Windows Memory Dump Analysis

Accelerated

Version 6

Part 1: Process User Space

Dmitry Vostokov
Software Diagnostics Services

Prerequisites

Basic Windows troubleshooting

WinDbg Commands

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

Training Goals

- Part 1A: Review fundamentals
- Part 1B: Review x64 disassembly
- Part 1C: Learn how to analyze process dumps
- Part 2A: Review fundamentals
- Part 2B: Review x64 disassembly
- Part 2C: Learn how to analyze kernel dumps
- Part 2D: Learn how to analyze complete (physical memory) dumps
- Part 2E: Learn how to analyze minidumps

Training Principles

- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content and examples

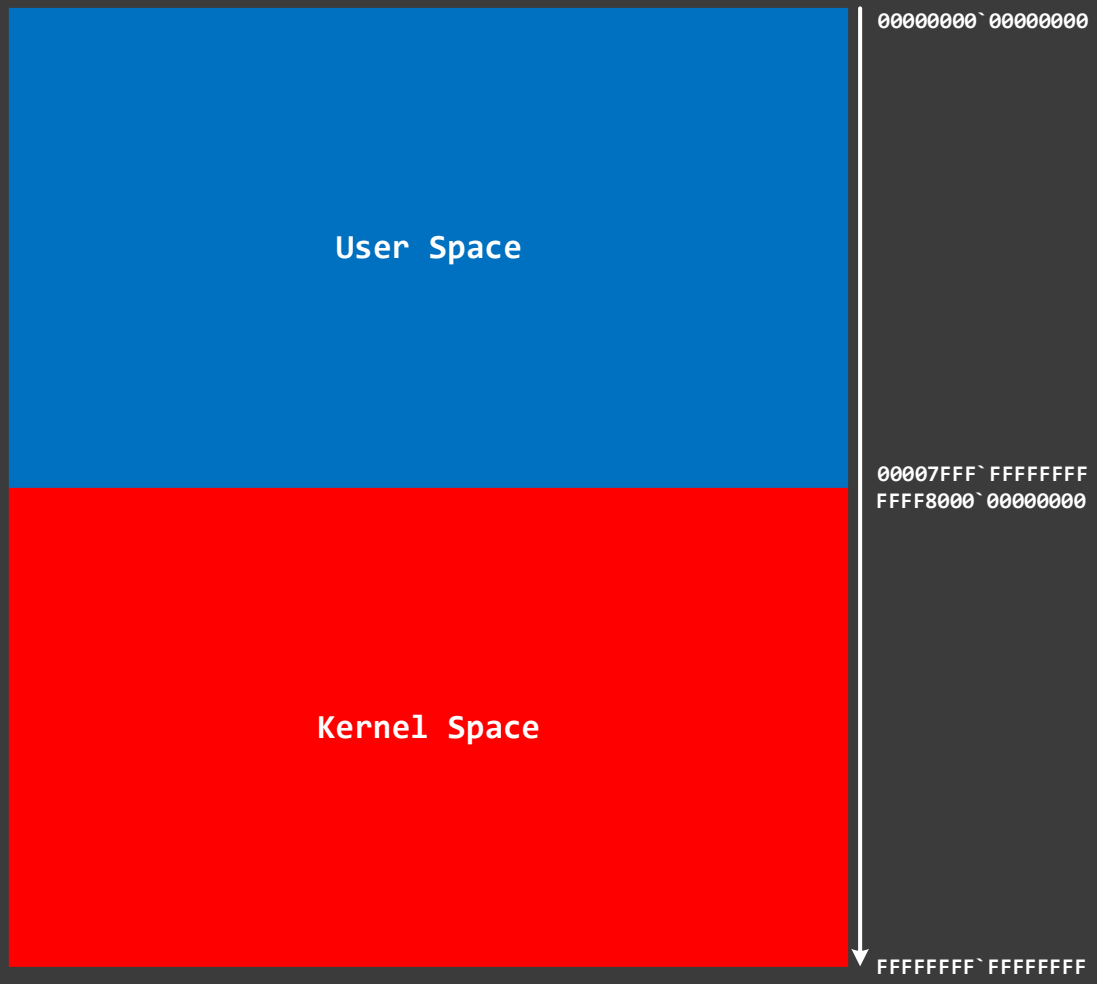
Coverage (Part 1)

- Windows 10 and 11
- Both x64* and x86 code, WOW64
- x64 disassembly review
- Preliminary .NET analysis
- Process memory dumps
- Crashes, hangs, memory and handle leaks, CPU spikes

