Windows Debugging\(^3\) Accelerated
Prerequisites

- Debugging in Visual Studio

or

- Basic crash dump analysis
Why WinDbg?

- Production debugging
- Redistributable
- Kernel mode debugging
Training Goals

- Review fundamentals
- Learn live debugging techniques
- See how software diagnostics is used during debugging
Training Principles

 Talk only about what I can show
 Lots of pictures
 Lots of examples
 Original content and examples
Course Idea

Debugging TV

- [www.debugging.tv](http://www.debugging.tv) (almost 50 episodes)
- PDB Symbols: episodes 0x01 – 0x04
- Kernel debugging setup: episode 0x25
Schedule Summary

Day 1
- Debugging Fundamentals (30 minutes)
- User Mode Debugging (2 hour and 30 minutes)

Day 2
- User Mode Debugging (30 minutes)
- Kernel Mode Debugging (1 hour and 30 minutes)
- Managed Debugging (1 hour)
Part 1: Fundamentals
Memory Space\(^3\)
Execution Mode³
Codes$^3$
Debugging Techniques³

- Breakpoints
- Tracing
- Inspection
Patterns³

- Debugging
- Elementary
- Memory Analysis
Pattern Mapping

- Software Incident
- Elementary Diagnostics
- Software Diagnostics
- Memory Analysis
- Debugging
- Debugging Implementation
Elementary Diagnostics

- Functional
  - Use-case Deviation

- Non-functional
  - Crash
  - Hang (includes delays)
  - Counter Value (includes resource leaks, CPU spikes)
  - Error Message
Analysis Patterns

- Memory (dump) analysis catalogue
- Software trace analysis catalogue
Pattern-Driven Analysis

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

**Checklist**: [http://www.dumpanalysis.org/windows-memory-analysis-checklist](http://www.dumpanalysis.org/windows-memory-analysis-checklist)
Unified Debugging Patterns

- Analysis (software diagnostics)
- Architecture/Design of debugging
- Implementation of debugging
- Usage/presentation of debugging (for example, Watch dialog)
Space Review (x86)

User Space

Kernel Space

0:000> lm
start end module name
013e0000 013f5000 App
10000000 10039000 WinCRT
70120000 70134000 MyDLL
72a40000 72a91000 winspool
73b00000 73930000 uxtheme
73ec0000 73ed3000 dwmapi
746b0000 746bc000 CRYPTBASE
746c0000 74720000 sspicli
74a00000 74a83000 clbcatq
74a90000 74b20000 gd132
74b20000 74baf000 oleaut32
74b20000 74baf000 gdi32
74bc0000 74bba000 lpk
74bc0000 74c6c000 msvcr71
74ca0000 74d6c000 msctf
750e0000 7517b000 advapi32
751e0000 752f0000 kernel32
752f0000 75300000 sechost
75390000 753d7000 KERNELBASE
753e0000 75437000 shlwapi
756d0000 756cd000 ole32
756d0000 756cd000 usp10
764d0000 765c0000 rpcrt4
765c0000 766c0000 user32
766c0000 76720000 imm32
76fe0000 77160000 ntdll
Space Review (x64)

