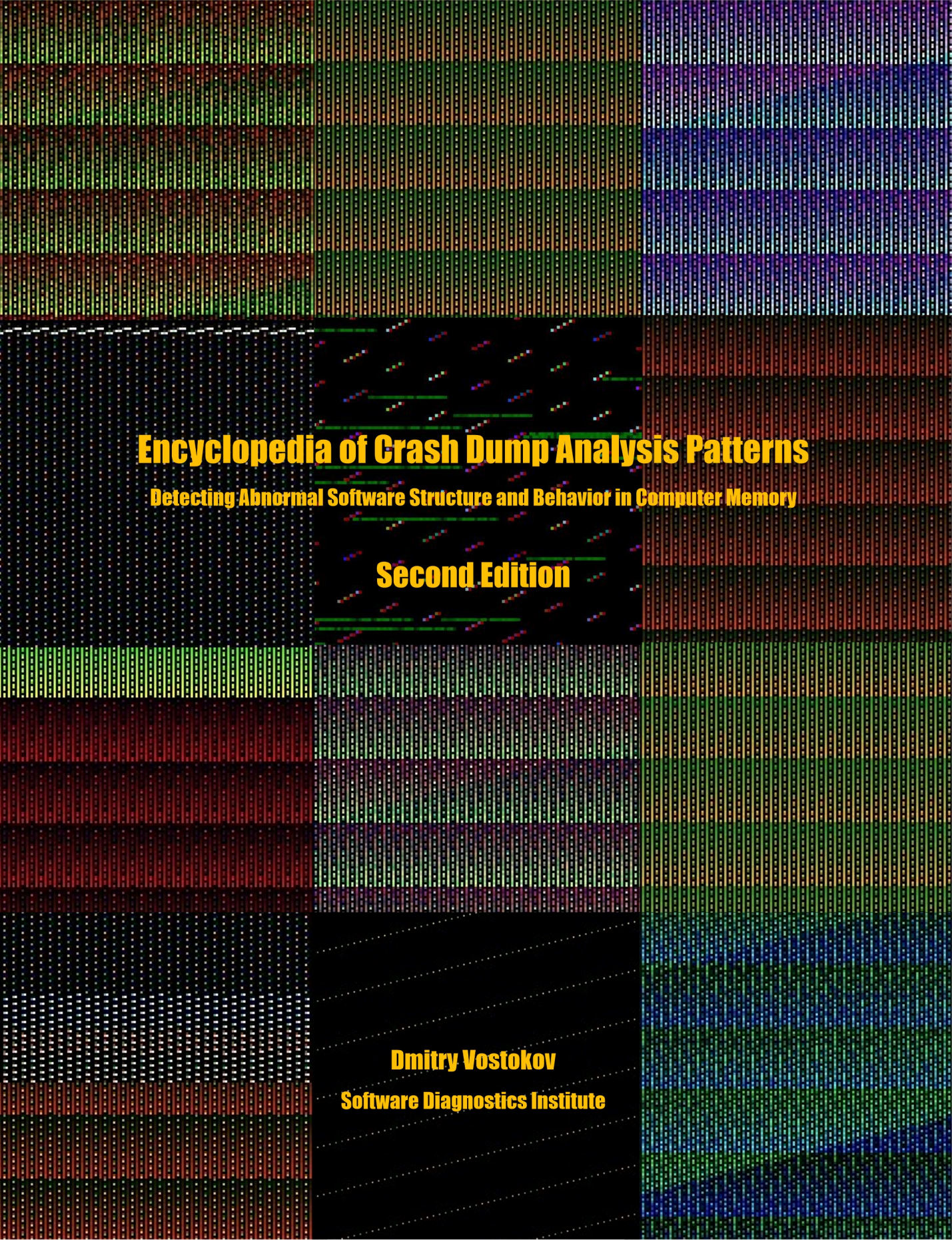
Encyclopedia of Crash Dump Analysis Patterns
Detecting Abnormal Software Structure and Behavior in Computer Memory
Second Edition
Dmitry Vostokov
Software Diagnostics Institute

Published by OpenTask, Republic of Ireland

OpenTask books are available through booksellers and distributors worldwide. For further information or comments send requests to press@opentask.com.

A CIP catalog record for this book is available from the British Library.