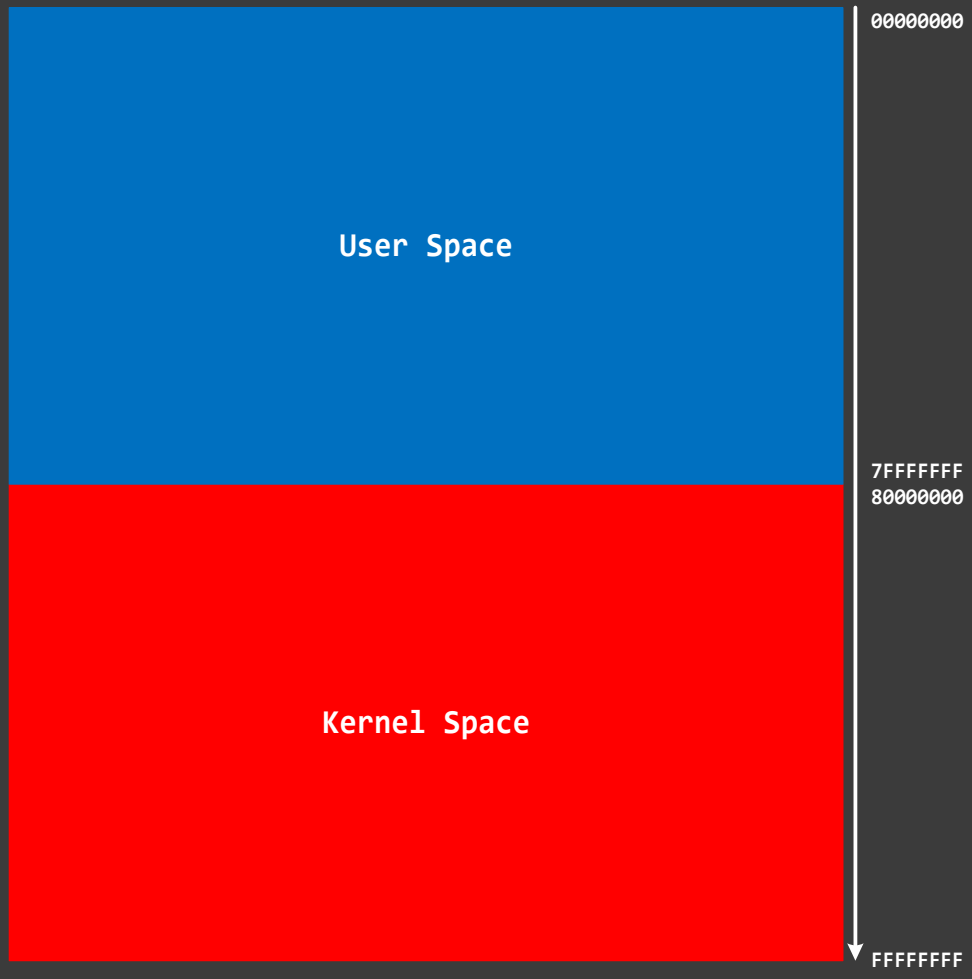
* Most of the exercises are focused on x64 code. For their x86 equivalents from older Windows versions, please refer to the previous fourth edition of this course.

Part 1A: Fundamentals

Process Space (x64)

