Data-only Pwning Microsoft Windows Kernel: Exploitation of Kernel Pool Overflows on Microsoft Windows 8.1

Nikita Tarakanov,

6th of August, BlackHat USA 2014

Las-Vegas, USA

Agenda

- Introduction
- Basic of previous attacks
- New idea
- Mitigations
- Q&A

Introduction

- Pool overflow exploitation techniques are quite well studied: from Windows XP/2003 times to Windows 7/8 present
- Most of them target Pool internal algos/structures
- Microsoft makes Pool overflows exploitation harder and harder
- New ideas/techniques should appear!

Pool basics

Pool Header 32-bits

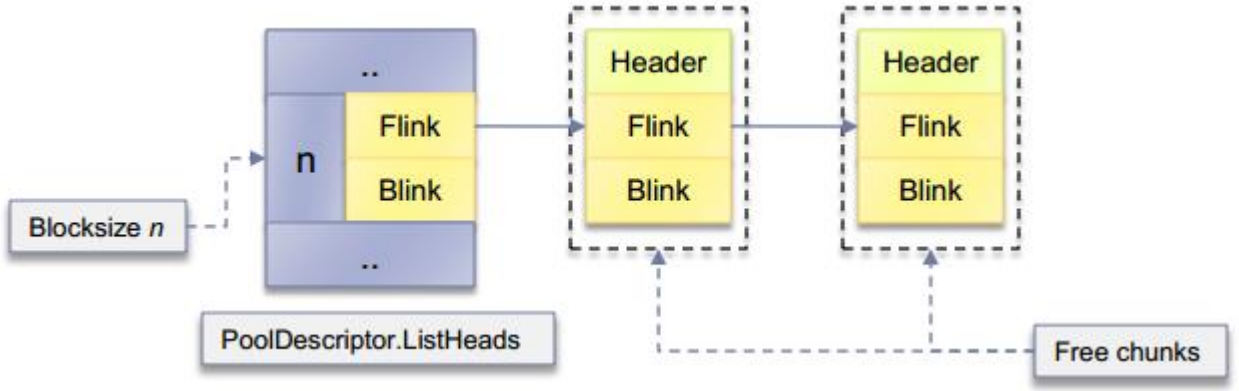
- kd> **dt nt!_POOL_HEADER**
- +0x000 PreviousSize : Pos 0, 9 Bits
- +0x000 PoolIndex : Pos 9, 7 Bits
- +0x002 BlockSize : Pos 0, 9 Bits
- +0x002 PoolType : Pos 9, 7 Bits
- +0x004 PoolTag : Uint4B
- PreviousSize: BlockSize of the preceding chunk
- PoolIndex: Index into the associated pool descriptor 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