ISBN-13: 978-1-908043-83-2 (Paperback)

First printing, 2017

Version 1.08 (March 2017)

Regular Data

This pattern generalizes ASCII and UNICODE-type (00xx00yy) data found in memory to domain-specific data formats such as bitmaps and vector data. An example of the latter could be a sequence of ...0xxx0yyy... (xxx are triplets of hex digits). A typical usage of this pattern is an analysis of corrupt dynamic memory blocks (process heap, kernel pool) where continuity of regular data across block boundary points to a possible **Shared Buffer Overwrite** (page 868).

Corrupt Structure

Typical signals of **Corrupt Structure** include:

- **Regular Data** (page 833) such as ASCII and UNICODE fragments over substructures and pointer areas
- Large values where you expect small and vice versa
- User space address values where we expect kernel space and vice versa
- Malformed and partially zeroed _LIST_ENTRY data (see exercise C3[22] for linked list navigation)
- Memory read errors for pointer dereferences or inaccessible memory indicators (**??**)
- Memory read error at the end of the linked list while traversing structures

```
0: kd> dt _ERESOURCE ffffd0002299f830
ntdll!_ERESOURCE
+0x000 SystemResourcesList : _LIST_ENTRY [ 0xffffc000`07b64800 - 0xffffe000`02a79970 ]
+0x010 OwnerTable       : 0xffffe000`02a79940 _OWNER_ENTRY
+0x018 ActiveCount      : 0n0
+0x01a Flag             : 0
+0x01a ReservedLowFlags : 0 ''
+0x01b WaiterPriority   : 0 ''
+0x020 SharedWaiters    : 0x00000000`00000001 _KSEMAPHORE
+0x028 ExclusiveWaiters : 0xffffe000`02a79a58 _KEVENT
+0x030 OwnerEntry       : _OWNER_ENTRY
+0x040 ActiveEntries    : 0
+0x044 ContentionCount  : 0
+0x048 NumberOfSharedWaiters : 0x7b64800
+0x04c NumberOfExclusiveWaiters : 0xffffc000
+0x050 Reserved2        : (null)
+0x058 Address          : 0xffffd000`2299f870 Void
+0x058 CreatorBackTraceIndex : 0xffffd000`2299f870
+0x060 SpinLock         : 1

0: kd> dt _ERESOURCE ffffd0002299d830
ntdll!_ERESOURCE
+0x000 SystemResourcesList : _LIST_ENTRY [ 0x000001e0`00000280 - 0x00000000`00000004 ]
+0x010 OwnerTable       : 0x00000000`0000003c _OWNER_ENTRY
+0x018 ActiveCount      : 0n0
+0x01a Flag             : 0
+0x01a ReservedLowFlags : 0 ”
+0x01b WaiterPriority   : 0 ”
+0x020 SharedWaiters    : 0x0000003c`000001e0 _KSEMAPHORE
+0x028 ExclusiveWaiters : (null)
+0x030 OwnerEntry       : _OWNER_ENTRY
+0x040 ActiveEntries    : 0
+0x044 ContentionCount  : 0x7f
+0x048 NumberOfSharedWaiters : 0x7f
+0x04c NumberOfExclusiveWaiters : 0x7f
```

[22] Advanced Windows Memory Dump Analysis with Data Structures, Second Edition, http://www.patterndiagnostics.com/advanced-windows-memory-dump-analysis-book

```
+0x050 Reserved2        : 0x00000001`00000001 Void
+0x058 Address          : 0x00000000`00000005 Void
+0x058 CreatorBackTraceIndex : 5
+0x060 SpinLock         : 0
```