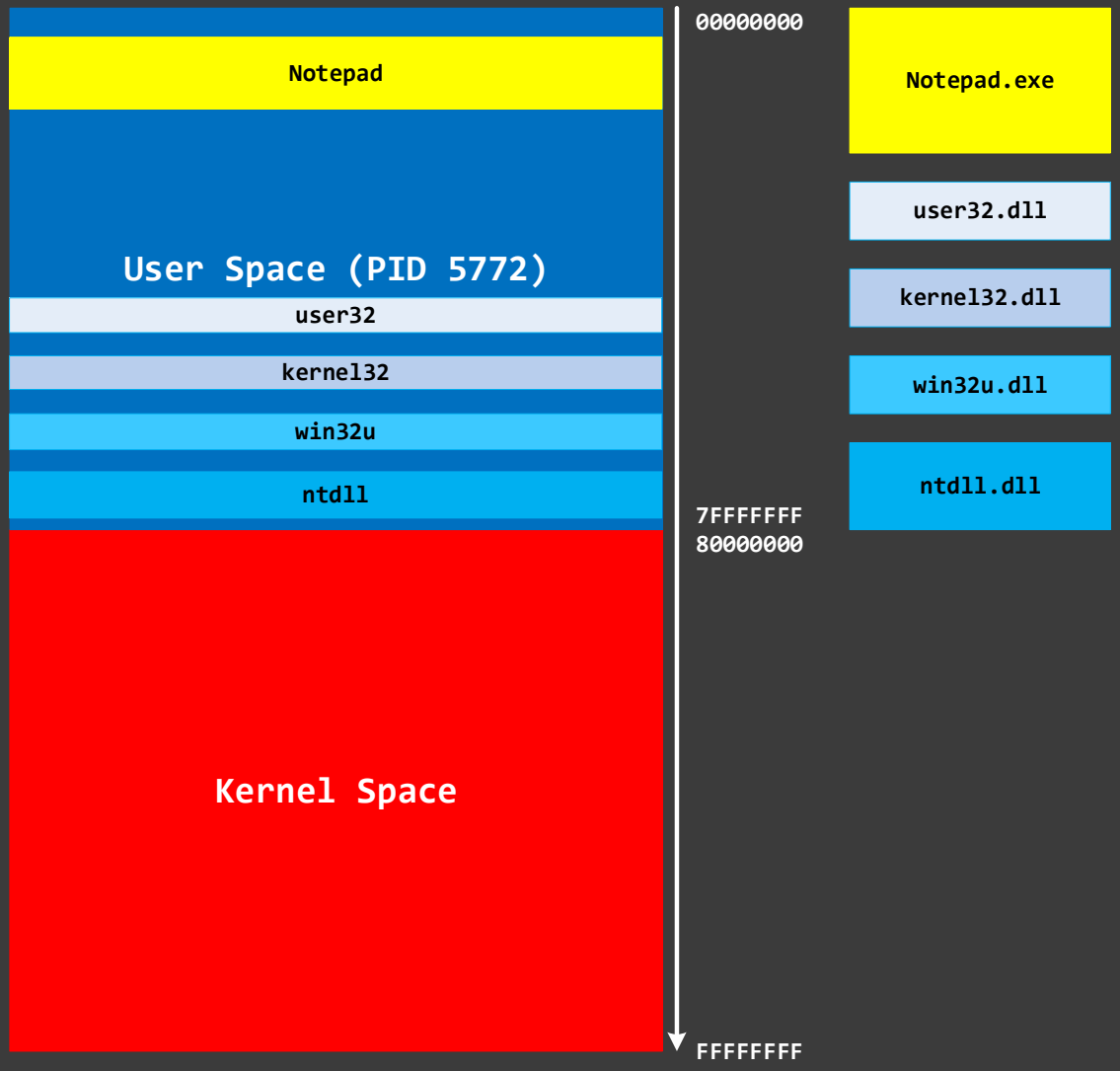


Process Space (x86)

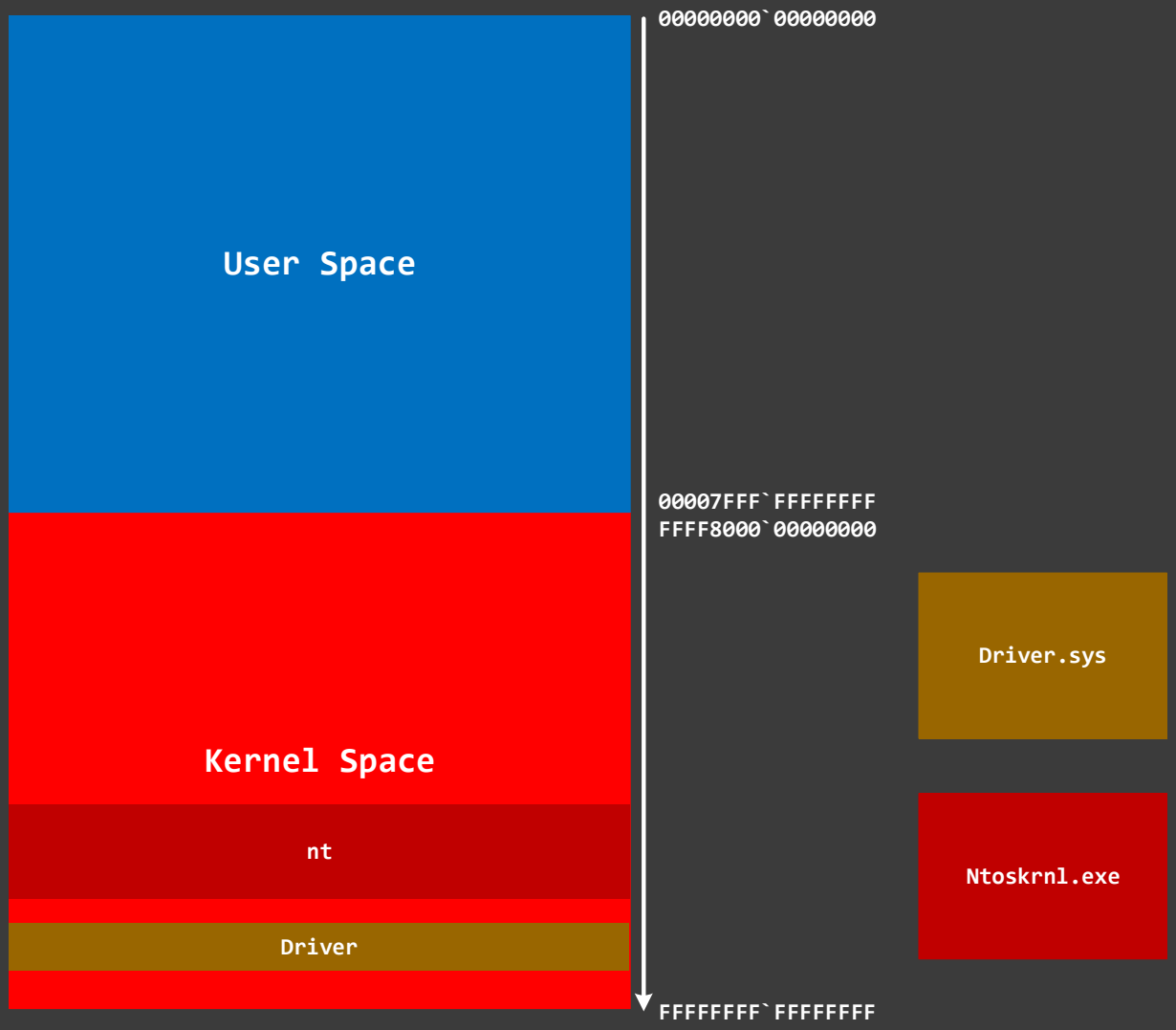


Application/Process/Module (x64)

