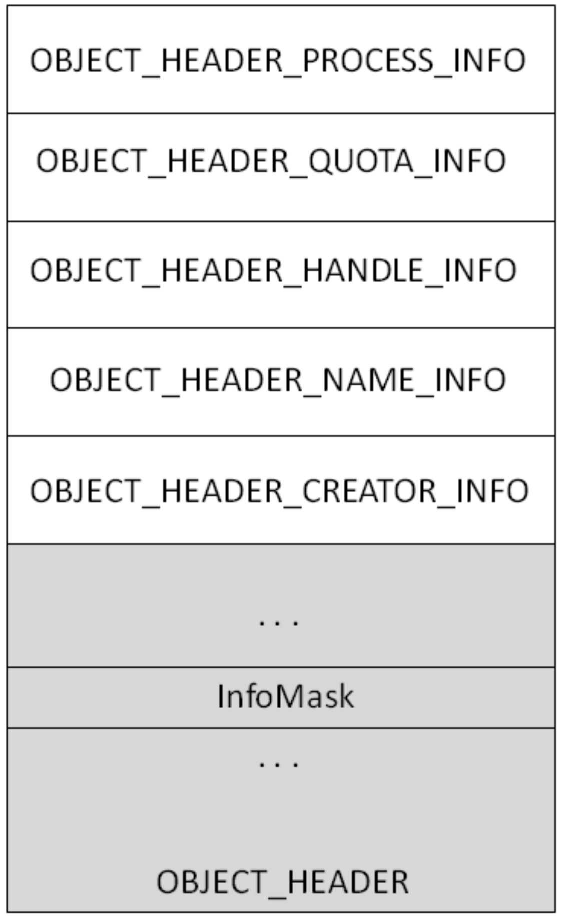


```
mov     rcx, [rsp+98h+var_60] ; rcx is Factory object
mov     rax, [rcx+10h]   ; deref third QWORD
mov     rdx, [rax+40h]   ; deref controlled pointer
test    edi, edi         ; edi is under attacker's control
jnz     short loc_140234D0C
mov     edi, cs:KeNumberProcessors_0
mov     rcx, [rsp+98h+var_60] ; Object

                         ; CODE XREF: NtSetInformationWorkerFactory+1B7↑j
mov     [rdx+2Ch], edi   ; overwrite arbitrary memory by controlled value
```

DKOHM+DKOM is killed in windows 10

- TypeIndex is encoded ☹️

DKOOHM for the rescue!



Optional Headers

- Located before OBJECT_HEADER

- Hence not triggering TypeIndex check!

Optional Headers: Quota Info

- kd> dt nt!_OBJECT_HEADER_QUOTA_INFO
-    +0x000 PagedPoolCharge : Uint4B
- +0x004 NonPagedPoolCharge : Uint4B
  +0x008 SecurityDescriptorCharge : Uint4B
- +0x00c Reserved1   : Uint4B
- **+0x010 SecurityDescriptorQuotaBlock : Ptr64 Void**
- +0x018 Reserved2   : Uint8B

DKOOHM attack over Quota Info

- CloseHandle on smashed Quota Info leads to different scenarios/primitives:
  - Arbitrary Decrement
  - Arbitrary Free

Optional Headers: Name Info

- typedef struct _OBJECT_HEADER_NAME_INFO {
  - **struct _OBJECT_DIRECTORY* Directory;**
  - **struct _UNICODE_STRING Name;**
  - LONG32   ReferenceCount;
  - ULONG32   Reserved;
- }OBJECT_HEADER_NAME_INFO;

DKOOHM attack over Name Info

- Smash OBJECT_HEADER_NAME_INFO header
- Replace **NAME->Buffer** with kernel pointer
- Call CloseHandle on smashed object
- Get Arbitrary Free primitive
- Profit!

DKOHM new attack

- TypeIndex is protected(encoded)…
- PointerCount, HandleCount, NextToFree is **NOT!**
- Decrease PointerCount,HandleCount
- Force Dealloc
- Use-after-free!

Framework internals

- Object_Pool_Memory class
- Page Class
- Object Class
- Pool Manipulation / Exploitation algos

Object: Pool used

- Paged: Directory, SymbolicLink, Token, KeyedEvent, Section, Key

- NonPaged/NonPagedNX: Process, Thread, Job, Event etc…

- Some object types use Paged & NonPaged (NAME_INFO etc)

Object class

- Object type
- Pool type used
- Kernel Address
- Handle
- Size of object
- Size of allocated chunk
- Size of consumed memory (Paged – name of object)
- Flags – Optional headers

Page Class

- Kernel Address
- List of Objects
- Gaps(Allocated/Controlled ranges, Uncontrolled ranges)

Object_Pool_memory Class

- Manages Objects & Pool Memory
- Free, Allocated, Controlled Memory & Pages
- Manipulation algos

Pool Manipulation / Exploitation algos

Classes of Pool Memory Corruptions

- Memset(Pool_mem, Const, Const/Var)

- Memcpy(Pool_mem_1, Mem_2(cotrolled/SemiControlled), Const/Var)

- Out-of-bounds write(s)

Memset Corruption Class

- Currently exploitable(partially) on Win 7 only
- DKOHM (0x0/0x1 TypeIndex) attack
- Make Hole at the bottom of a page
- Corrupt TypeIndex of object at Adjacent Page
- Bypasses Pool Metadata checks
- EIP/RIP control

Memcpy Class

- Win 7 – DKOHM (TypeIndex 0x0/0x1) attack
- Win 8.1 – DKOHM/DKOM/DKOOHM attack
- Win 10 – DKOOHM attack

Memcpy class

- Make a hole
- Corrupt Optional Header
- Via Arbitrary Decrement: length/size of GDI/USER object -> AAR/AAW
- Via Arbitrary Decrement: Decrease PointerCount -> use-after-free
- Via Arbitrary Free: Free Bound Object -> trigger dereference

Out-of-bounds write

- Currently not implemented
- Need to make map of every Object(Type) Body
- (Probably will be done in the end of this year)

Q&A

- Thanks!!!

References

- Tarjei Mandt BH US 2012
- Nikita Tarakanov HITB AMS 2013
- Nikita Tarakanov BH US 2014