However, we need to be sure that we supplied the correct pointer to **dt** WinDbg command. One of the signs that the pointer was incorrect is memory read errors or all zeroes:

```
0: kd> dt _ERESOURCE ffffd000229af830
ntdll!_ERESOURCE
+0x000 SystemResourcesList : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
+0x010 OwnerTable : (null)
+0x018 ActiveCount : 0n0
+0x01a Flag : 0
+0x01a ReservedLowFlags : 0 ''
+0x01b WaiterPriority : 0 ''
+0x020 SharedWaiters : (null)
+0x028 ExclusiveWaiters : (null)
+0x030 OwnerEntry : _OWNER_ENTRY
+0x040 ActiveEntries : 0
+0x044 ContentionCount : 0
+0x048 NumberOfSharedWaiters : 0
+0x04c NumberOfExclusiveWaiters : 0
+0x050 Reserved2 : (null)
+0x058 Address : (null)
+0x058 CreatorBackTraceIndex : 0
+0x060 SpinLock : 0

0: kd> dt _ERESOURCE ffffd00022faf830
ntdll!_ERESOURCE
+0x000 SystemResourcesList : _LIST_ENTRY
+0x010 OwnerTable       : ????
+0x018 ActiveCount      : ??
+0x01a Flag             : ??
+0x01a ReservedLowFlags : ??
+0x01b WaiterPriority   : ??
+0x020 SharedWaiters    : ????
+0x028 ExclusiveWaiters : ????
+0x030 OwnerEntry       : _OWNER_ENTRY
+0x040 ActiveEntries    : ??
+0x044 ContentionCount  : ??
+0x048 NumberOfSharedWaiters : ??
+0x04c NumberOfExclusiveWaiters : ??
+0x050 Reserved2        : ????
+0x058 Address          : ????
+0x058 CreatorBackTraceIndex : ??
+0x060 SpinLock         : ??
Memory read error ffffd00022faf890
```

Dynamic Memory Corruption

Kernel Pool

If kernel pools are corrupt then calls that allocate or free memory result in bugchecks C2 or 19 and in other fewer frequent bugchecks (from Google stats[49]):

BugCheck C2: BAD_POOL_CALLER	1600
BugCheck 19: BAD_POOL_HEADER	434
BugCheck C5: DRIVER_CORRUPTED_EXPOOL	207
BugCheck DE: POOL_CORRUPTION_IN_FILE_AREA	106
BugCheck D0: DRIVER_CORRUPTED_MMPOOL	8
BugCheck D6: DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION	3
BugCheck CD: PAGE_FAULT_BEYOND_END_OF_ALLOCATION	2
BugCheck C6: DRIVER_CAUGHT_MODIFYING_FREED_POOL	0

Bugchecks 0xC2 and 0×19 have parameters in their arguments that tell the type of detected pool corruption. We should refer to WinDbg help for details or use the variant of **!analyze** command where we can supply optional bugcheck arguments:

```
1: kd> !analyze -show c2
BAD_POOL_CALLER (c2)
The current thread is making a bad pool request.  Typically this is at a bad IRQL level or double freeing
the same allocation, etc.
Arguments:
Arg1: 00000000, The caller is requesting a zero byte pool allocation.
Arg2: 00000000, zero.
Arg3: 00000000, the pool type being allocated.
Arg4: 00000000, the pool tag being used.
```

[49] Bugcheck Frequencies, Memory Dump Analysis Anthology, Volume 2, page 429

```
1: kd> !analyze -show 19 2 1 1 1
BAD_POOL_HEADER (19)
The pool is already corrupt at the time of the current request.
This may or may not be due to the caller.
The internal pool links must be walked to figure out a possible cause of
the problem, and then special pool applied to the suspect tags or the driver
verifier to a suspect driver.
Arguments:
Arg1: 00000002, the verifier pool pattern check failed.  The owner has likely corrupted the pool block
Arg2: 00000001, the pool entry being checked.
Arg3: 00000001, size of the block.
Arg4: 00000001, 0.
```

If we enable special pool on suspected drivers, we may get these bugchecks too with the following Google frequency:

BugCheck C1: SPECIAL_POOL_DETECTED_MEMORY_CORRUPTION	59
BugCheck D5: DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL	5
BugCheck CC: PAGE_FAULT_IN_FREED_SPECIAL_POOL	1

