3/18/2015

Assembly-Level Debugging

Alexandro Sánchez Bach

Example

- Issue found in a multi-threaded algorithm.
- Random worker threads crashing randomly at completely unrelated points.

Call Stack

Name	Language
ntdll.dll!0000000077b79997()	Unknown
ntdll.dll!0000000077b76a1f()	Unknown
ntdll.dll!0000000077b76978()	Unknown
kernel32.dll!0000000077a259f5()	Unknown
ntdll.dll!0000000077b5c541()	Unknown

Call Stack

Name	Language
sysfer.dll!000000007563af22()	Unknown
ntdll.dll!0000000077b5b8b4()	Unknown
ntdll.dll!0000000077b76aab()	Unknown
ntdll.dll!0000000077b76978()	Unknown
kernel32.dll!0000000077a259f5()	Unknown
ntdll.dll!0000000077b5c541()	Unknown

Call Stack

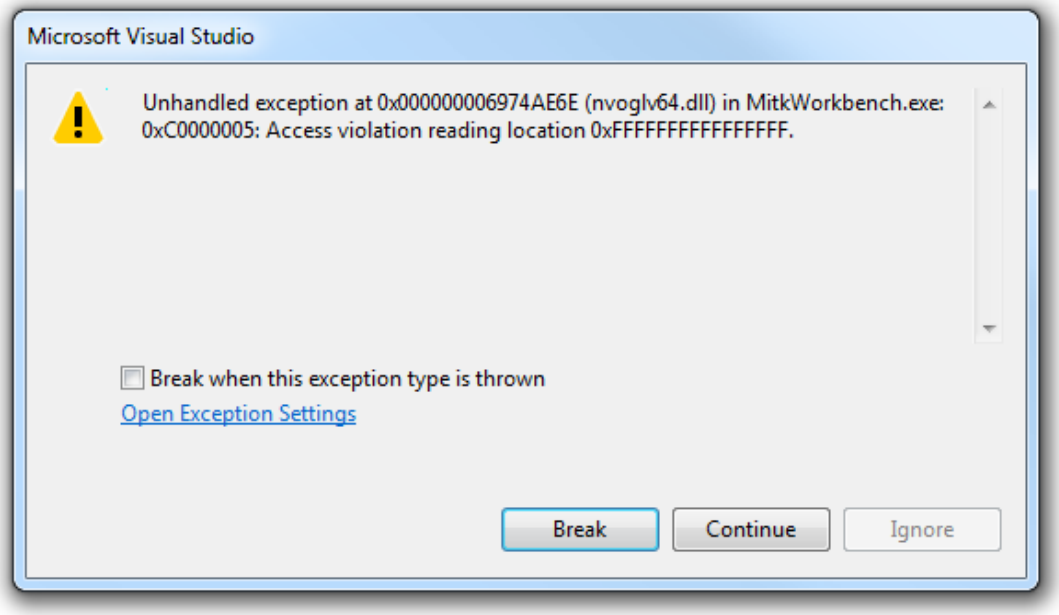
Name	Language
ntdll.dll!0000000077bf40cf()	Unknown
ntdll.dll!0000000077bf4746()	Unknown
ntdll.dll!0000000077bf5952()	Unknown
ntdll.dll!0000000077bf7604()	Unknown
ntdll.dll!0000000077b8fb2a()	Unknown
ntdll.dll!0000000077b834d8()	Unknown
ntdll.dll!0000000077c070dd()	Unknown
ntdll.dll!0000000077bcb5aa()	Unknown
ntdll.dll!0000000077b834d8()	Unknown
msvcr110d.dll!_heap_alloc_base(unsigned __int64 size) Line 57	C
msvcr110d.dll!_heap_alloc_dbg_impl(unsigned __int64 nSize, int nBlockUse, const char * szFile	C++
msvcr110d.dll!_nh_malloc_dbg_impl(unsigned __int64 nSize, int nhFlag, int nBlockUse, const	C++
msvcr110d.dll!_calloc_dbg_impl(unsigned __int64 nNum, unsigned __int64 nSize, int nBlockU	C++
msvcr110d.dll!_calloc_dbg(unsigned __int64 nNum, unsigned __int64 nSize, int nBlockUse, co	C++
msvcr110d.dll!__CRTDLL_INIT(void * hDllHandle, unsigned long dwReason, void * lpreserved)	C
msvcr110d.dll!_CRTDLL_INIT(void * hDllHandle, unsigned long dwReason, void * lpreserved)	C
ntdll.dll!0000000077b5c78c()	Unknown
ntdll.dll!0000000077b5c44f()	Unknown
ntdll.dll!0000000077b5c34e()	Unknown

