.NET Core
Memory Dump Analysis
Accelerated

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

Basic .NET Core programming and debugging

WinDbg Commands

We use these boxes to introduce some WinDbg commands used in practice exercises
Training Goals

- Review fundamentals
- Learn how to analyze process dumps
- Learn necessary commands in context
- Cover CoreCLR x64
Training Principles

- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content
Part 1: Fundamentals
Memory Space (x86)
Memory Space (x64)

Kernel Space

User Space

© 2022 Software Diagnostics Services
User / Managed Space

WinDbg Commands

lmv command lists all loaded modules (EXE and DLLs)
Types/Assemblies/Modules

WinDbg Commands

- `lmv` command lists all loaded modules (EXE and DLLs)
- `!IP2MD` command shows type method and module address
- `!DumpModule` command shows module name
WinDbg Commands

~<n>s command switches between threads

k command shows unmanaged stack trace

!Threads command shows managed threads

!CLRStack command shows managed stack trace

Managed Assembly Code

TID 106c

User Space (PID 23c0)

TID 1edc

LINQPad6

JIT code

coreclr

ntdll

kernel32

Compiles coreclr

TID 106c

00000000`00000000 00007fff`ffffffff

00000000`00000000 00007fff`ffffffff
Thread Stack Raw Data

WinDbg Commands

Get stack range:
!teb

Dump raw data:
dc / dps / dpp / dpa / dpu

Dump managed references:
!DumpStackObjects

User Space (PID e364)

JIT code

coreclr

kernel32

ntdll

User Stack for TID 102

Managed Heap

LINQPad6
Thread Stack Trace

User Stack for TID 102

Return address Module!FunctionA+110

Return address Module!FunctionB+220

Return address Module!FunctionC+130

FunctionA()
{
    ...
    FunctionB();
    ...
}
FunctionB()
{
    ...
    FunctionC();
    ...
}
FunctionC()
{
    ...
    FunctionD();
    ...
}

WinDbg Commands

0:000> k
Module!FunctionD
Module!FunctionC+130
Module!FunctionB+220
Module!FunctionA+110
Thread Stack Trace (no PDB)

User Stack for TID 102

FunctionA()
{
    ...
    FunctionB();
    ...
}
FunctionB()
{
    ...
    FunctionC();
    ...
}
FunctionC()
{
    ...
    FunctionD();
    ...
}

Symbol file Module.pdb

FunctionA 22000 - 23000
FunctionB 32000 - 33000
FunctionC 43000 - 44000
FunctionD 54000 - 55000

No symbols for Module

WinDbg Commands

0:000> k
Module+0
Module+43130
Module+32220
Module+22110
Thread Stack Trace (JIT Code)
### Example: Unmanaged Stack Trace

<table>
<thead>
<tr>
<th>#</th>
<th>Child-SP</th>
<th>RetAddr</th>
<th>Call Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00000029 6e3add78 00007ffa 7eb90458</td>
<td>0x00007ffa 7eb90458</td>
<td>win32u!NtUserWaitMessage+0x14</td>
</tr>
<tr>
<td>01</td>
<td>00000029 6e3ade40 00007ffa d724d61</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x581</td>
</tr>
<tr>
<td>02</td>
<td>00000029 6e3ade40 00007ffa d7247026</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x416</td>
</tr>
<tr>
<td>03</td>
<td>00000029 6e3ade40 00007ffa d6e6f21b</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x46</td>
</tr>
<tr>
<td>04</td>
<td>00000029 6e3ade40 00007ffa 7cb021e4</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x3b</td>
</tr>
<tr>
<td>05</td>
<td>00000029 6e3ade40 00007ffa d7247026</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x416</td>
</tr>
<tr>
<td>06</td>
<td>00000029 6e3ae0a0 00007ffa 7ca2534</td>
<td>0x00007ffa 7cb021e4</td>
<td>System.Windows.Forms!...+0x581</td>
</tr>
<tr>
<td>07</td>
<td>00000029 6e3ae0a0 00007ffa 7e97015c2</td>
<td>0x00007ffa 7cb021e4</td>
<td>System.Windows.Forms!...+0x416</td>
</tr>
<tr>
<td>08</td>
<td>00000029 6e3ae0a0 00007ffa d6e6f21b</td>
<td>0x00007ffa 7cb021e4</td>
<td>System.Windows.Forms!...+0x46</td>
</tr>
<tr>
<td>09</td>
<td>00000029 6e3ae0a0 00007ffa 7cb021e4</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x3b</td>
</tr>
<tr>
<td>0a</td>
<td>00000029 6e3ae0a0 00007ffa d6e6f21b</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x46</td>
</tr>
<tr>
<td>0b</td>
<td>00000029 6e3ae0a0 00007ffa 7cb021e4</td>
<td>0x00007ffa 7eb90458</td>
<td>System.Windows.Forms!...+0x3b</td>
</tr>
</tbody>
</table>