Here is one example of nonpaged pool corruption detected during *free* operation with the following **!analyze -v** output:

```
BAD_POOL_HEADER (19)
The pool is already corrupt at the time of the current request.
This may or may not be due to the caller.
The internal pool links must be walked to figure out a possible cause of
the problem, and then special pool applied to the suspect tags or the driver
verifier to a suspect driver.
Arguments:
Arg1: 00000020, a pool block header size is corrupt.
Arg2: a34583b8, The pool entry we were looking for within the page.
Arg3: a34584f0, The next pool entry.
Arg4: 0a270001, (reserved)

POOL_ADDRESS:  a34583b8 Nonpaged pool

PROCESS_NAME:  process.exe

CURRENT_IRQL:  2

STACK_TEXT:
b80a60cc 808927bb nt!KeBugCheckEx+0x1b
b80a6134 80892b6f nt!ExFreePoolWithTag+0x477
b80a6144 b9591400 nt!ExFreePool+0xf
WARNING: Stack unwind information not available. Following frames may be wrong.
b80a615c b957b954 driver+0x38400
b80a617c b957d482 driver+0x22954
b80a61c0 b957abf4 driver+0x24482
b80a6260 b957ccef driver+0x21bf4
b80a62a8 8081df65 driver+0x23cef
b80a62bc f721ac45 nt!IofCallDriver+0x45
b80a62e4 8081df65 fltMgr!FltpDispatch+0x6f
```

```
b80a62f8 b99de70b nt!IofCallDriver+0x45
b80a6308 b99da6ee filter!Dispatch+0xfb
b80a6318 8081df65 filter!dispatch+0x6e
b80a632c b9bdebfe nt!IofCallDriver+0x45
b80a6334 8081df65 2ndfilter!Redirect+0x7ea
b80a6348 b9bd1756 nt!IofCallDriver+0x45
b80a6374 b9bd1860 3rdfilter!PassThrough+0x136
b80a6384 8081df65 3rdfilter!Dispatch+0x80
b80a6398 808f5437 nt!IofCallDriver+0x45
b80a63ac 808ef963 nt!IopSynchronousServiceTail+0x10b
b80a63d0 8088978c nt!NtQueryDirectoryFile+0x5d
b80a63d0 7c8285ec nt!KiFastCallEntry+0xfc
00139524 7c8274eb ntdll!KiFastSystemCallRet
00139528 77e6ba40 ntdll!NtQueryDirectoryFile+0xc
00139830 77e6bb5f kernel32!FindFirstFileExW+0x3d5
00139850 6002665e kernel32!FindFirstFileW+0x16
00139e74 60026363 process+0x2665e
0013a328 60027852 process+0x26363
0013a33c 60035b58 process+0x27852
0013b104 600385ff process+0x35b58
0013b224 612cb643 process+0x385ff
0013b988 612cc109 dll!FileDialog+0xc53
0013bba0 612cb47b dll!FileDialog+0x1719
0013c2c0 7739b6e3 dll!FileDialog+0xa8b
0013c2ec 77395f82 USER32!InternalCallWinProc+0x28
0013c368 77395e22 USER32!UserCallDlgProcCheckWow+0x147
0013c3b0 7739c9c6 USER32!DefDlgProcWorker+0xa8
0013c3d8 7c828536 USER32!__fnDWORD+0x24
0013c3d8 808308f4 ntdll!KiUserCallbackDispatcher+0x2e
b80a66b8 8091d6d1 nt!KiCallUserMode+0x4
b80a6710 bf8a2622 nt!KeUserModeCallback+0x8f
b80a6794 bf8a2517 win32k!SfnDWORD+0xb4
b80a67dc bf8a13d9 win32k!xxxSendMessageToClient+0x133
b80a6828 bf85ae67 win32k!xxxSendMessageTimeout+0x1a6
b80a684c bf8847a1 win32k!xxxWrapSendMessage+0x1b
b80a6868 bf8c1459 win32k!NtUserfnNCDESTROY+0x27
b80a68a0 8088978c win32k!NtUserMessageCall+0xc0
b80a68a0 7c8285ec nt!KiFastCallEntry+0xfc
0013c3d8 7c828536 ntdll!KiFastSystemCallRet
0013c3d8 808308f4 ntdll!KiUserCallbackDispatcher+0x2e
b80a6b7c 8091d6d1 nt!KiCallUserMode+0x4
b80a6bd4 bf8a2622 nt!KeUserModeCallback+0x8f
b80a6c58 bf8a23a0 win32k!SfnDWORD+0xb4
b80a6ca0 bf8a13d9 win32k!xxxSendMessageToClient+0x118
b80a6cec bf85ae67 win32k!xxxSendMessageTimeout+0x1a6
b80a6d10 bf8c148c win32k!xxxWrapSendMessage+0x1b
b80a6d40 8088978c win32k!NtUserMessageCall+0x9d
b80a6d40 7c8285ec nt!KiFastCallEntry+0xfc
0013f474 7c828536 ntdll!KiFastSystemCallRet
0013f4a0 7739d1ec ntdll!KiUserCallbackDispatcher+0x2e
0013f4dc 7738cf29 USER32!NtUserMessageCall+0xc
0013f4fc 612d3276 USER32!SendMessageA+0x7f
0013f63c 611add41 dll!SubWindow+0x3dc6
0013f658 7739b6e3 dll!SetWindowText+0x37a1
0013f684 7739b874 USER32!InternalCallWinProc+0x28
0013f6fc 7739ba92 USER32!UserCallWinProcCheckWow+0x151
```

```
0013f764 7739bad0 USER32!DispatchMessageWorker+0x327
0013f774 61221ca8 USER32!DispatchMessageW+0xf
0013f7e0 0040156d dll!MainLoop+0x2c8
0013ff24 00401dfa process+0x156d
0013ffc0 77e6f23b process+0x1dfa
0013fff0 00000000 kernel32!BaseProcessStart+0x23

MODULE_NAME: driver

IMAGE_NAME:  driver.sys
```