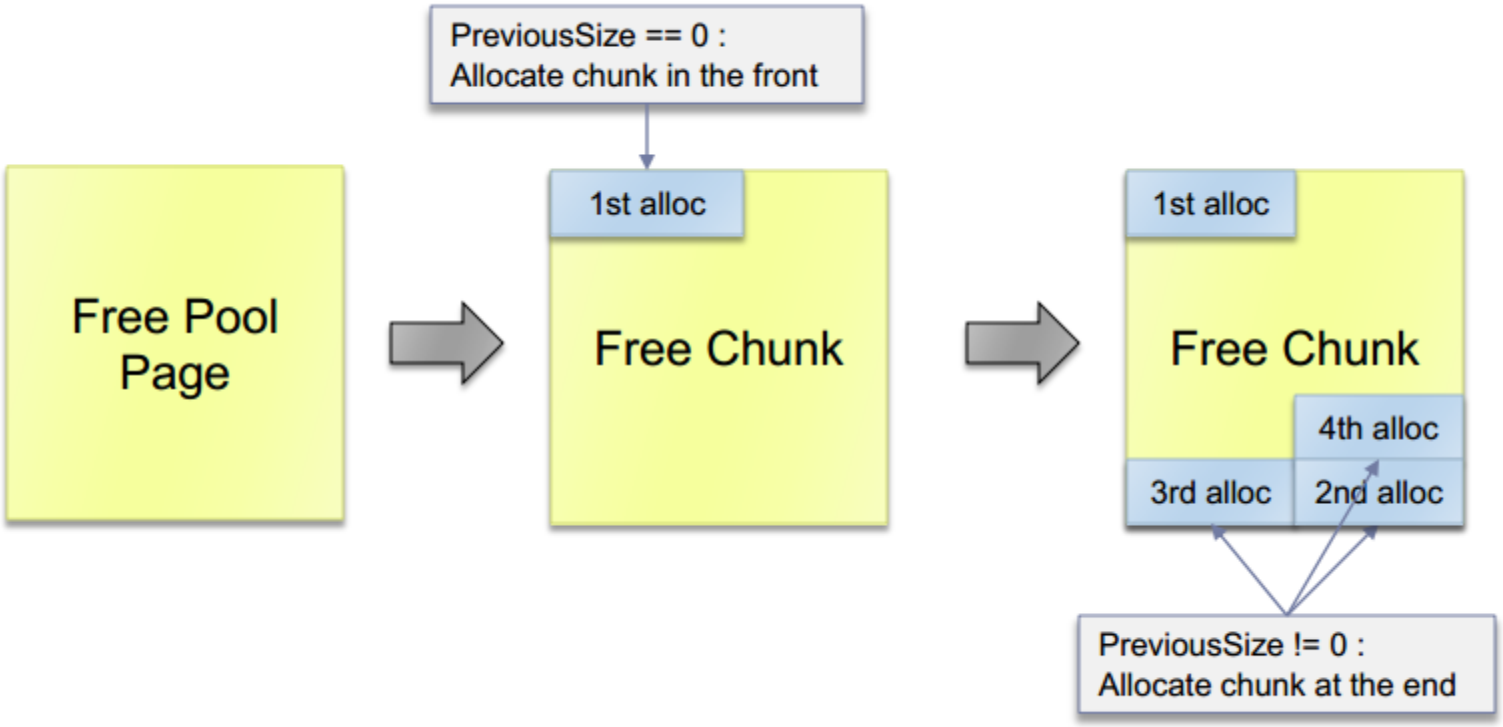
- kd> **dt nt!_POOL_HEADER**
- +0x000 PreviousSize : Pos 0, 8 Bits
- +0x000 PoolIndex : Pos 8, 8 Bits
- +0x000 BlockSize : Pos 16, 8 Bits
- +0x000 PoolType : Pos 24, 8 Bits
- +0x004 PoolTag : Uint4B
- +0x008 ProcessBilled : Ptr64 _EPROCESS
- BlockSize: (NumberOfBytes+0x1F) >> 4 (256 ListHeads entries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