User Space

- ntdll
- App
- MyDLL

Kernel Space

0:000> lm

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000`76be0000</td>
<td>00000000`76cff000</td>
<td>kernel32</td>
</tr>
<tr>
<td>00000000`76d00000</td>
<td>00000000`76dfa000</td>
<td>user32</td>
</tr>
<tr>
<td>00000000`76e00000</td>
<td>00000000`76fa9000</td>
<td>ntdll</td>
</tr>
<tr>
<td>000007fe`f9d20000</td>
<td>000007fe`f9d37000</td>
<td>App</td>
</tr>
<tr>
<td>000007fe`fb700000</td>
<td>000007fe`fb718000</td>
<td>MyDLL</td>
</tr>
<tr>
<td>000007fe`fb900000</td>
<td>000007fe`fb9e6000</td>
<td>dwmapi</td>
</tr>
<tr>
<td>000007fe`fd100000</td>
<td>000007fe`fd19f000</td>
<td>uitheme</td>
</tr>
<tr>
<td>000007fe`fd4e0000</td>
<td>000007fe`fd54c000</td>
<td>KERNELBASE</td>
</tr>
<tr>
<td>000007fe`fd600000</td>
<td>000007fe`fd749000</td>
<td>clbcatq</td>
</tr>
<tr>
<td>000007fe`fd940000</td>
<td>000007fe`fd9a1b000</td>
<td>advapi32</td>
</tr>
<tr>
<td>000007fe`fd9d0000</td>
<td>000007fe`fd949000</td>
<td>rpcrt4</td>
</tr>
<tr>
<td>000007fe`fda20000</td>
<td>000007fe`fda1b000</td>
<td>gdi32</td>
</tr>
<tr>
<td>000007fe`fda50000</td>
<td>000007fe`fda6e000</td>
<td>ole32</td>
</tr>
<tr>
<td>000007fe`fda90000</td>
<td>000007fe`fda8c000</td>
<td>imm32</td>
</tr>
<tr>
<td>000007fe`fdae0000</td>
<td>000007fe`fdae0000</td>
<td>lpk</td>
</tr>
<tr>
<td>000007fe`fed10000</td>
<td>000007fe`fed20000</td>
<td>msctf</td>
</tr>
<tr>
<td>000007fe`fed20000</td>
<td>000007fe`fee29000</td>
<td>sechost</td>
</tr>
<tr>
<td>000007fe`fee50000</td>
<td>000007fe`fee50000</td>
<td>usp10</td>
</tr>
<tr>
<td>000007fe`fee80000</td>
<td>000007fe`fee80000</td>
<td>msvcr32</td>
</tr>
<tr>
<td>000007fe`f8030000</td>
<td>000007fe`f8070000</td>
<td>oleaut32</td>
</tr>
</tbody>
</table>
Thread Stack Trace

```
FunctionA()
{
    ...  
    FunctionB();
    ...
}
FunctionB()
{
    ...  
    FunctionC();
    ...
}
FunctionC()
{
    ...  
    FunctionD();
    ...
}
```
Thread Stack Trace (no PDB)

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

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

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

Module+43130

Symbol file Module.pdb

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

No symbols for Module

© 2013 Software Diagnostics Services
void main()
{
    foo();
    crash();
}

void foo()
{
    char sz[256] = "Some String";
    bar();
}

void bar()
{
    do();
}

void crash()
{
    WER();
}
First vs. Second Chance

- **First chance exceptions**
  WinDbg is notified of an exception, you can ignore it

- **Second chance exceptions**
  If the exception wasn’t handled (for example, by a catch block) WinDbg is notified again

- Relation to crash dumps
Part 2: Practice Exercises
Links

- Applications:
  Not available in preview version

- Exercise Transcripts:
  Not available in preview version
Warning

Because this is live debugging due to differences in actual systems and also ASLR (Address Space Layout Randomization) when you launch applications actual addresses in WinDbg command output may be different from shown in exercise transcripts.
Exercise 0

- **Goal:** Install Debugging Tools for Windows and learn how to set up symbols correctly

- **Memory Analysis Patterns:** Incorrect Stack Trace
User Mode Debugging

Exercises D1-D8
Exercise D1

- **Goal:** Learn how code generation parameters can influence process execution behavior

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Exception Stack Trace

- **Debugging Implementation Patterns:** Scope, Variable Value, Type Structure, Code Breakpoint
Exercise D2

- **Goal:** Learn how to use hardware breakpoints to catch data corruption

- **Elementary Diagnostics Patterns:** Counter Value

- **Memory Analysis Patterns:** Unloaded Module, Memory Leak (process heap), Abnormal Value *(from trace analysis patterns)*

- **Debugging Implementation Patterns:** Break-in, Code Breakpoint, Scope, Variable Value, Data Breakpoint
Exercise D3

- **Goal:** Learn how to navigate parameters, static and local variables, and data structures

- **Elementary Diagnostics Patterns:** Crash

- **Memory Analysis Patterns:** Exception Stack Trace, Stack Overflow, String Parameter, Module Variable