Call Stack

Name	Language
ntdll.dll!0000000077bf40cf()	Unknown
ntdll.dll!0000000077bf4746()	Unknown
ntdll.dll!0000000077bf5952()	Unknown
ntdll.dll!0000000077bf7604()	Unknown
ntdll.dll!0000000077b914e5()	Unknown
ntdll.dll!0000000077b840fd()	Unknown
ntdll.dll!0000000077bf9be9()	Unknown
ntdll.dll!0000000077b9db50()	Unknown
ntdll.dll!0000000077b840fd()	Unknown
kernel32.dll!0000000077a31a4a()	Unknown
nvoglv64.dll!0000000069c941c4()	Unknown
nvoglv64.dll!0000000069c9af80()	Unknown
ntdll.dll!0000000077b5b8b4()	Unknown
ntdll.dll!0000000077b76aab()	Unknown
ntdll.dll!0000000077b76978()	Unknown
kernel32.dll!0000000077a259f5()	Unknown
ntdll.dll!0000000077b5c541()	Unknown

Example

- Our IDE is not helping much either.

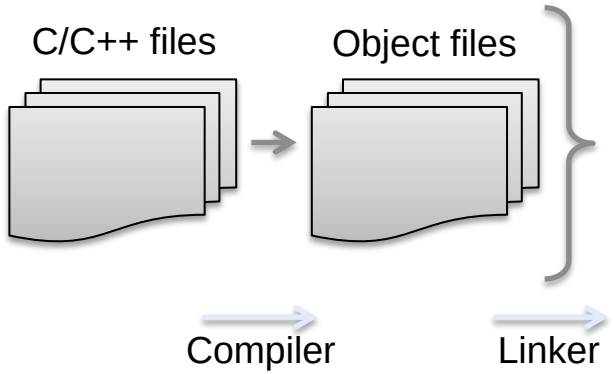


- No source code context.
- No time/motivation to make wild guesses about what happened.

Assembly-Level Debugging

- Known in *Visual Studio* as **Address-Level Debugging**, and provide access to:
  - Memory windows
  - Disassembly panel
  - CPU registers panel
  - Memory write/execute breakpoints
- Additionally, other solutions (e.g. *OllyDbg*) provide breakpoints on memory reading attempts as well.
- Alternatives (discussed later):
  - *GDB*
  - *Xcode*

