



Defect
↑
Detect →



Windows Memory Dump Analysis Extended

Extensions
Database and Event Stream Processing
Data Science and Visualization
Machine Learning and AI

Second Edition

Dmitry Vostokov
Software Diagnostics Services

Extended Windows Memory Dump Analysis: Using and Writing WinDbg Extensions, Database and Event Stream Processing, Data Science and Visualization, Machine Learning and AI, Second Edition

Published by OpenTask, Republic of Ireland

Copyright © 2024 by OpenTask

Copyright © 2024 by Software Diagnostics Services

Copyright © 2024 by Dmitry Vostokov

Copyright © 2024 by Dublin School of Security

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, transmitted in any form or by any means, or used for training artificial intelligence systems without the prior written permission of the publisher.

Product and company names mentioned in this book may be trademarks of their owners.

OpenTask books and magazines 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: 978-1-912636-51-8 (Paperback)

Revision 2.00 (November 2024)

Contents

About the Author.....	5
Introduction.....	7
Practice Exercises	17
Exercise E0: Download, set up, and verify your WinDbg or Debugging Tools for Windows installation or Docker Debugging Tools for Windows image.....	21
Exercise ES1: Explore Patterns WinDbg Extension.....	35
Exercise ES2: Explore MEX WinDbg Extension	41
Exercise ES3: Explore DbgKit WinDbg Extension.....	63
Exercise ES4: Explore win32kext WinDbg Extension.....	105
Exercise ES5: Explore SwishDbgExt WinDbg Extension.....	112
Exercise ES6: Explore Occhext WinDbg Extension	132
Exercise ES7: Explore pykd WinDbg Extension.....	144
Exercise ES8: Explore snapshot WinDbg Extension.....	175
Exercise ES9: Explore WinDbg Copilot.....	189
Exercise ES10: Explore ChatDBG WinDbg Extension	206
Exercise EW1: Writing WinDbg Extension (WdbgExts C API).....	222
Exercise EW2: Writing WinDbg Extension (DbgEng COM API).....	237
Exercise EW3: Writing WinDbg Extension (ExtExtension C++ API)	251
Exercise EW4: Writing WinDbg Extension (Rust)	264
Exercise EP1: Install Kafka Environment.....	274
Exercise EP2: Connect WinDbg to Kafka	277
Exercise EP3: Configure Kafka Connect to Send WinDbg Output	281
Exercise ED1: Install MongoDB Environment.....	290
Exercise ED2: Connect WinDbg to MongoDB.....	297
Exercise EV1: Install Jupyter Notebook Environment	307
Exercise EV2: Execution Residue Visualization.....	316
Exercise EV3: Find Anomalies in Stack Traces	332
Exercise ML1: Compute Similarity of Stack Traces.....	340
Exercise ML2: Group Stack Traces into Clusters.....	344
Exercise ML3: Identify Anomalous Stack Traces	347
Conclusion	355

Exercise ES1: Explore Patterns WinDbg Extension

Goal: Explore the patterns⁴ WinDbg extension

1. Launch WinDbg.
2. Open `\EWMDA-Dumps\Complete\x64\MEMORY-W11.DMP`.
3. We get the dump file loaded:

```
Microsoft (R) Windows Debugger Version 10.0.27668.1000 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\EWMDA-Dumps\Complete\x64\MEMORY-W11.DMP]
Kernel Bitmap Dump File: Full address space is available

***** Path validation summary *****
Response                               Time (ms)      Location
Deferred                                0              srv*
Symbol search path is: srv*
Executable search path is:
Windows 10 Kernel Version 22000 MP (2 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS Personal
Edition build lab: 22000.1.amd64fre.co_release.210604-1628
Kernel base = 0xfffff806`61e00000 PsLoadedModuleList = 0xfffff806`62a296b0
Debug session time: Sat Nov 13 23:17:16.607 2021 (UTC + 1:00)
System Uptime: 0 days 0:03:06.813
Loading Kernel Symbols
.....
.....
.....
..
Loading User Symbols
.....
Loading unloaded module list
.....
For analysis of this file, run !analyze -v
nt!KeBugCheckEx:
fffff806`62215590 mov     qword ptr [rsp+8],rcx ss:0018:ffffbe82`96f64670=000000000000000a
```

4. We load the x64 version of the *patterns* extension:

```
0: kd> .load C:\EWMDA-Dumps\patterns\x64\patterns
```

⁴ The latest version can be downloaded from: <https://www.patterndiagnostics.com/patterns-extension>.