We see that WinDbg pointed to *driver.sys* by using a procedure described in Component Identification article[50].

Any OS component could corrupt the pool prior to the detection as the bugcheck description says: "The pool is already corrupt at the time of the current request.". What other evidence can reinforce our belief in *driver.sys*? Let's look at our pool entry tag first:

```
1: kd> !pool a34583b8
Pool page a34583b8 region is Nonpaged pool
 a3458000 size:  270 previous size:    0  (Allocated)  Thre (Protected)
 a3458270 size:   10 previous size:  270  (Free)       RxIr
 a3458280 size:   40 previous size:   10  (Allocated)  Vadl
 a34582c0 size:   98 previous size:   40  (Allocated)  File (Protected)
 a3458358 size:    8 previous size:   98  (Free)       Vadl
 a3458360 size:   50 previous size:    8  (Allocated)  Gsem
 a34583b0 size:    8 previous size:   50  (Free)       CcSc
*a34583b8 size:  138 previous size:    8  (Allocated) *DRIV
  Owning component : Unknown (update pooltag.txt)
a34584f0 is not a valid large pool allocation, checking large session pool…
a34584f0 is freed (or corrupt) pool
Bad allocation size @a34584f0, zero is invalid

***
*** An error (or corruption) in the pool was detected;
*** Attempting to diagnose the problem.
***
*** Use !poolval a3458000 for more details.
***

Pool page [ a3458000 ] is __inVALID.

Analyzing linked list...
[ a34583b8 --> a34583d8 (size = 0x20 bytes)]: Corrupt region
[ a34583f8 --> a34585e8 (size = 0x1f0 bytes)]: Corrupt region

Scanning for single bit errors...

None found
```

[50] Component Identification, Memory Dump Analysis Anthology, Volume 1, page 46

We see that the tag is DRIV, and we know either from the association or from similar problems in the past that it belongs to *driver.sys*. Let's dump our pool entry contents to see if there are any symbolic hints in it:

```
1: kd> dps a34583b8
a34583b8 0a270001
a34583bc 5346574e
a34583c0 00000000
a34583c4 00000000
a34583c8 b958f532 driver+0×36532
a34583cc a3471010
a34583d0 0000012e
a34583d4 00000001
a34583d8 00041457
a34583dc 05af0026
a34583e0 00068002
a34583e4 7b9ec6f5
a34583e8 ffffff00
a34583ec 73650cff
a34583f0 7461445c
a34583f4 97a10061
a34583f8 ff340004
a34583fc c437862a
a3458400 6a000394
a3458404 00000038
a3458408 00000000
a345840c bf000000
a3458410 bf0741b5
a3458414 f70741b5
a3458418 00000000
a345841c 00000000
a3458420 00000000
a3458424 00000000
a3458428 05000000
a345842c 34303220
a3458430 31323332
a3458434 ff322d36
```