Free Chunks

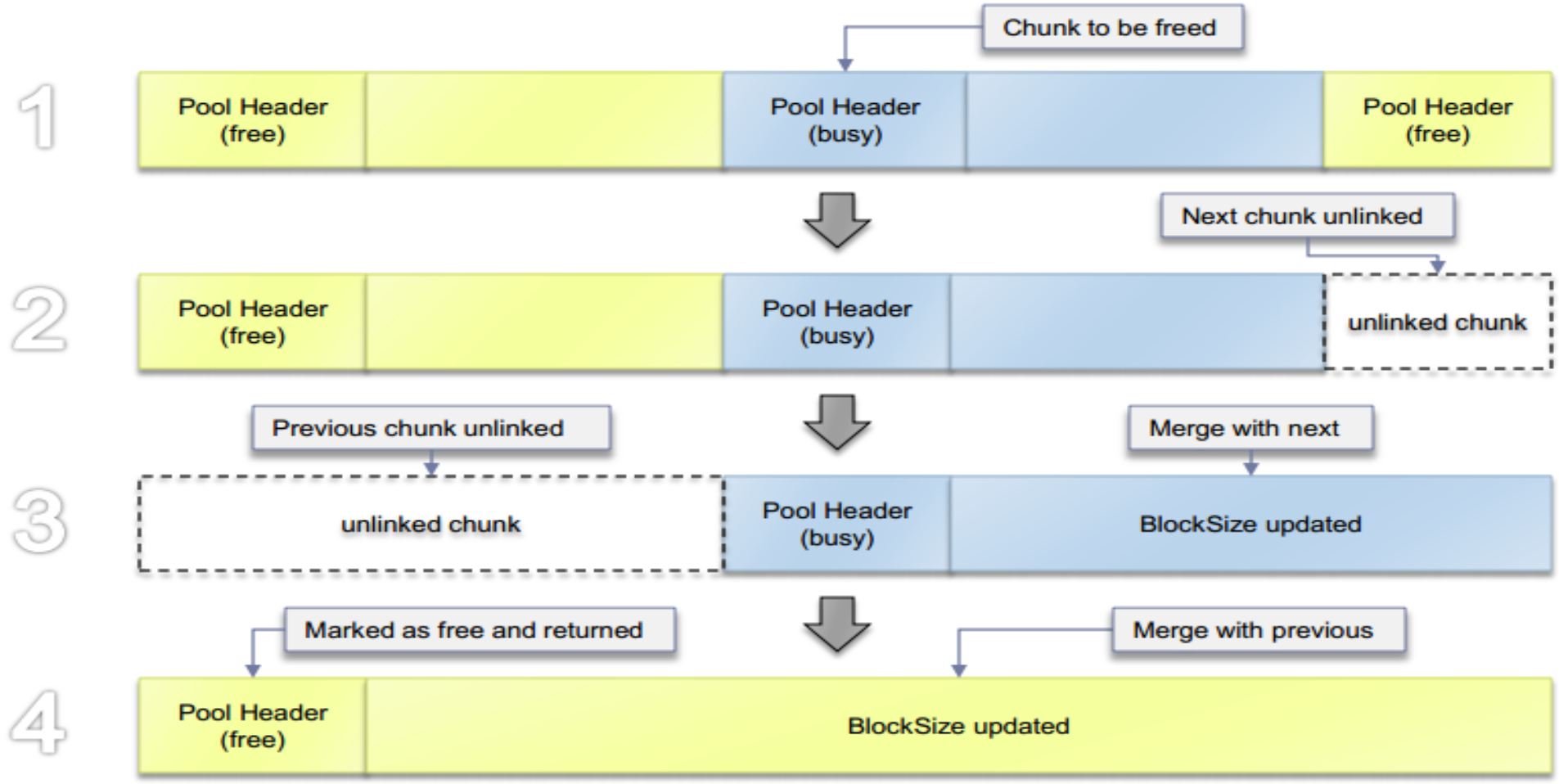
- If a pool chunk is freed to a pool descriptor ListHeads list, the header is followed by a **LINK_ENTRY** structure
- Pointed to by the ListHeads doubly-linked list
- kd> **dt nt!_LIST_ENTRY**
- +0x000 Flink : Ptr32 _LIST_ENTRY
- +0x004 Blink : Ptr32 LIST ENTRY