5. We can also check if it was loaded successfully:

```
0: kd> .chain
Extension DLL search Path:
[...]
Extension DLL chain:
C:\EWMMA-Dumps\patterns\x64\patterns: image 2.1.0.0, API 0.1.0, built Sun Sep  8 12:42:15 2024
[path: C:\EWMMA-Dumps\patterns\x64\patterns.dll]
DbgEngCoreDMExt: image 10.0.27668.1000, API 0.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\winext\DbgEngCoreDMExt.dll]
wdfkd: image 10.0.27668.1000, API 1.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\winext\wdfkd.dll]
MachOBinComposition: image 10.0.27668.1000, API 0.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\winext\MachOBinComposition.dll]
ELFBinComposition: image 10.0.27668.1000, API 0.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\winext\ELFBinComposition.dll]
dbghelp: image 10.0.27668.1000, API 10.0.6,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\dbghelp.dll]
exts: image 10.0.27668.1000, API 1.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\WINXP\exts.dll]
kext: image 10.0.27668.1000, API 1.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\winext\kext.dll]
kdexts: image 10.0.27668.1000, API 1.0.0,
[path: C:\Program Files\WindowsApps\Microsoft.WindowsDbg_1.2407.24003.0_x64__8wekyb3d8bbwe\amd64\winxp\kdexts.dll]
```

6. Since the extension is on top, we can use **!help** instead of **!patterns.help**:

```
0: kd> !help
Patterns Debugger Extension DLL (Version 2.1.0.0 [std]). Copyright © 2015-2024 Software Diagnostics
Services. All rights reserved.

Commands:
  lst                - Shows the current list of memory analysis pattern categories
  lst category      - Shows the current list of memory analysis patterns for the specified category
  sdl abbreviation - Opens a pattern description from Software Diagnostics Library
  chk               - Shows the current memory analysis checklist categories
  chk category     - Shows the current memory analysis checklist for the specified category
  eula              - Shows license terms
```

7. The extension supports DML hyperlinks when possible (for example, in the GUI debugger version such as WinDbg, so the console-based Docker environment doesn't have it). **!chk** command outputs Memory Analysis Checklist⁵ categories:

```
0: kd> !chk
The following memory analysis checklist categories are available:

General [G]
Application [A]
System [S]
BSOD [B]
.NET [.]
```

8. In the DML version, we can click on a category and also use the [Back](#) link, but for non-DML environments, we need to specify a category for the **!chk** command:

```
0: kd> !chk S
Memory Analysis Checklist for System Hang:

Default analysis (!analyze -v -hang)
ERESOURCE contention (!locks)
Processes and virtual memory including session space (!vm 4)
```

⁵ <https://www.dumpanalysis.org/windows-memory-analysis-checklist>

```

Important services are present and not hanging (for example, terminal or IMA services for Citrix environments)
Pools (!poolused)
Waiting threads (!stacks)
Critical system queues (!xqueue f)
I/O (!irpfind)
The list of all thread stack traces (!process 0 3f)
LPC/ALPC chain for suspected threads (!lpc message or !alpc /m after search for "Waiting for reply to LPC" or "Waiting for reply to ALPC" in !process 0 3f output)
Mutants (search for "Mutants - owning thread" in !process 0 3f output)
Critical sections for suspected processes (!cs -l -o -s)
Sessions, session processes (!session, !sprocess)
Processes (size, handle table size) (!process 0 0)
Running threads (!running)
Ready threads (!ready)
DPC queues (!dpcs)
The list of APCs (!apc)
Internal queued spinlocks (!qlocks)
Computer name (dS srv!srvcomputername)
File cache, VACB (!filecache)
File objects for blocked thread IRPs (!irp -> !fileobj)
Network (!ndiskd.miniports and !ndiskd.pktpools)
Disk (!scsikd.classext -> !scsikd.classext class_device 2)
Modules rdbss, mrxdav, mup, mrxsmb in stack traces
Functions Ntfs!Ntfs* and nt!Fs* in stack traces

Back

```

Note: The extension uses the currently published checklist at the time of this writing. It may be amended in the future, so please download the most recent version of the extension for your practical analysis needs.