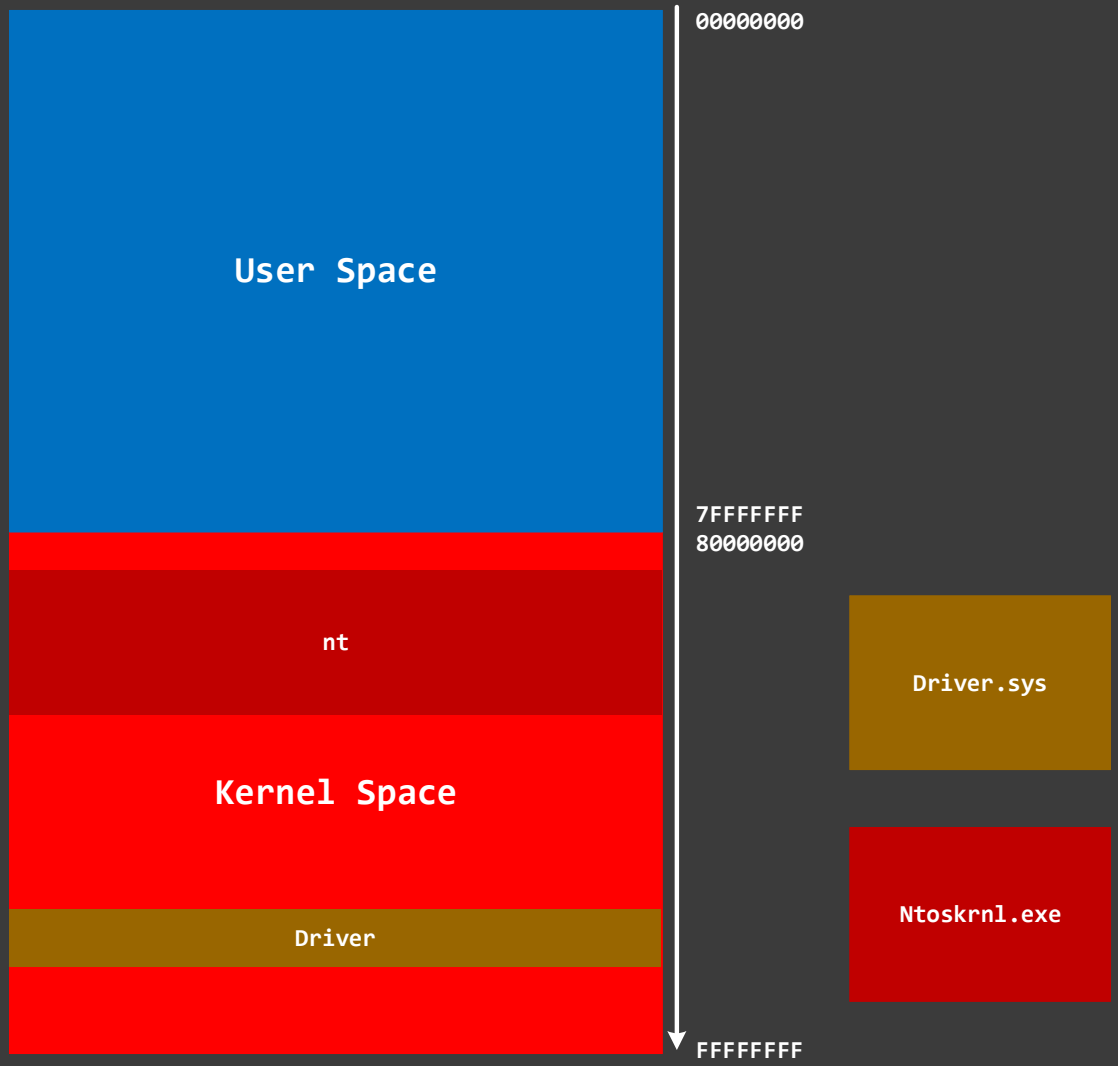
Application/Process/Module (x86)