Assembly



Disassembly

```
...                 ...    ...
0000000077B833E9    shr    rcx,4
0000000077B833ED    cmp    rcx,rax
0000000077B833F0    jae    0000000077B84B29
0000000077B833F6    mov    r8d,dword ptr [rdx+8]
0000000077B833FA    dec    r8d
0000000077B833FD    cmp    rcx,r8
0000000077B83400    jae    0000000077B834AA
0000000077B83406    mov    eax,dword ptr [rdx+18h]
0000000077B83409    sub    rcx,rax
0000000077B8340C    cmp    dword ptr [rdx+0Ch],ebx
0000000077B8340F    je     0000000077B83414
0000000077B83411    xor    rax, rax
0000000077B83414    mov    qword ptr [rcx + 8],rax
0000000077B83418    lea    rbx,[rax+rcx*8]
0000000077B8341C    test   rbx,rbx
0000000077B8341F    je     0000000077B834B3
0000000077B83425    mov    rcx,qword ptr [rbx+8]
0000000077B83429    test   cl,1
0000000077B8342C    je     0000000077B834B3
0000000077B83432    dec    rcx
0000000077B83435    mov    rdx,rsi
0000000077B83438    mov    dword ptr [rsp+110h],2
0000000077B83443    call   0000000077B84ED0
0000000077B83448    test   rax,rax
0000000077B8344B    je     0000000077B834B3
0000000077B8344D    mov    rdi,rax
0000000077B83450    test   bpl,8
0000000077B83454    jne    0000000077B85921
0000000077B8345A    test   rdi,rdi
0000000077B8345D    je     0000000077BCBACC
0000000077B83463    test   r14d,r14d
0000000077B83466    jne    0000000077BCBBC2
0000000077B8346C    cmp    byte ptr [7FFE0380h],0
0000000077B83474    mov    r15,qword ptr [rsp+0D0h]
0000000077B8347C    mov    r14,qword ptr [rsp+0D8h]
0000000077B83484    mov    r13,qword ptr [rsp+0E0h]
0000000077B8348C    mov    rbx,qword ptr [rsp+118h]
0000000077B83494    jne    0000000077BCBC32
0000000077B8349A    mov    rax,rdi
0000000077B8349D    add    rsp,0E8h
0000000077B834A4    pop    r12
0000000077B834A6    pop    rdi
0000000077B834A7    pop    rsi
0000000077B834A8    pop    rbp
...                 ...    ...
```

```
xor   rax, rax
mov   qword ptr [rcx + 8], rax
```

Registers

General Purpose registers:
RAX, RBX, RCX, RDX, RSP, RBP, RSI, RDI, R8, R9, R10, R11, R12, R13, R14, R15

FPU/SSE/AVX extension registers

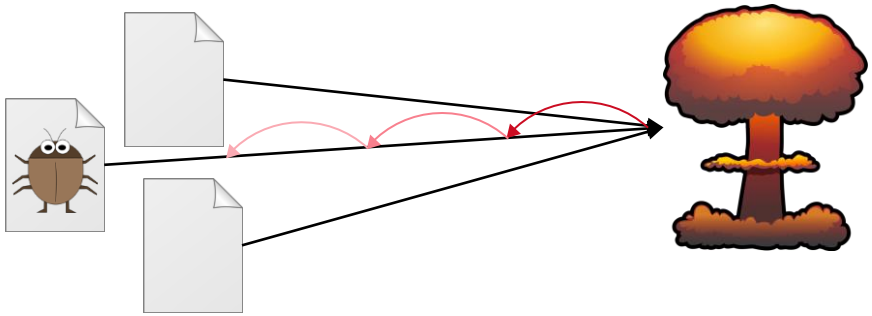
Segment registers

Memory

```
...                 ...                     ...
0x000000000699F720  1b 00 10 00 80 1f 00 00  ....€...
0x000000000699F728  33 00 00 00 00 00 00 00  3.......
0x000000000699F730  00 00 2b 00 00 02 00 00  ..+.....
0x000000000699F738  00 00 00 00 00 00 00 00  ........
0x000000000699F748  e0 44 c3 77 00 00 00 00  àDÃw....
0x000000000699F750  00 00 00 00 00 00 00 00  ........
0x000000000699F758  3b 0f b5 77 00 00 00 00  ;.µw....
0x000000000699F760  70 57 bf 03 00 00 02 00  pW¿.....
0x000000000699F768  00 00 00 00 00 00 01 00  ........
0x000000000699F778  30 d3 dd 03 00 00 00 00  0ÓÝ.....
...                 ...                     ...
```