#### Method Information:

- **Method Name:** LINQPad.UIProgram.Go(System.String[])
- **Class:** 00007ffa 7c803a28
- **MethodTable:** 00007ffa 7c7f5aa0
- **mdToken:** 00000000 060001A4
- **Module:** 00007ffa 7c7f5aa0
- **IsJitted:** yes
- **Current CodeAddr:** 00007ffa 7ca2534
- **Version History:**
  - **ILCodeVersion:** 00000000 00000000
  - **ReJIT ID:** 0
  - **IL Addr:** 00000000 00000000
  - **CodeAddr:** 00007ffa 7ca2534
- **(MinOptJitted)**
  - **NativeCodeVersion:** 00000000 00000000

#### Module Information:

- **Name:** C:\Program Files\LINQPad6\LINQPad.GUI.dll
- **Attributes:** PEFile
- **SupportsUpdateableMethods:**
- **Assembly:** 00000177c5cdece0
- **BaseAddress:** 00000177DFDE0000
- **PEFile:** 00000177c5cdece0
- **ModuleId:** 00007ffa 7c7f5aa0
- **ModuleIndex:** 00000000 00000000
- **LoaderHeap:** 00000000 00000000
- **TypeDefToMethodTableMap:** 00007ffa 7c7f5aa0
- **TypeRefToMethodTableMap:** 00007ffa 7c7f5aa0
- **FieldDefToDescMap:** 00007ffa 7c7f5aa0
- **MemberRefToDescMap:** 00000000 00000000
- **TypeReferencesMap:** 00007ffa 7c7f5aa0
- **BaseAddress:** 00000177DFDE0000
- **IL Addr:** 00000000 00000000
- **CodeAddr:** 00007ffa 7ca2534
- **Version:**
  - **ILCodeVersion:** 00000000 00000000
  - **ReJIT ID:** 0
- **NativeCodeVersion:** 00000000 00000000

© 2022 Software Diagnostics Services
Pattern-Oriented Diagnostic Analysis

**Diagnostic Pattern:** a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

**Diagnostic Problem:** a set of indicators (symptoms, signs) describing a problem.

**Diagnostic Analysis Pattern:** a common recurrent analysis technique and method of diagnostic pattern identification in a specific context.

**Diagnostics Pattern Language:** common names of diagnostic and diagnostic analysis patterns. The same language for any operating system: Windows, Mac OS X, Linux, ...

**Checklist:** [http://www.dumpanalysis.org/windows-memory-analysis-checklist](http://www.dumpanalysis.org/windows-memory-analysis-checklist)
Part 2: Practice Exercises
Links

- Memory Dumps:
  Included in Exercise 0

- Exercise Transcripts:
  Included in this book
Exercise 0

- **Goal:** Install Debugging Tools for Windows or WinDbg Preview or pull Docker WinDbg image, and check that symbols are set up correctly

- **Patterns:** Stack Trace; Incorrect Stack Trace; Truncated Stack Trace

- **Commands:** k

- \`\`\`ANETCMDA-Dumps\Exercise-0-Download-Setup-WinDbg.pdf\`\`
Process Memory Dumps

Practice Exercises PNC1-PNC8
Modeling with LINQPad 6/7

```csharp
void Main()
{
    new ClassMain().Main();
}

// Define other methods and classes here

public class ClassMain
{
    public bool time2stop = false;

    public void Main()
    {
        while (!time2stop)
        {
            DoWork();
        }

    
    volatile int inSensor, outSensor;

    void DoWork()
    {
        outSensor ^= inSensor;
    }

    
    Ready
```

http://www.linqpad.net/
Exercise PNC1

- **Goal:** Learn how to use the SOS WinDbg extension to analyze managed space for the presence of exceptions

- **Patterns:** Stack Trace Collection (Unmanaged Space); CLR Thread; Software Exception; Exception Stack Trace; Managed Code Exception; Managed Stack Trace; Invalid Pointer; NULL Pointer (Data)

- **Commands:** .logopen, version, !peb, ~*k, ~*kL, .load, !pe, ~*e, Imv, .chain, .unload, !analyze -v, !CLRStack, .logclose

- \ANETCMDA-Dumps\Exercise-PNC1-Analysis-process-dump-ApplicationA.pdf
Exercise PNC2

- **Goal:** Compare the 64-bit process memory dump from exercise PNC1 with a 32-bit process memory dump

- **Patterns:** Platform-Specific Debugger

- \`\`\ANETCMDA-Dumps\Exercise-PNC2-Analysis-process-dump-ApplicationA-32.pdf\`\`
Exercise PNC3

- **Goal:** Learn how to find problem assemblies, modules, classes and methods, disassemble code, analyze CPU spikes

- **Patterns:** Active Thread; Manual Dump (Process); Technology-Specific Subtrace (JIT .NET Code); Spiking Thread; JIT Code; Annotated Disassembly (JIT .NET Code)

- **Commands:** `!analyze -v -hang, !IP2MD, !runaway, ~<>k, !U, !DumpMD, !DumpClass, !DumpMT, !DumpModule, !DumpAssembly, !DumpDomain, !DumpIL`

- \`\`\`ANETCMDA-Dumps\Exercis-PNC3-Analysis-process-dump-LINQPadB.pdf`\`\`
Exercise PNC4

- **Goal:** Learn how to recognize and analyze deadlocks using SOS, execution residue, handled exceptions, dump object references.