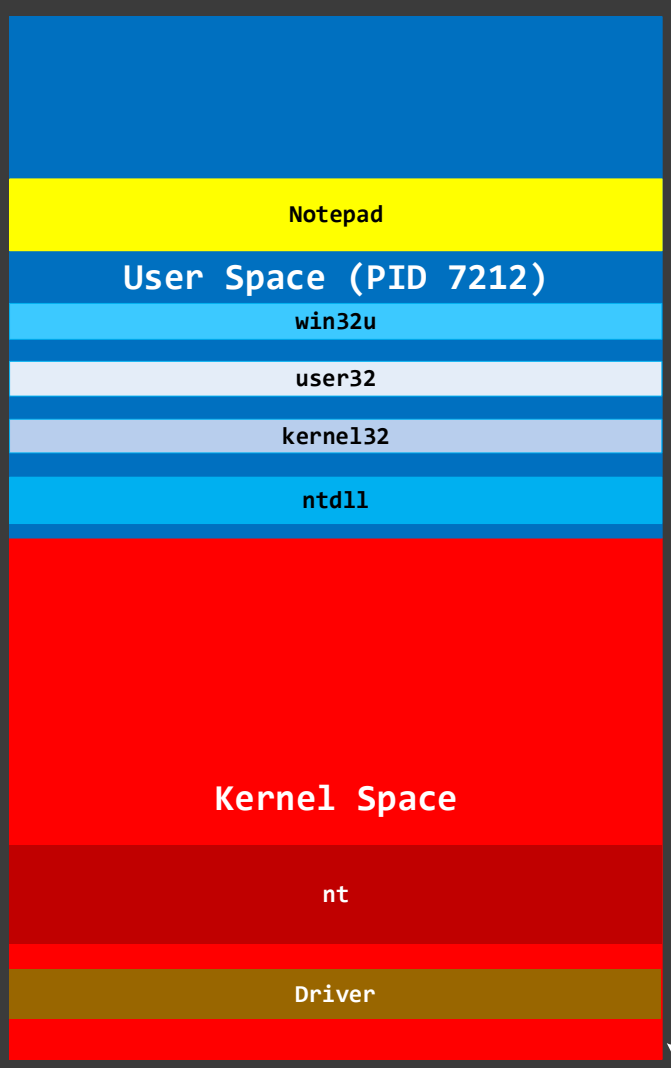
OS Kernel/Driver/Module (x64)



OS Kernel/Driver/Module (x86)



Process Virtual Space (x64)