- **Debugging Implementation Patterns:** Break-in, Scope, Variable Value, Type Structure
Exercise D4

- **Goal:** Learn how to use conditional breakpoints to log behavior

- **Elementary Diagnostics Patterns:** Use-case Deviation

- **Memory Analysis Patterns:** -

- **Debugging Implementation Patterns:** Break-in, Code Breakpoint, Breakpoint Action
Exercise D5

- **Goal:** Learn how to debug multiple processes and their deadlock

- **Elementary Diagnostics Patterns:** Crash, Hang

- **Memory Analysis Patterns:** Exception Stack Trace, NULL Data Pointer, Execution Residue, Hidden Exception, Handled Exception, Deadlock

- **Debugging Implementation Patterns:** Break-in
Expected Behavior

Thread A
- Acquires Mutex A
- Waits for Mutex B
- Acquires Mutex B
- Releases Mutex B
- Releases Mutex A
- Acquires Mutex A

Thread B
- Acquires Mutex B
- Releases Mutex B
- Waits for Mutex A
- Releases Mutex A
- Acquires Mutex A
- Releases Mutex A
Deadlock

Thread A
- Acquires Mutex A
- Waits for Mutex B

Thread B
- Acquires Mutex B
- NewFeature()
- Waits for Mutex A
Exercise D6

- **Goal:** Learn how to recognize when we need kernel level debugging

- **Elementary Diagnostics Patterns:** Hang, Counter Value

- **Memory Analysis Patterns:** Abnormal Value (*from trace analysis patterns*), Spiking Thread

- **Debugging Implementation Patterns:** Break-in, Code Breakpoint, Data Breakpoint, Code Trace
Exercise D7

○ **Goal:** Learn how to manipulate threads to debug race conditions

○ **Elementary Diagnostics Patterns:** Crash

○ **Memory Analysis Patterns:** Exception Stack Trace, NULL Code Pointer

○ **Debugging Implementation Patterns:** Frozen Thread
Exercise D8

- **Goal:** Learn how to inspect heap for signs of corruption
- **Elementary Diagnostics Patterns:** Crash
- **Memory Analysis Patterns:** Dynamic Memory Corruption (process heap), Module Variable, Exception Stack Trace
- **Debugging Implementation Patterns:** Break-in
Kernel Mode Debugging

Exercises KD6, KD9, KD10
Exercise K0

- **Goal:** Setup VMware kernel debugging environment
Space Review (x64)

User Space
- ntdll
- App
- MyDLL
- hal
- nt
- driver

Kernel Space
- win32k
- TSDDD
- cdd

0: kd> lmk

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ffffff802`b309f000</td>
<td>ffffff802`b30a8000</td>
<td>kd</td>
</tr>
<tr>
<td>ffffff802`b3a1d000</td>
<td>ffffff802`b3a89000</td>
<td>hal</td>
</tr>
<tr>
<td>ffffff802`b3a89000</td>
<td>ffffff802`b41d2000</td>
<td>nt</td>
</tr>
<tr>
<td>[...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ffffff880`01a2f000</td>
<td>ffffff880`01a4b000</td>
<td>disk</td>
</tr>
<tr>
<td>ffffff880`01a53000</td>
<td>ffffff880`01c36000</td>
<td>Ntfs</td>
</tr>
<tr>
<td>ffffff880`01c36000</td>
<td>ffffff880`01c51000</td>
<td>ksecdd</td>
</tr>
<tr>
<td>[...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ffffff960`0007a000</td>
<td>ffffff960`0046f000</td>
<td>win32k</td>
</tr>
<tr>
<td>ffffff960`006d1000</td>
<td>ffffff960`006da000</td>
<td>TSDDD</td>
</tr>
<tr>
<td>ffffff960`008a4000</td>
<td>ffffff960`008da000</td>
<td>cdd</td>
</tr>
</tbody>
</table>
Space Review

User Space

Kernel Space

Complete stack traces (x64 + x86)
Context Switch

We always see the current process space

User Space

current process A
(Notepad)