9. The **!lst** command lists analysis pattern categories, and we can navigate back and forth in the DML version. In the non-DML version, we need to specify the category as a parameter:

```

0: kd> !lst
Memory Analysis Pattern Categories:

Hooksware Patterns [H]
Wait Chain Patterns [W]
DLL Link Patterns [L]
Memory Consumption Patterns [M]
Dynamic Memory Corruption Patterns [C]
Deadlock and Livelock Patterns [D]
Contention Patterns [N]
Stack Overflow Patterns [O]
.NET / CLR / Managed Space Patterns [.]
Stack Trace Patterns [S]
Symbol Patterns [Y]
Exception Patterns [E]
Meta-Memory Dump Patterns [-]
Module Patterns [!]
Optimization Patterns [I]
Thread Patterns [T]
Process Patterns [P]
Executive Resource Patterns [X]
Falsity and Coincidence Patterns [F]
RPC, LPC and ALPC Patterns [R]
Hidden Artifact Patterns [A]

```

Pointer Patterns [*]
Frame Patterns [+]
CPU Consumption Patterns [^]
Region Patterns [#]
Malware Analysis Patterns [@]

0: kd> !lst S

Stack Trace Patterns:

Stack Trace [STR]
Stack Trace Collection (unmanaged space) [STCU]
Special Stack Trace [SSTR]
Exception Stack Trace [ESTR]
Dual Stack Trace [DSTR]
Truncated Stack Trace [TSTR]
Managed Stack Trace [MSTR]
Incorrect Stack Trace [ISTR]
Stack Trace Set [STSE]
Stack Trace Collection (managed space) [STCM]
Stack Trace Collection (predicate) [STCP]
Empty Stack Trace [EMST]
Stack Trace Collection (I/O requests) [STCI]
Stack Trace Change [STCH]
First Fault Stack Trace [FFST]
Critical Stack Trace [CSTR]
RIP Stack Trace [RSTR]
Glued Stack Trace [GSTR]
Rough Stack Trace (unmanaged space) [RSTU]
Past Stack Trace [PSTR]
Stack Trace (I/O request) [STIO]
Stack Trace (file system filters) [STFS]
Stack Trace (database) [STDB]
Variable Subtrace [VASU]
Technology-Specific Subtrace (COM interface invocation) [TSCI]
Technology-Specific Subtrace (dynamic memory) [TSDM]
Technology-Specific Subtrace (JIT .NET code) [TSJN]
Technology-Specific Subtrace (COM client call) [TSCC]
Internal Stack Trace [INST]
Stack Trace Collection (CPUs) [STCC]
Stack Trace Surface [STSU]
Hidden Stack Trace [HSTR]
Constant Subtrace [COSU]
Stack Trace Signature [STSI]
Quotient Stack Trace [QSTR]
Module Stack Trace [MOST]
Coincidental Frames [COFR]
Least Common Frame [LCFR]
Foreign Module Frame [FMFR]
Unified Stack Trace [USTR]
Aggregated Frames [AGFR]
Stack Trace (I/O devices) [SIOD]
Stack Trace Motif [STMO]
Stack Trace Race [STRA]
Source Stack Trace [SRCS]
Hidden Stack [HIST]
Interrupt Stack [INTS]
Frame Trace [FRTR]
False Frame [FAFR]
Procedure Call Chain [PCCH]
Rough Stack Trace (managed space) [RSTM]

Rough Stack Trace Collection (unmanaged space) [RSCU]

Caller-n-Callee [CNCA]

Shadow Stack Trace [SHST]

Foreign Stack [FSTA]

Annotated Stack Trace [ANST]

Back

10. If we click on an individual analysis pattern in the DML-based version, the default web browser should open the appropriate link from the Software Diagnostics Library⁶ (which requires prior login). In the non-DML version, we need to use the `!sdl` command with the corresponding four-letter pattern tag:

```
0: kd> !sdl RSCU
```

Software Diagnostics Library

Structural and Behavioral Patterns for Software Diagnostics, Forensics and Prognostics

« Exception Stack Trace, Stored Exception, Translated Exception, Execution Residue, Hidden Exception, NULL Pointer, Exception Module, Stack Trace Motif, No Component Symbols, and Coincidental Symbolic Information: pattern cooperation

Crash Dump Analysis Patterns (Part 281)

We have parallels between various [Stack Trace](#) analysis patterns and corresponding [Stack Trace Collection](#) analysis patterns, for example, for unmanaged space. The same can be done between [Rough Stack Trace](#) and the new analysis pattern that we call **Rough Stack Trace Collection**, for example, for unmanaged space. In WinDbg, such a collection can be done using a similar `script` but with `dpS` command instead. In essence, it is a collection of symbolic [Execution Residue](#) from all thread stack regions. This analysis pattern may help in identification of [Ubiquitous Components](#) not visible on stack traces, and [Past Stack Traces](#), for example, corresponding to various leaks.

11. We can unload the extension anytime if we no longer need it:

```
0: kd> .unload C:\EWMDDA-Dumps\patterns\x64\patterns
```

Note: We recommend exiting WinDbg after each exercise to avoid possible confusion and glitches.

⁶ <https://www.dumpanalysis.org/blog>