00000000`00000000

00007FF6`00000000

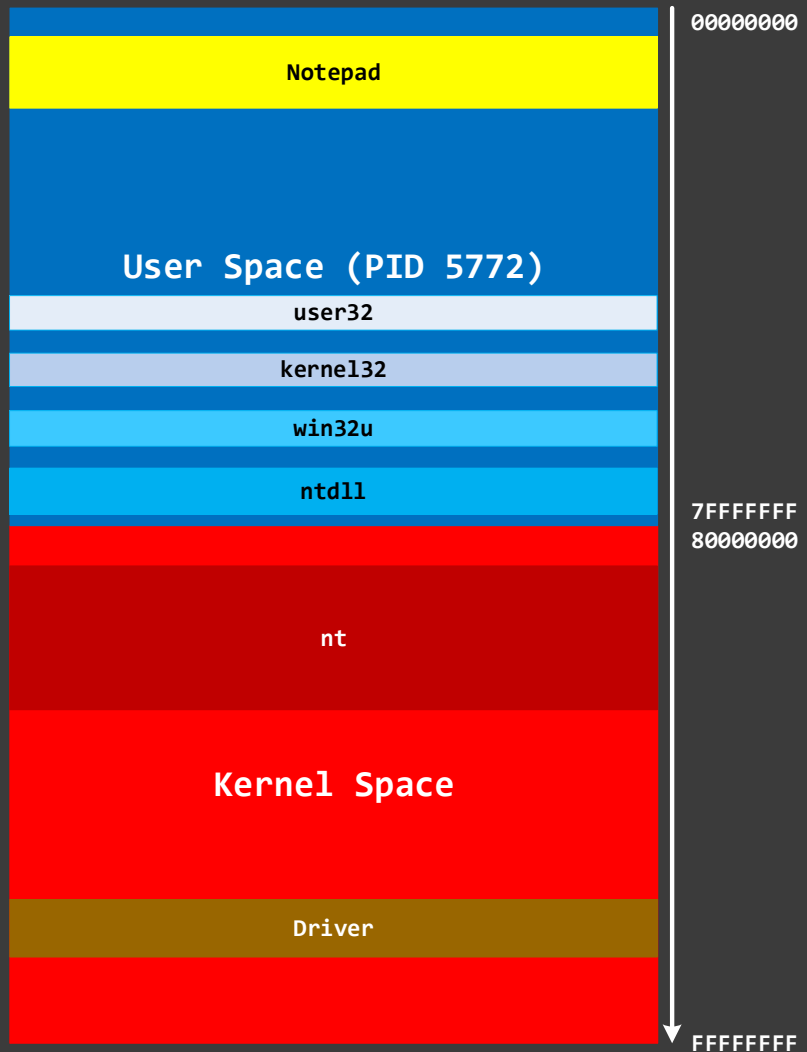
00007FFF`FFFFFFFF

FFFF8000`00000000

FFFFFFFF`FFFFFFFF

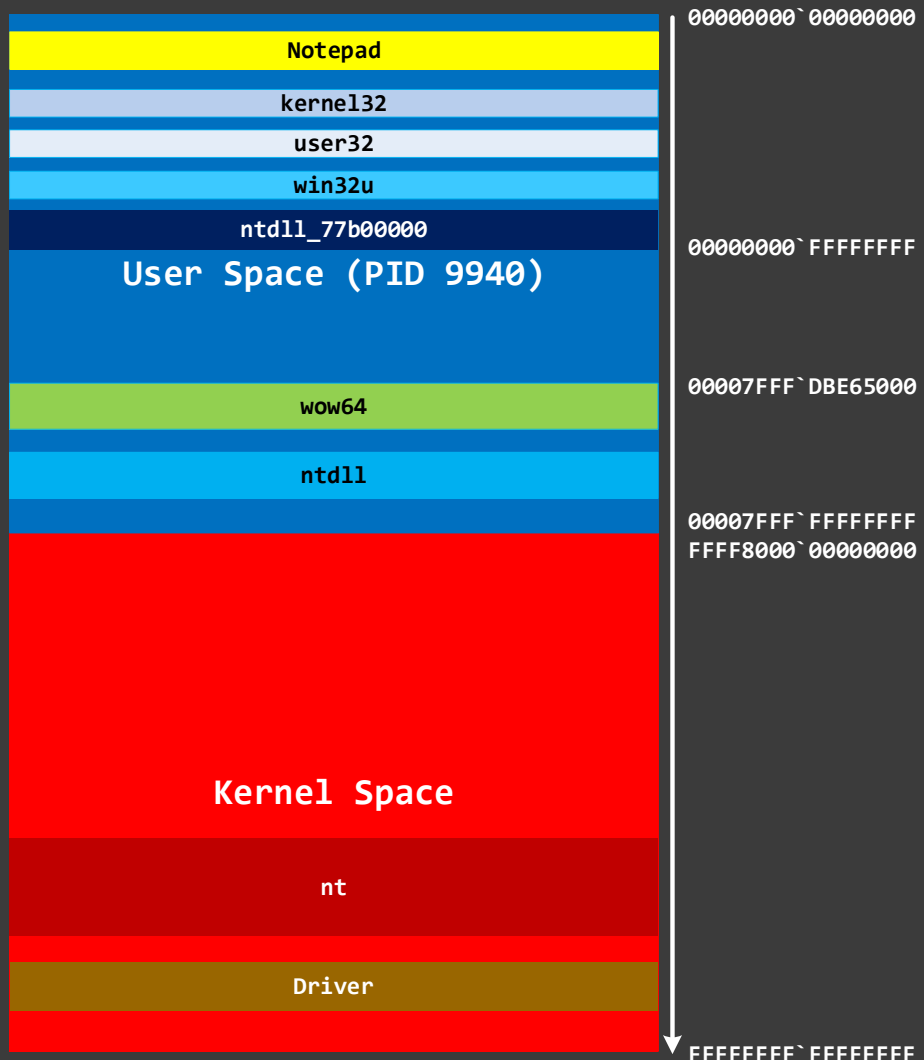
00000000`00000000 ...
FFFFFFFF`FFFFFFFF

Process Virtual Space (x86)