Indeed we see that the possible code pointer *driver+0×36532* and the code around this address look normal:

```
3: kd> .asm no_code_bytes
Assembly options: no_code_bytes

3: kd> u b958f532
driver+0x36532:
b958f532 push    2Ch
b958f534 push    offset driver+0x68d08 (b95c1d08)
b958f539 call    driver+0x65c50 (b95bec50)
b958f53e mov     byte ptr [ebp-19h],0
b958f542 and     dword ptr [ebp-24h],0
b958f546 call    dword ptr [driver+0x65f5c (b95bef5c)]
b958f54c mov     ecx,dword ptr [ebp+0Ch]
b958f54f cmp     eax,ecx
```

```
3: kd> ub b958f532
driver+0x36528:
b958f528 leave
b958f529 ret     18h
b958f52c int     3
b958f52d int     3
b958f52e int     3
b958f52f int     3
b958f530 int     3
b958f531 int     3
```

Comments

Here is one of the asked questions.

Q. I have the same problem with BAD_POOL_HEADER; I see the pool header e173c1d8 is corrupt and marked as *Ppen. Could you help me to know what is that?

```
BAD_POOL_HEADER (19)
The pool is already corrupt at the time of the current request.
This may or may not be due to the caller.
The internal pool links must be walked to figure out a possible cause of
the problem, and then special pool applied to the suspect tags or the driver
verifier to a suspect driver.
Arguments:
Arg1: 00000020, a pool block header size is corrupt.
Arg2: e173c1d8, The pool entry we were looking for within the page.
Arg3: e173c1f8, The next pool entry.
Arg4: 0c040404, (reserved)

Debugging Details:
——————


BUGCHECK_STR: 0×19_20

POOL_ADDRESS: e173c1d8

CUSTOMER_CRASH_COUNT: 6

DEFAULT_BUCKET_ID: COMMON_SYSTEM_FAULT

PROCESS_NAME: System

LOCK_ADDRESS: 8055b4e0 — (!locks 8055b4e0)

Resource @ nt!PiEngineLock (0×8055b4e0) Available

WARNING: SystemResourcesList->Flink chain invalid. Resource may be corrupted, or already deleted.

WARNING: SystemResourcesList->Blink chain invalid. Resource may be corrupted, or already deleted.
```

```
1 total locks

PNP_TRIAGE:
Lock address : 0×8055b4e0
Thread Count : 0
Thread address: 0×00000000
Thread wait : 0×0

LAST_CONTROL_TRANSFER: from 8054b583 to 804f9f33

STACK_TEXT:
f79f392c 8054b583 00000019 00000020 e173c1d8 nt!KeBugCheckEx+0×1b
f79f397c 8058fc05 e173c1e0 00000000 00000000 nt!ExFreePoolWithTag+0×2a3
f79f39dc 8059161d 85804dd0 e10f29a8 f79f3a48 nt!PipMakeGloballyUniqueId+0×3a9
f79f3ad0 8059222b 8593a7c8 861d33e8 8593a7c8 nt!PipProcessNewDeviceNode+0×185
f79f3d24 805927fa 8593a7c8 00000001 00000000 nt!PipProcessDevNodeTree+0×16b
f79f3d54 804f698e 00000003 8055b5c0 8056485c nt!PiRestartDevice+0×80
f79f3d7c 8053876d 00000000 00000000 863c1da8 nt!PipDeviceActionWorker+0×168
f79f3dac 805cff64 00000000 00000000 00000000 nt!ExpWorkerThread+0xef
f79f3ddc 805460de 8053867e 00000001 00000000 nt!PspSystemThreadStartup+0×34
00000000 00000000 00000000 00000000 00000000 nt!KiThreadStartup+0×16

STACK_COMMAND: kb

FOLLOWUP_IP:
nt!ExFreePoolWithTag+2a3
8054b583 8b45f8 mov eax,dword ptr [ebp-8]

SYMBOL_STACK_INDEX: 1

SYMBOL_NAME: nt!ExFreePoolWithTag+2a3

FOLLOWUP_NAME: MachineOwner

MODULE_NAME: nt

IMAGE_NAME: ntkrpamp.exe

DEBUG_FLR_IMAGE_TIMESTAMP: 4802516a

FAILURE_BUCKET_ID: 0×19_20_nt!ExFreePoolWithTag+2a3

BUCKET_ID: 0×19_20_nt!ExFreePoolWithTag+2a3

Followup: MachineOwner
———
```

```
0: kd> !pool e173c1d8
Pool page e173c1d8 region is Unknown
e173c000 size: 28 previous size: 0 (Allocated) CMVa
e173c028 size: 8 previous size: 28 (Free) 0…
e173c030 size: 10 previous size: 8 (Allocated) MmSt
e173c040 size: 48 previous size: 10 (Allocated) ScSh
e173c088 size: 68 previous size: 48 (Allocated) ScNc
e173c0f0 size: 8 previous size: 68 (Free) Ntf0
e173c0f8 size: 10 previous size: 8 (Allocated) ObDi
e173c108 size: 28 previous size: 10 (Allocated) CMVa
e173c130 size: 80 previous size: 28 (Allocated) IoNm
e173c1b0 size: 8 previous size: 80 (Free) Sect
e173c1b8 size: 20 previous size: 8 (Allocated) CMVa
*e173c1d8 size: 20 previous size: 20 (Allocated) *Ppen
Pooltag Ppen : routines to perform device enumeration, Binary : nt!pnp
GetUlongFromAddress: unable to read from 80565d50
e173c1f8 is not a valid small pool allocation, checking large pool…
unable to get pool big page table - either wrong symbols or pool tagging is disabled
e173c1f8 is freed (or corrupt) pool
Bad previous allocation size @e173c1f8, last size was 4

***
*** An error (or corruption) in the pool was detected;
*** Pool Region unknown (0xFFFFFFFFE173C1F8)
***
*** Use !poolval e173c000 for more details.
***

0: kd> !poolval e173c000
Pool page e173c000 region is Unknown

Validating Pool headers for pool page: e173c000

Pool page [ e173c000 ] is __inVALID.

Analyzing linked list…
[ e173c1d8 -> e173c2c0 (size = 0xe8 bytes)]: Corrupt region

Scanning for single bit errors…

None found

0: kd> dps e173c1d8
e173c1d8 0c040404
e173c1dc 6e657050
e173c1e0 00260032
e173c1e4 00370061
e173c1e8 00360035
e173c1ec 00370035
e173c1f0 00260032
e173c1f4 004e0030
e173c1f8 00520054
e173c1fc 004c004f
e173c200 002e0053
```

```
[...]
e173c214 00360032
e173c218 00300030
e173c21c 0c12040a
e173c220 e24e4d43
e173c224 00010001
e173c228 7224c689
e173c22c e17b53a4
e173c230 23230077
e173c234 4448233f
[...]
e173c24c 44264431
e173c250 375f5645
e173c254 26353036
```