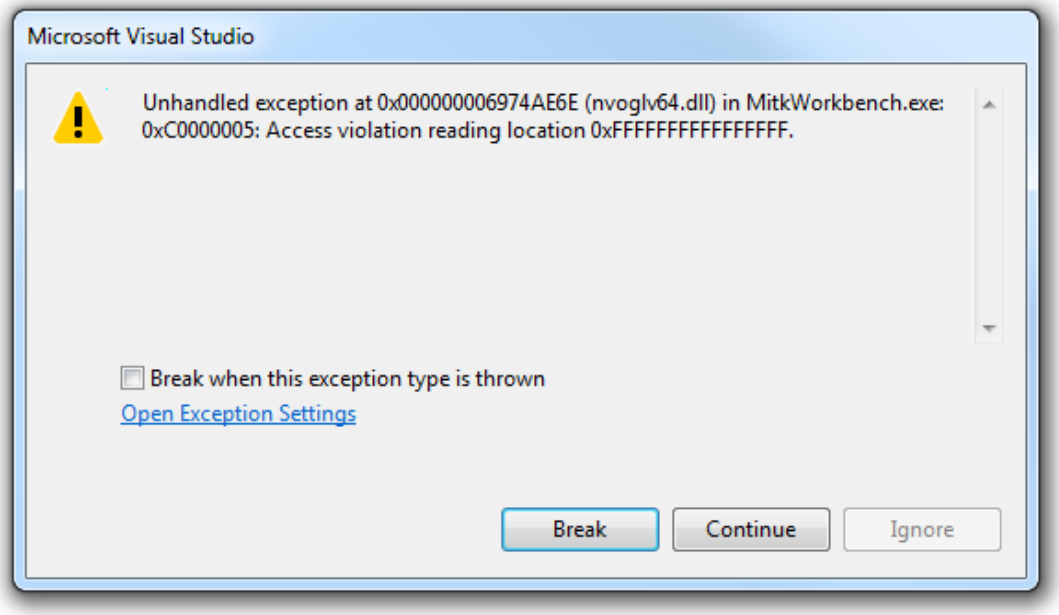
Motivation

- Spotting bugs where the information provided by the IDE's standard debugging tools is not enough.
- Non-trivial bugs in multi-threaded algorithms.
- "Random" bugs.
- Trashed stack or heap overflow related bugs.
- Tracking back the cause of the bug from its effects.



Example

- Back to our MITK example.



Example

- Current instruction in the disassembly panel.

```
Address: 000000007563af22()
Viewing Options
000000007563AF1A  push   rdi
000000007563AF1B  sub    rsp,20h
000000007563AF1F  mov    rbx,rcx
000000007563AF22  mov    rcx,qword ptr [rcx+38h]
000000007563AF26  test   rcx,rcx
000000007563AF29  je     000000007563AF30
```

- Current state of the thread.

```
Registers
RAX = 000000007563AF0C RBX = 437F00003DA8A8A3 RCX = 437F00003DA8A8A3
RDX = 0000000000000000 RSI = 0000000000000010 RDI = 0000000000000001
R8  = 000000000B63FAB8 R9  = 0000000000000000 R10 = 0000000000000000
R11 = 00000000002A0DD0 R12 = 000000000A7AD8F0 R13 = 000007FFFFFDF000
R14 = 0000000000000002 R15 = 0000000000000000 RIP = 000000007563AF22
RSP = 000000000B63F950 RBP = 0000000000000007 EFL = 00010204

0x000000003da8a8db = 0000000000000000
```

Example

- Current instruction in the disassembly panel.

```
Address: 000000007563af22()
Viewing Options
000000007563AF1A  push   rdi
000000007563AF1B  sub    rsp,20h
000000007563AF1F  mov    rbx,rcx
000000007563AF22  mov    rcx,qword ptr [rcx+38h]
000000007563AF26  test   rcx,rcx
000000007563AF29  je     000000007563AF30
```

- Current state of the thread.