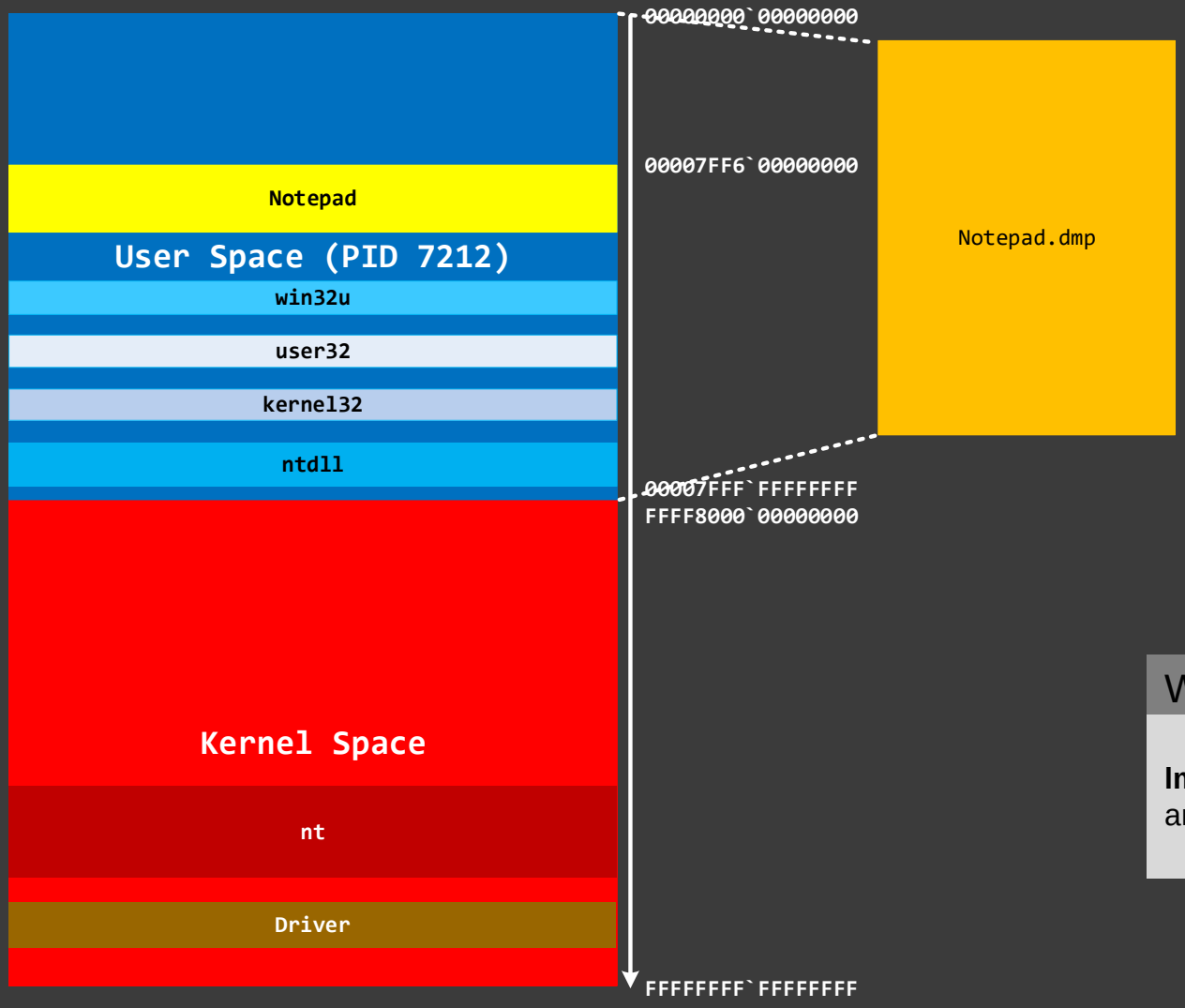


00000000 ... FFFFFFFF

Process Virtual Space (WOW64)



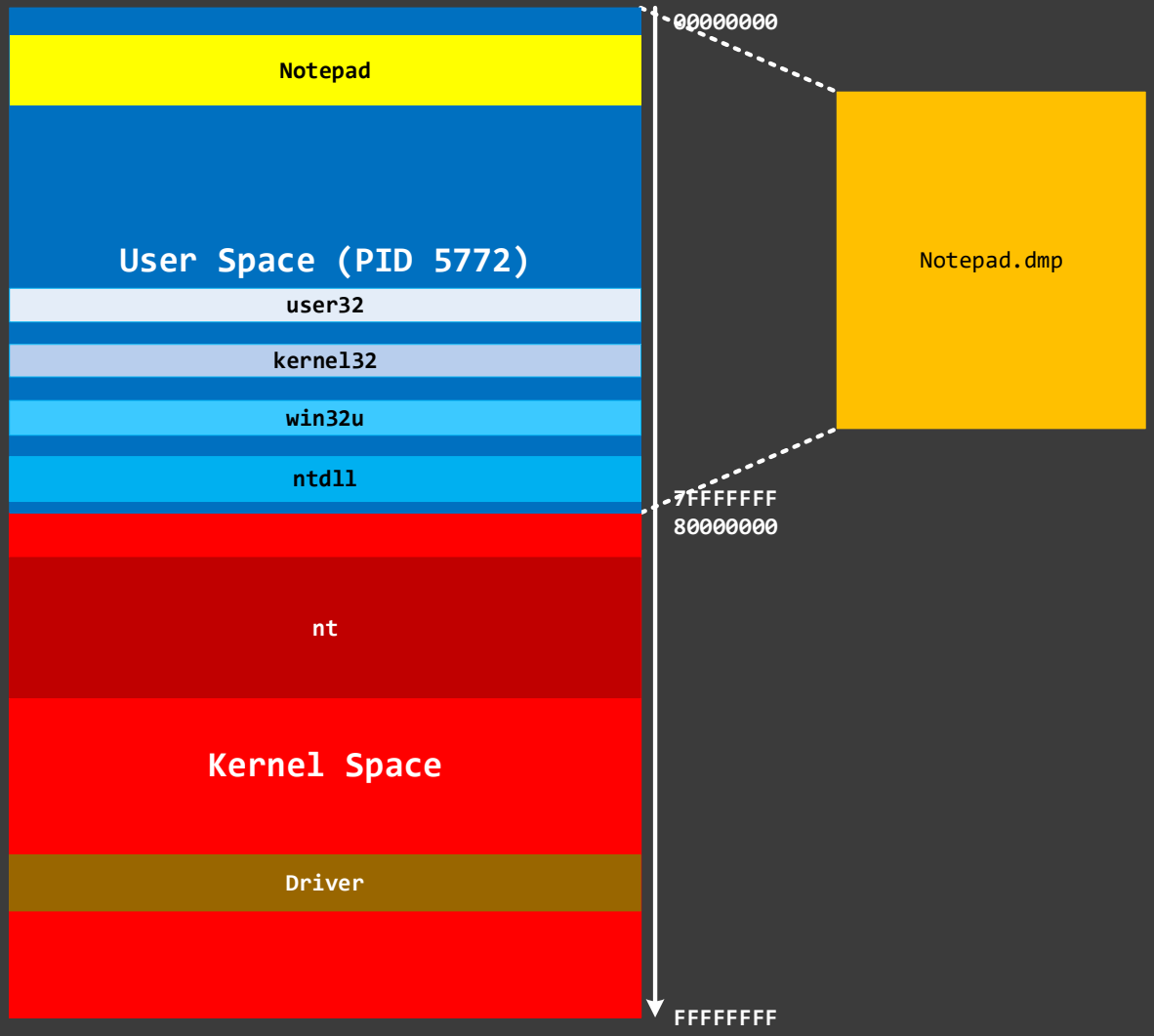
Process Memory Dump (x64)