- **Patterns:** Special Thread (.NET CLR); Wait Chain (CLR Monitors); Deadlock (Managed Space); Execution Residue (User and Managed Spaces); Value References; Hidden Exception (User and Managed Spaces); Handled Exception (.NET CLR); Coincidental Symbolic Information; Rough Stack Trace (Unmanaged Space); Caller-n-Callee

- **Commands:** ~<>s, !Threads, !syncblk, !DumpObj, ub, dps, !DumpStack, !teb, dpS, !DumpStackObjects

- ANETCMDA-Dumps\Exercise-PNC4-Analysis-process-dump-LINQPadC.pdf
Deadlock

Thread #13
(owns)
Thread #14
(waiting)

Object
0000014a4fa39b00

Thread #13
(waiting)
Thread #14
(owns)

Object
0000014a4fa39b40

Thread #14

Exercise PNC5

- **Goal:** Learn how to analyze multiple managed exceptions

- **Patterns:** Stack Trace Collection (Managed Space); Multiple Exceptions (Managed Space)

- **Commands:** .sympath+, !help

- `\ANETCMDA-Dumps\Exercise-PNC5-Analysis-process-dump-ApplicationD.pdf`
Exercise PNC6

- **Goal:** Learn how to diagnose heap and handle leaks

- **Patterns:** Handle Leak; Object Distribution Anomaly (.NET Heap); Memory Leak (.NET Heap)

- **Commands:** !heap, !address, !DumpHeap, ?, !eeheap, !GCHandles, !FinalizeQueue, !handle

- \ANETCMDA-Dumps\Exercise-PNC6-Analysis-process-dump-LINQPadD.pdf
Exercise PNC7

- **Goal:** Learn how to recognize and analyze heap corruption

- **Patterns:** Regular Data; Dynamic Memory Corruption (Managed Heap)

- **Commands:** .formats, !VerifyHeap

- \ANETCMDA-Dumps\Exercise-PNC7-Analysis-process-dump-LINQPadE.pdf
Exercise PNC8

- **Goal:** Learn how to navigate virtual memory, search stack traces and memory for data and objects

- **Patterns:** Stack Overflow (Managed Space); Stack Trace Set

- **Commands:** .kframes, !VMMAP, !address, !ListNearObj, !ObjSize, !uniqstack, !findstack, !DumpArray, dpa, dpu, s

- \ANETCMDA-Dumps\Exercise-PNC8-Analysis-process-dump-LINQPadF.pdf
Pattern Links

CLR Thread
Managed Code Exception
Stack Trace Collection
Memory Leak
JIT Code
Managed Stack Trace
Multiple Exceptions
Caller-n-Callee
Hidden Exception
Technology-Specific Subtrace
Stack Overflow
Dynamic Memory Corruption
Special Thread
Execution Residue
Handled Exception
Annotated Disassembly
Wait Chain
Deadlock
Object Distribution Anomaly
Incorrect Stack Trace
Execution Residue
NULL Pointer (Data)
Handle Leak
Software Exception
Exception Stack Trace
Spiking Thread
Hidden Exception
Platform-Specific Debugger
Regular Data
Coincidental Symbolic Information
Active Thread
Truncated Stack Trace
Value References
Manual Dump
Stack Trace Set
Rough Stack Trace
Stack Trace Collection

CLR-related and managed
(full list)

Unmanaged user space

© 2022 Software Diagnostics Services
SOS Checklist

- CLR module and SOS extension versions (lmv and .chain)
- Managed exceptions (~*e !pe -nested)
- Managed threads (!Threads -special)
- Managed stack traces (!CLRStack -all)
- Managed execution residue (~*e !DumpStackObjects)
- Managed heap (!VerifyHeap, !DumpHeap -stat and !eeheap -gc)
- GC handles (!GCHandle)
- Finalizer queue (!FinalizeQueue)
- Sync blocks (!syncblk)
Resources

- WinDbg Help / [WinDbg.org](https://winDbg.org) (quick links)
- [DumpAnalysis.org](https://dumpAnalysis.org) / [SoftwareDiagnostics.Institute](https://softwareDiagnostics.Institute) / [PatternDiagnostics.com](https://patternDiagnostics.com)
- [Debugging.TV](https://debugging.TV) / [YouTube.com/DebuggingTV](https://youtube.com/debuggingTV) / [YouTube.com/PatternDiagnostics](https://youtube.com/patternDiagnostics)
- WinDbg images [https://hub.docker.com/r/patterndiagnostics/windbg](https://hub.docker.com/r/patterndiagnostics/windbg)
- .NET Runtime [https://github.com/dotnet/runtime](https://github.com/dotnet/runtime)
- Pro .NET Memory Management: For Better Code, Performance, and Scalability
- *Encyclopedia of Crash Dump Analysis Patterns, Third Edition*
- *Memory Dump Analysis Anthology (Diagnomicon)*
Q&A

Please send your feedback using the contact form on PatternDiagnostics.com
Thank you for attendance!