Allocation order



Merging Pool Chunks



Basic of previous attacks

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

Object Metadata

- OBJECT_HEADER
- Optional headers
- Object’s body

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

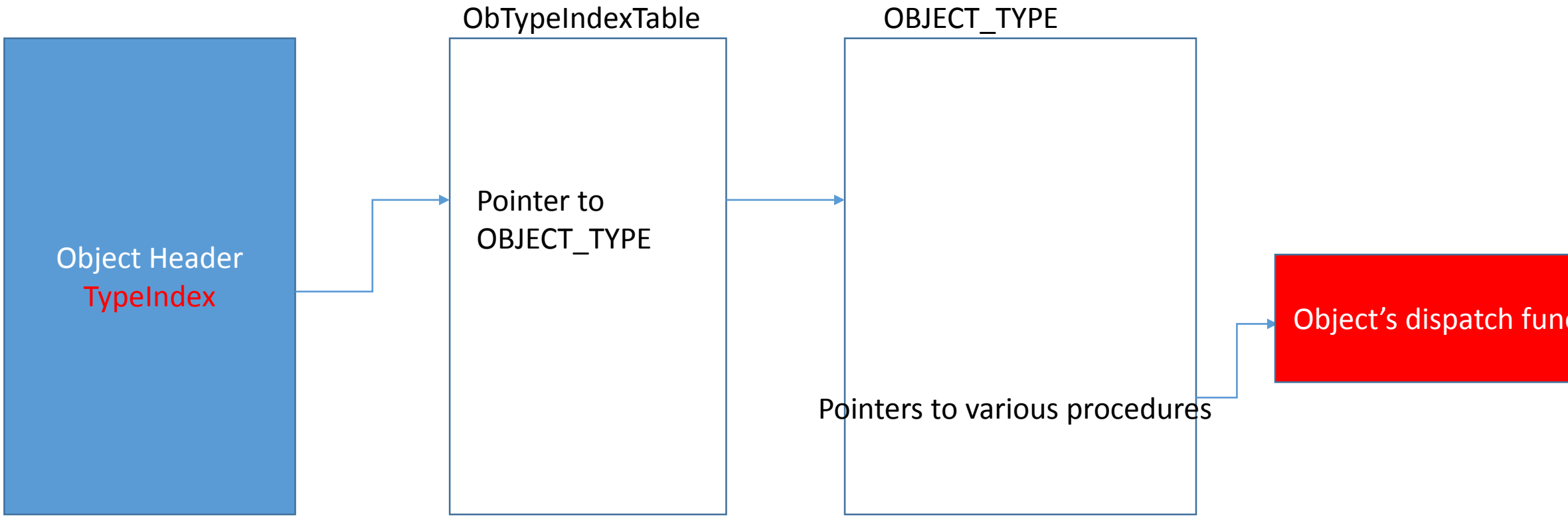
OBJECT_TYPE

- 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

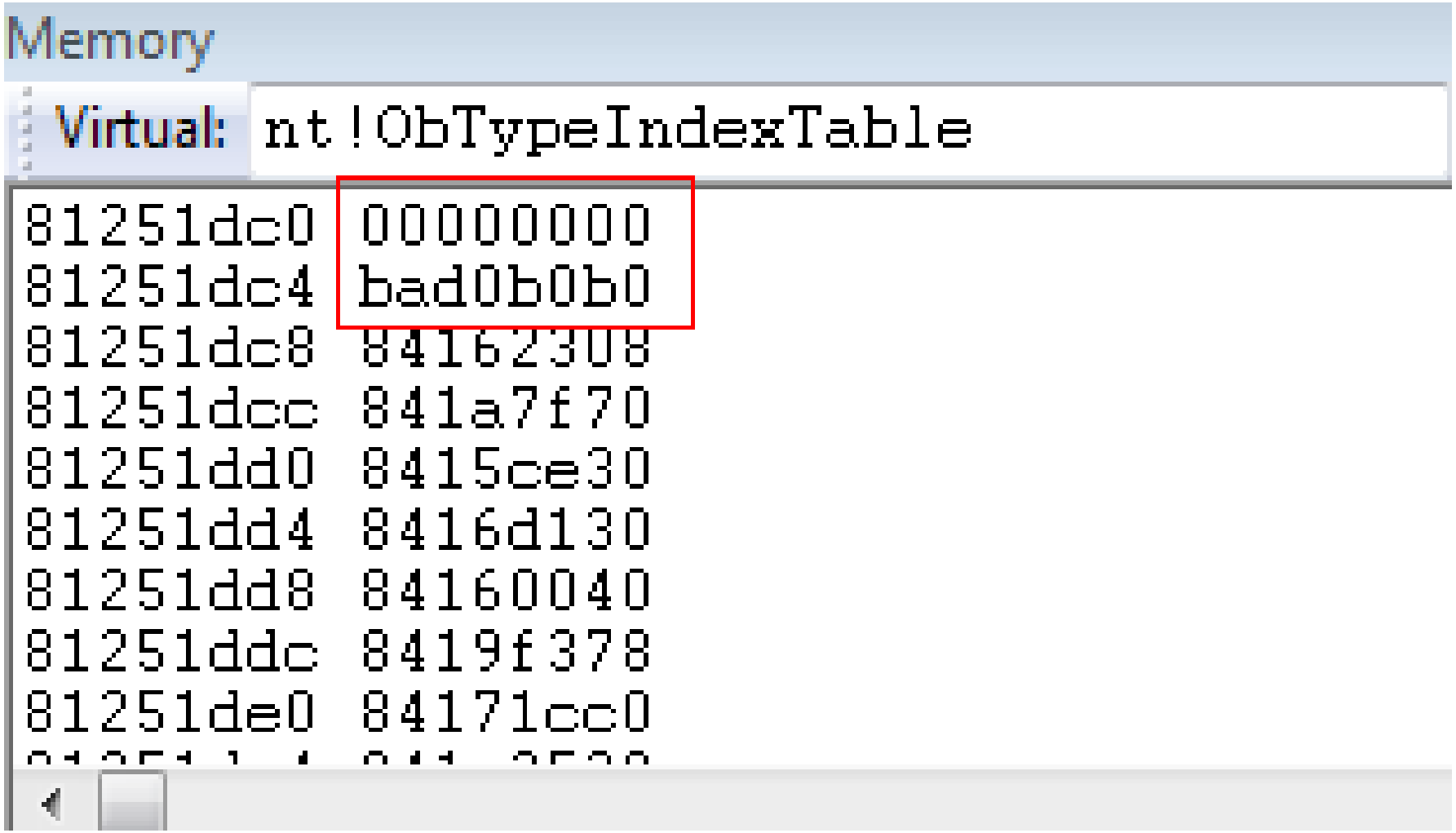