Kernel Space

User Space

current process B
(AppD.exe)

Kernel Space

Context switch
Common Commands

- `.logopen <file>`
  Opens a log file to save all subsequent output

- View commands
  Dump everything or selected processes and threads (context changes automatically)

- Switch commands
  Switch to a specific process or thread
View Commands

- **!process 0 0**
  Lists all processes

- **!process <address> 3f**
  Lists process information including CPU times, environment, modules and its thread stack traces

- **!thread <address> 1f**
  Shows thread information and stack trace
Switch Commands

- `.process /r /p <address>`
  Switches to a specified process. Its context becomes current. Reloads symbol files for user space. Now we can use commands like !cs

  ```
  0: kd> .process /r /p fffffa80044d8b30
  Implicit process is now fffffa80`044d8b30
  Loading User Symbols
  ........................................
  ```

- `.thread <address>`
  Switches to a specified thread. Assumes the current process context.
  Now we can use commands like k*

- `.thread /r /p <address>`
  The same as the previous command but makes the thread process context current and reloads symbol files for user space:

  ```
  0: kd> .thread /r /p fffffa80051b7060
  Implicit thread is now fffffa80`051b7060
  Implicit process is now fffffa80`044d8b30
  Loading User Symbols
  ........................................
  ```
Exercise KD6

- **Goal:** Learn how to use kernel level debugging to catch corruption caused by a driver or other process.

- **Elementary Diagnostics Patterns:** Abnormal Value *(from trace analysis patterns)*, Spiking Thread.

- **Memory Analysis Patterns:** Abnormal Value *(from trace analysis patterns)*, Spiking Thread.

- **Debugging Implementation Patterns:** Code Breakpoint, Data Breakpoint, Code Trace.
Exercise KD9

- **Goal:** Learn how to debug a 32-bit process under x64 Windows

- **Elementary Diagnostics Patterns:** Hang

- **Memory Analysis Patterns:** Abnormal Value (*from trace analysis patterns*), Spiking Thread

- **Debugging Implementation Patterns:** Break-in
Exercise KD10

- **Goal:** Learn how to debug handle leaks in user and kernel mode debugging

- **Elementary Diagnostics Patterns:** Counter Value

- **Memory Analysis Patterns:** Handle Leak, Historical Information

- **Debugging Implementation Patterns:** Break-in, Usage Trace
Managed Debugging

Exercise MD11
Modeling with LINQPad

```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;
    }
```

http://www.linqpad.net/

Download x64 Build
User / Managed Space

User Space (PID d98)

LINQPad

user32

ntdll

CLR
Process Threads

.user32

.ntdll

User Space (PID d98)