A. Pooltag Ppen : routines to perform device enumeration, Binary : nt!pnp

Ppen is from PnP manager

If we search for this stack trace frame:

```
PipMakeGloballyUniqueId+0×3a9
```

we find the discussion of a similar BSOD and perhaps a solution[51].

Also, please note that the content of the address e173c1f8 and the content of nearby addresses are covered by UNICODE **Regular Data** (page 833). We may want to check these **String Hints** (page 960).

Pool corruption may also cause access violations in pool management with general bugchecks such as KMODE_EXCEPTION_NOT_HANDLED (1e):

```
0: kd> k
 # Child-SP RetAddr Call Site
 00 fffff802`d2afe568 fffff802`d103db86 nt!KeBugCheckEx
 01 fffff802`d2afe570 fffff802`d0fc652d nt!KiFatalExceptionHandler+0×22
 02 fffff802`d2afe5b0 fffff802`d0e83139 nt!RtlpExecuteHandlerForException+0xd
 03 fffff802`d2afe5e0 fffff802`d0e815a8 nt!RtlDispatchException+0×429
 04 fffff802`d2afece0 fffff802`d0fcb0c2 nt!KiDispatchException+0×144
 05 fffff802`d2aff3c0 fffff802`d0fc957d nt!KiExceptionDispatch+0xc2
 06 fffff802`d2aff5a0 fffff802`d10af119 nt!KiGeneralProtectionFault+0xfd
 *** ERROR: Module load completed but symbols could not be loaded for vmci.sys
 07 fffff802`d2aff730 fffff800`84456159 nt!ExAllocatePoolWithTag+0×7c9
 08 fffff802`d2aff810 fffff800`84457131 vmci+0×6159
 09 fffff802`d2aff840 fffff800`8445398b vmci+0×7131
 0a fffff802`d2aff890 fffff802`d0eec3c0 vmci+0×398b
 0b fffff802`d2aff8c0 fffff802`d0eebad9 nt!KiExecuteAllDpcs+0×270
 0c fffff802`d2affa10 fffff802`d0fc323a nt!KiRetireDpcList+0xe9
 0d fffff802`d2affc60 00000000`00000000 nt!KiIdleLoop+0×5a
```