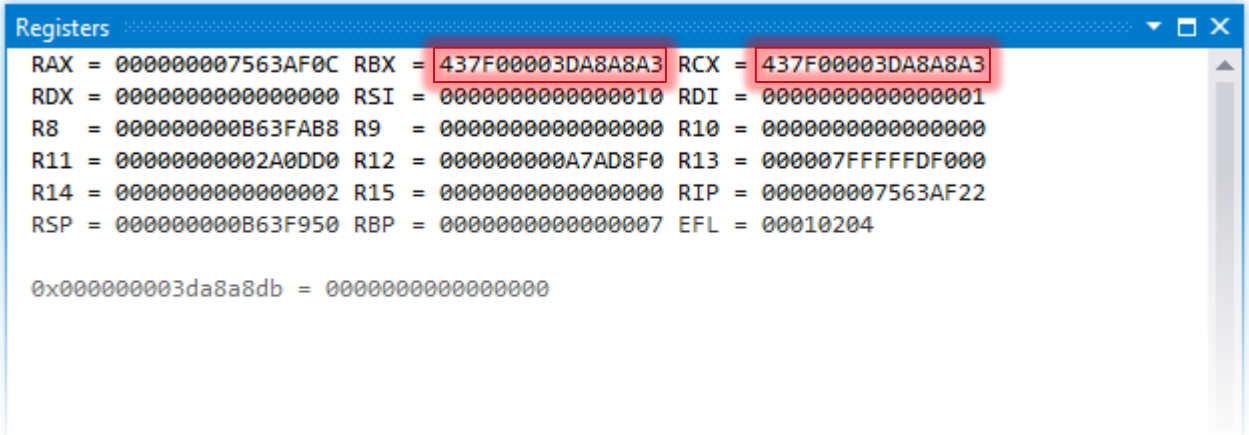
```
Registers
RAX = 000000007563AF0C RBX = 437F00003DA8A8A3 RCX = 437F00003DA8A8A3
RDX = 0000000000000000 RSI = 0000000000000010 RDI = 0000000000000001
R8  = 000000000B63FAB8 R9  = 0000000000000000 R10 = 0000000000000000
R11 = 00000000002A0DD0 R12 = 000000000A7AD8F0 R13 = 000007FFFFFDF000
R14 = 0000000000000002 R15 = 0000000000000000 RIP = 000000007563AF22
RSP = 000000000B63F950 RBP = 0000000000000007 EFL = 00010204

0x000000003da8a8db = 0000000000000000
```

Example

- *Guessing** data types from their raw representation.
  - Floats: `~0x40000000` or `~0xC0000000`
  - Doubles: `~0x4000000000000000` or `~0xC000000000000000`
  - Addresses: `0x0000XXXXXXXXXXXX` or `0xFFFF000000000000`
  - Integers: `0xXXXXXXXXXXXXXXXX`
  - Uninit. data**: `0xCCCCCCCC` (stack) or `0xCDCDCDCD` (heap)

- Current state of the thread.



* **Disclaimer**: This is not science, but just a shortcut.
** Only on debug configuration. Magic values for Visual Studio only.

Example

- Where did the RCX value come from?

```
push    rbp
...
mov     r12,rax
...
mov     rcx,qword ptr[rax+18h]
...
mov     rax,rbx
...
call    000000007563AF1A
...
pop rpb
ret
```

Address: 000000007563af22()

Viewing Options

```
000000007563AF1A  push   rdi
000000007563AF1B  sub    rsp,20h
000000007563AF1F  mov    rbx,rcx
000000007563AF22  mov    rcx,qword ptr [rcx+38h]
000000007563AF26  test   rcx,rcx
000000007563AF29  je     000000007563AF30
000000007563AF2B  call   0000000075633E08
000000007563AF30  mov    rcx,qword ptr [rbx+48h]
000000007563AF34  test   rcx,rcx
```