Procedures

- 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)



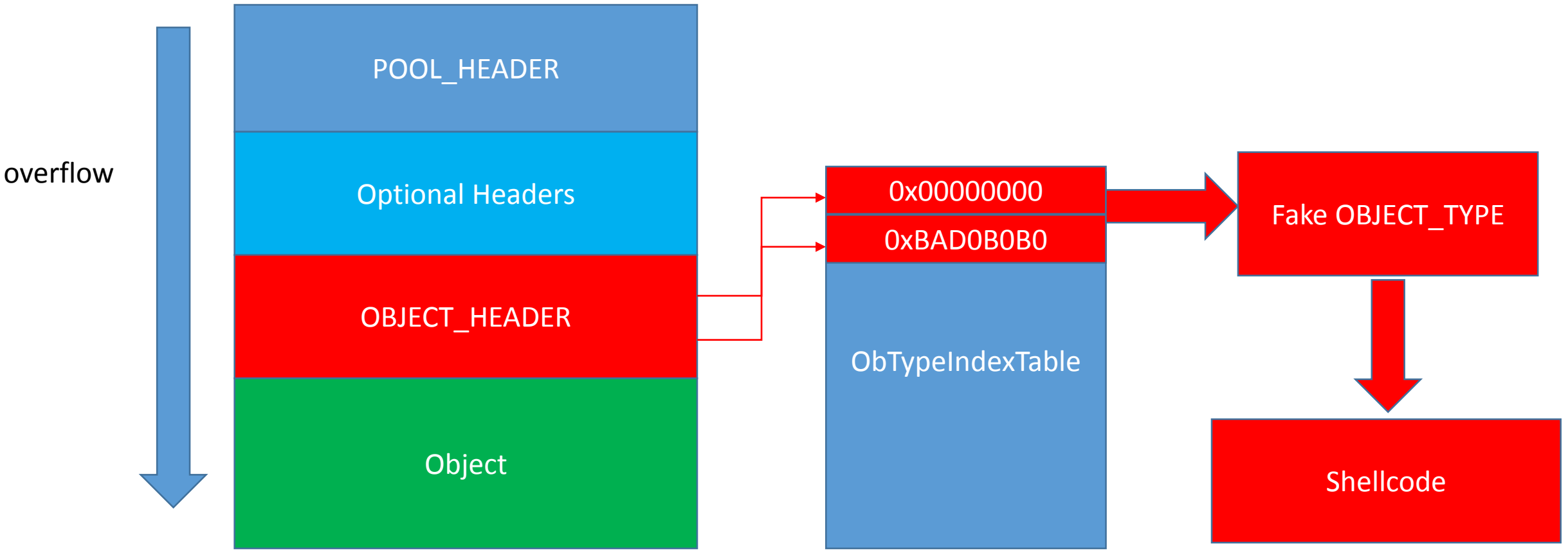
Object Type Index Table (x64)

Memory

Virtual: nt!ObTypeIndexTable

```
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)



New idea

- TBD

- TBD

New idea

- TBD

New idea

- TBD

New idea

- TBD
- TBD
- TBD

New idea

- TBD

New idea

- TBD
- TBD
- TBD
- TBD

Mitigations

- TBD
- TBD

Q&A