[51] http://social.microsoft.com/Forums/en/whssoftware/thread/98b381be-edff-43fb-b1d9-5307e6204733

Small Value

Sometimes we see the so-called **Small Values** in memory (such as on raw stack) or in CPU registers which can be ASCII or UNICODE value, some ID or even a handle. When in aggregates they can form a certain **Semantic Structure** (page 860) such as a PID.TID example or **Regular Data** (page 833) pattern. Here we illustrate a handle example (also an example of **Wait Chain** analysis in user space, page 1092):

```
0:000> kv
Child-SP          RetAddr           : Args to
Child                                                         : Call Site
00000000`0016de78 000007fe`fcf010dc : 00000000`02c79fa0 00000000`08c3faf0 00000000`021551f0
00000000`08c3fb00 : ntdll!NtWaitForSingleObject+0xa
00000000`0016de80 000007fe`f90e6d7f : 00000000`10b40010 00000000`10b40010 00000000`00000000
00000000`000007e0 : KERNELBASE!WaitForSingleObjectEx+0×79
[...]

0:000> !handle 00000000`000007e0 ff
Handle 00000000000007d0
  Type          Thread
  Attributes    0
  GrantedAccess 0x1fffff:
        Delete,ReadControl,WriteDac,WriteOwner,Synch
        Terminate,Suspend,Alert,GetContext,SetContext,SetInfo,QueryInfo,SetToken,
Impersonate,DirectImpersonate
  HandleCount   5
  PointerCount  9
  Name          <none>
  Object specific information
    Thread Id   278c.a58
    Priority    13
    Base Priority 0

0:000> ~~[a58]s
ntdll!NtWaitForMultipleObjects+0xa:
00000000`770c186a c3              ret

0:002> kv
Child-SP          RetAddr           : Args to
Child                                                         : Call Site
00000000`0f6af758 000007fe`fcf01430 : 00000000`00000025 00000000`00000000 00000000`00000000
000007fe`e35a1fb0 : ntdll!NtWaitForMultipleObjects+0xa
00000000`0f6af760 00000000`76e61220 : 00000000`0f6af8a8 00000000`0f6af890 00000000`00000000
00000000`00000000 : KERNELBASE!WaitForMultipleObjectsEx+0xe8
[...]

0:002> dp 00000000`0f6af890 L4
00000000`0f6af890  00000000`00000dbc 00000000`000007c0
00000000`0f6af8a0  00000000`00000000 00000000`00000000
```

```
0:002> !handle dbc ff
Handle 0000000000000dbc
  Type          Thread
  Attributes    0
  GrantedAccess 0x1fffff:
         Delete,ReadControl,WriteDac,WriteOwner,Synch
         Terminate,Suspend,Alert,GetContext,SetContext,SetInfo,QueryInfo,SetToken,
Impersonate,DirectImpersonate
  HandleCount   2
  PointerCount  4
  Name          <none>
  Object specific information
    Thread Id   278c.24ac
    Priority    14
    Base Priority 0

0:002> !handle 7c0 ff
Handle 00000000000007c0
  Type          Thread
  Attributes    0
  GrantedAccess 0x1fffff:
         Delete,ReadControl,WriteDac,WriteOwner,Synch
         Terminate,Suspend,Alert,GetContext,SetContext,SetInfo,QueryInfo,SetToken,
Impersonate,DirectImpersonate
  HandleCount   2
  PointerCount  4
  Name          <none>
  Object specific information
    Thread Id   278c.628
    Priority    14
    Base Priority 0
```

Comments

Sometimes a small value can be coincidentally a valid handle value and at the same time a valid thread or process id.