Example

- The base address is still preserved in `R12`.
- The value that is written into RCX is at the address: `R12 + 0x18`.

```
Registers
RAX = 000000007563AF0C RBX = 437F00003DA8A8A3 RCX = 437F00003DA8A8A3
RDX = 0000000000000000 RSI = 0000000000000010 RDI = 0000000000000001
R8  = 000000000B63FAB8 R9  = 0000000000000000 R10 = 0000000000000000
R11 = 00000000002A0DD0 R12 = 000000000A7AD8F0 R13 = 000007FFFFFDF000
R14 = 0000000000000002 R15 = 0000000000000000 RIP = 000000007563AF22
RSP = 000000000B63F950 RBP = 0000000000000007 EFL = 00010204

0x000000003da8a8db = 0000000000000000
```

Example

- Current state of the memory:

+ 0x18

```
Memory 1
Address: 0x000000000A7AD8F0
0x000000000A7AD8F0  ca dc 5c 3e eb e0 60 3e  ÊÜ\>ëà`>
0x000000000A7AD8F8  eb e0 60 3d 00 00 7f 43  ëà`=...C
0x000000000A7AD900  83 8a 8a 3e 82 8e 8e 3e  ƒŠŠ>.ŽŽ>
0x000000000A7AD908  a3 a8 a8 3d 00 00 7f 43  £¨¨=...C
0x000000000A7AD910  a3 92 92 3e a2 9a 9a 3e  £’’>¢šš>
0x000000000A7AD918  b2 a0 20 3e 00 00 7f 43  .  >...C
0x000000000A7AD920  db de de 3e e9 f0 70 3e  ÛÞÞ>éðp>
0x000000000A7AD928  91 9c 1c 3e 00 00 7f 43  ‘œ.>...C
0x000000000A7AD930  c6 bf 3f 3f 8b 8b 0b 3f  Æ¿??...?
```

Little endianness

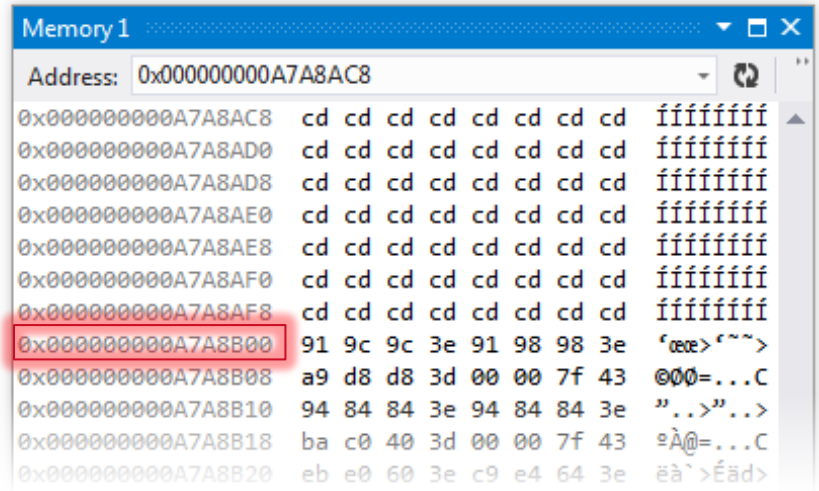
- Current state of the thread:

```
Registers
RAX = 000000007563AF0C RBX = 437F00003DA8A8A3 RCX = 437F00003DA8A8A3
RDX = 0000000000000000 RSI = 0000000000000010 RDI = 0000000000000001
R8  = 000000000B63FAB8 R9  = 0000000000000000 R10 = 0000000000000000
R11 = 00000000002A0DD0 R12 = 000000000A7AD8F0 R13 = 000007FFFFFDF000
R14 = 0000000000000002 R15 = 0000000000000000 RIP = 000000007563AF22
RSP = 000000000B63F950 RBP = 0000000000000007 EFL = 00010204

0x000000003da8a8db = 0000000000000000
```