 LINQPad

00000000 00000000

000007ff ffffffff

JIT code

CLR

.load <a path to SOS>
Loads sos WinDbg extension

~<n>s command switches between threads

k command shows unmanaged stack trace

!Threads command shows managed threads

!CLRStack command shows managed stack trace
Stack Trace Example

0:000> k
Child-SP  RetAddr           Call Site
00000000`007ce4c8  000007fe`e4cc2014 USER32!NtUserWaitMessage+0xa
00000000`007ce4d0  000007fe`e4cc158c System_Windows_Forms_ni+0x312014
00000000`007ce6d0  000007fe`e4cc0f3f System_Windows_Forms_ni+0x31158c
00000000`007ce820  000007fe`89f9572d System_Windows_Forms_ni+0x310f3f
00000000`007ce8b0  000007fe`89f91c2c  0x000007fe`89f9572d
00000000`007ceeb40  000007fe`89f9086c  0x000007fe`89f91c2c
00000000`007cf000  000007fe`89f90439  0x000007fe`89f9086c
00000000`007cf070  000007fe`89f9012d  0x000007fe`89f90439
00000000`007cf0a0  000007fe`e9607d33  0x000007fe`89f9012d
00000000`007cf0e0  000007fe`e96056e6 clr!CallDescrWorkerInternal+0x83
00000000`007cf120  000007fe`e96057af clr!CallDescrWorkerWithHandler+0x4a
00000000`007cf160  000007fe`e96c9120 clr!MethodDescCallSite::CallTargetWorker+0x2e6
00000000`007cf310  000007fe`e96c986e clr!RunMain+0x1e7
00000000`007cf4e0  000007fe`e96c9786 clr!Assembly::ExecuteMainMethod+0xb6
00000000`007cf7d0  000007fe`e96c92e2 clr!SystemDomain::ExecuteMainMethod+0x45e
00000000`007cf90  000007fe`e96c923a clr!ExecuteEXE+0x3f
00000000`007cfe00  000007fe`e96c5a90 clr!CorExeMainInternal+0xae
00000000`007cfe90  000007fe`fada74e5 clr!CorExeMain+0x14
00000000`007fed0  000007fe`f945b21 mscoreei!CorExeMain+0xe0
00000000`007cff20  00000000`77a2652d MSCOREE!CorExeMain_Exported+0x57
00000000`007cff50  00000000`77b5c521 KERNEL32!BaseThreadInitThunk+0xd
00000000`007cff80  00000000`00000000 ntdll!RtlUserThreadStart+0x1d

© 2013 Software Diagnostics Services
Examining JIT Code

```
0:000> !IP2MD 0x000007fe`89f9572d
MethodDesc: 000007fe89fd0288
Method Name: LINQPad.Program.Run(System.String, Boolean, System.String, Boolean, Boolean, Boolean, System.String)
Class: 000007fe89f81a60
MethodTable: 000007fe89fd04d0
mdToken: 00000000006000a82
Module: 000007fe89e42f90
IsJitted: yes
CodeAddr: 000007fe89f94c50
Transparency: Critical

0:000> !IP2MD System_Windows_Forms_ni+0x310f3f
MethodDesc: 000007fee49ebad8
Class: 000007fee49d08e0
MethodTable: 000007fee4da2670
mdToken: 00000000006006291
Module: 000007fee49b1000
IsJitted: yes
CodeAddr: 000007fee4cc0ed0
Transparency: Safe critical
```
Exercise MD11

**Goal:** Learn how to find problem modules, classes and methods, disassemble code, dump object references, recognize and analyze deadlocks using SOS(EX) extensions

**Elementary Diagnostics Patterns:** Crash, Hang

**Memory Analysis Patterns:** Exception Stack Trace, Handled Exception, NULL Data Pointer, Nested Exception, CLR Thread, JIT Code, Deadlock, Execution Residue

**Debugging Implementation Patterns:** Break-in
Deadlock

Thread #18 (owns)
SyncBlock: 000000001ed05018
Object: 0000000004747ed0
Thread #18 (waiting)

Thread #23 (owns)
SyncBlock: 000000001ed04bb8
Object: 0000000004747f10
Thread #23 (waiting)

Thread #23

Thread #23
Postmortem Debugging

- Memory dump collection
  - `.dump / .dumpcab` WinDbg commands
  - External methods

- Training
  - Accelerated Windows Memory Dump Analysis
  - Advanced Windows Memory Dump Analysis
  - Accelerated .NET Memory Dump Analysis
Analysis Pattern Links

Spiking Thread     CLR Thread
Heap Corruption    Exception Stack Trace
Execution Residue  Hidden Exception
Invalid Pointer    Managed Code Exception
Managed Stack Trace NULL Data Pointer
NULL Code Pointer  Stack Trace
Stack Trace Collection Stack Overflow
Deadlock (objects) Virtualized Process
Message Box        Memory Leak
False Function Parameters String Parameter
Handle Leak         Historical Information
Abnormal Value      Counter Value
Module Variable     Deadlock (managed)
Handled Exception   Nested Exception
JIT Code
Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org
- Debugging.TV / DebuggingTV YouTube Channel
- Windows Debugging: Practical Foundations
- x64 Windows Debugging: Practical Foundations
- Windows Debugging Notebook: Essential User Space WinDbg Commands
- Software Diagnostics Library
- Pattern-Driven Software Problem Solving
- Memory Dump Analysis Anthology
Q&A

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