Example

- Current state of the memory:



Uninitialized heap buffer

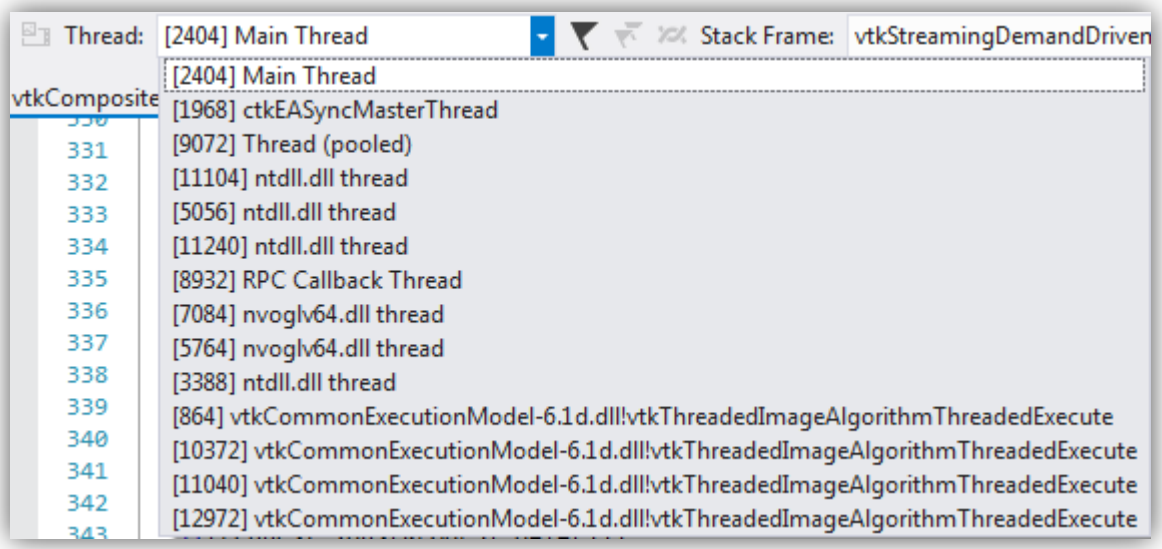
Floating-point data

. . .

Example

- If all pointers to the overflown buffer `0x0A7A8B00` had disappeared, there would be a memory leak.

- Therefore, this address should be still accessible inside *some* function(s) in the call stack of *some* thread(s).



Example

- Since the address was allocated by our own code, we have debugging symbols and the buffer is shown as:

Value	Type
...	...
0x000000000A7A8B00 {...}	float *
...	...

- We are no longer in a „*source code*“-less situation.
- Allocation happens in a fixed thread.
- Therefore, now we can continue using the standard debugging tools our IDE provides.

Extra motivation

- Outsmarting the compiler in hot loops.
- Gaining a better understanding of compilers.

```
            if ( !managedapp )
000000013F4AE4B3  cmp         dword ptr [managedapp (013F4C66B8h)],0
000000013F4AE4BA  jne         __tmainCRTStartup+1B8h (013F4AE4C8h)
                exit(mainret);
000000013F4AE4BC  mov         ecx,dword ptr [mainret (013F4C6698h)]
                exit(mainret);
000000013F4AE4C2  call        qword ptr [__imp_exit (013F4CB4A8h)]

            if (has_cctor == 0)
000000013F4AE4C8  cmp         dword ptr [has_cctor (013F4C669Ch)],0
000000013F4AE4CF  jne         __tmainCRTStartup+1C7h (013F4AE4D7h)
                _cexit();
000000013F4AE4D1  call        qword ptr [__imp__cexit (013F4CB548h)]

        }
```

- Reverse engineering.
- Fastest understanding of code written by others.

Questions

- Any questions?

Alternatives

- All of them are based on the same tools:
  - Memory panels. Mostly focused on:
    - Stack
    - Heap area
    - .bss sections
  - Disassembly panel
  - Registers panel
  - Memory read/write/execute breakpoints

- Alternatives:
  - GDB
  - Xcode

Alternatives

- GDB

```
gdb$ run
-----------------------------------------------------------------------[regs]
  EAX: BFFFF5FC  EBX: B7FCAFFC  ECX: B7FCD19C  EDX: 00000001  o d I t S z a P c
  ESI: BFFFF5F4  EDI: BFFFF580  EBP: BFFFF568  ESP: BFFFF550  EIP: 080483AA
  CS: 0073  DS: 007B  ES: 007B  FS: 0000  GS: 0033  SS: 007B
[007B:BFFFF550]-------------------------------------------------------[stack]
BFFFF5A0 : 00 00 00 00  F8 0F 00 B8 - 01 00 00 00  D0 82 04 08 ................
BFFFF590 : 70 F5 FF BF  D2 4D EB B7 - 00 00 00 00  00 00 00 00 p....M..........
BFFFF580 : FC AF FC B7  00 00 00 00 - 80 F5 FF BF  C8 F5 FF BF ................
BFFFF570 : 01 00 00 00  F4 F5 FF BF - FC F5 FF BF  6C 5B FF B7 ............l[..
BFFFF560 : 00 00 00 00  E0 0C 00 B8 - C8 F5 FF BF  14 4E EB B7 .............N..
BFFFF550 : FC AF FC B7  FC AF FC B7 - 18 95 04 08  FC AF FC B7 ................
[007B:BFFFF550]-------------------------------------------------------[data]
BFFFF550 : FC AF FC B7  FC AF FC B7 - 18 95 04 08  FC AF FC B7 ................
BFFFF560 : 00 00 00 00  E0 0C 00 B8 - C8 F5 FF BF  14 4E EB B7 .............N..
BFFFF570 : 01 00 00 00  F4 F5 FF BF - FC F5 FF BF  6C 5B FF B7 ............l[..
BFFFF580 : FC AF FC B7  00 00 00 00 - 80 F5 FF BF  C8 F5 FF BF ................
BFFFF590 : 70 F5 FF BF  D2 4D EB B7 - 00 00 00 00  00 00 00 00 p....M..........
BFFFF5A0 : 00 00 00 00  F8 0F 00 B8 - 01 00 00 00  D0 82 04 08 ................
BFFFF5B0 : 00 00 00 00  A0 5A FF B7 - B0 66 FF B7  F8 0F 00 B8 .....Z...f......
BFFFF5C0 : 01 00 00 00  D0 82 04 08 - 00 00 00 00  F1 82 04 08 ................
[0073:080483AA]-------------------------------------------------------[code]
0x80483aa <main+6>:     and    esp,0xfffffff0
0x80483ad <main+9>:     mov    eax,0x0
0x80483b2 <main+14>:    add    eax,0xf
0x80483b5 <main+17>:    add    eax,0xf
0x80483b8 <main+20>:    shr    eax,0x4
0x80483bb <main+23>:    shl    eax,0x4
0x80483be <main+26>:    sub    esp,eax
0x80483c0 <main+28>:    mov    eax,ds:0x80484f4
0x80483c5 <main+33>:    mov    DWORD PTR [ebp-24],eax
0x80483c8 <main+36>:    mov    al,ds:0x80484f8
0x80483cd <main+41>:    mov    BYTE PTR [ebp-20],al
0x80483d0 <main+44>:    sub    esp,0xc
0x80483d3 <main+47>:    push   0x80484f9
0x80483d8 <main+52>:    call   0x80482b8 <printf@plt>
0x80483dd <main+57>:    add    esp,0x10
0x80483e0 <main+60>:    leave
---------------------------------------------------------------------------
```

Alternatives

- Xcode