WinDbg Commands

lmv command lists modules and their description

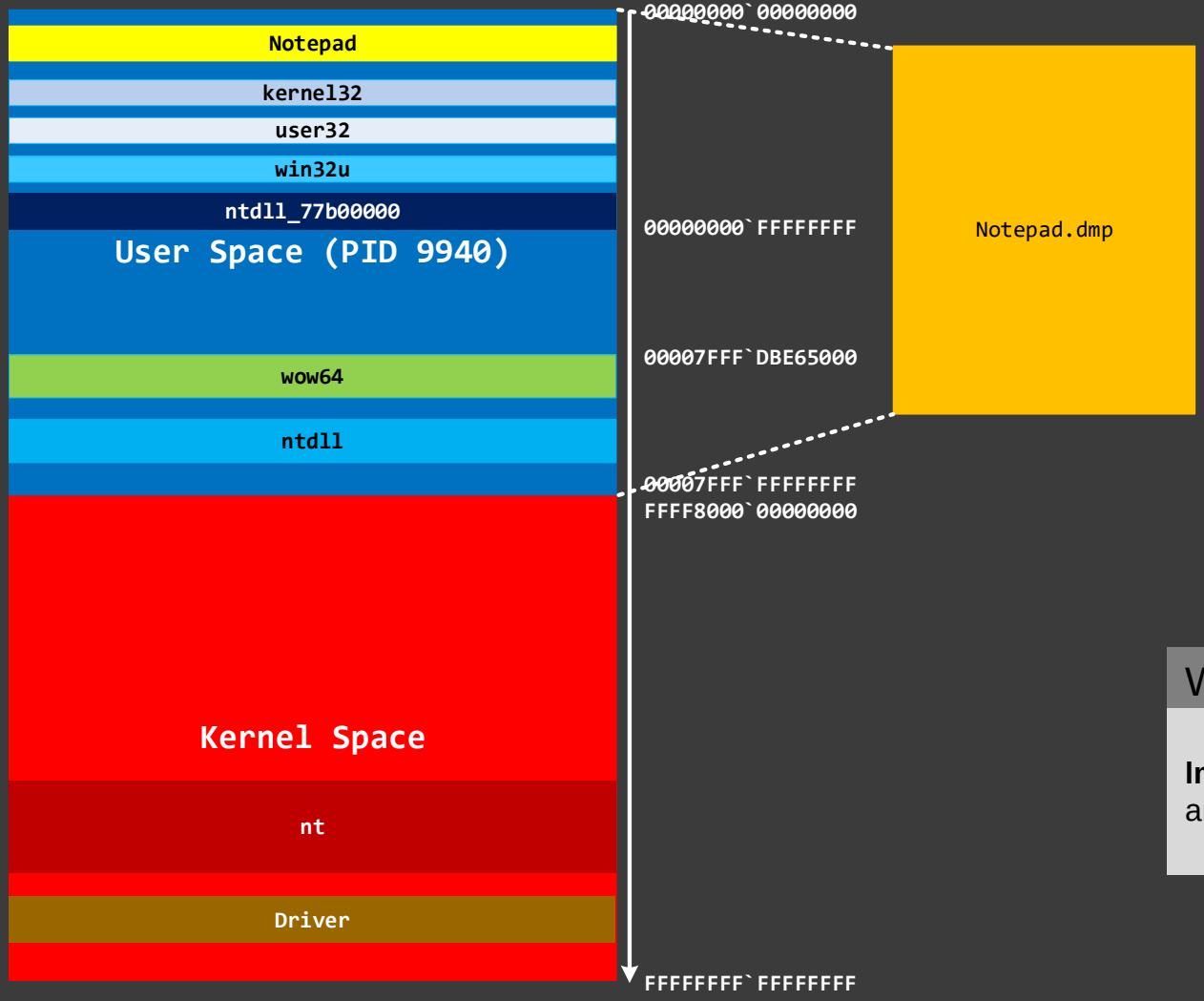
Process Memory Dump (x86)



WinDbg Commands

lmv command lists modules and their description

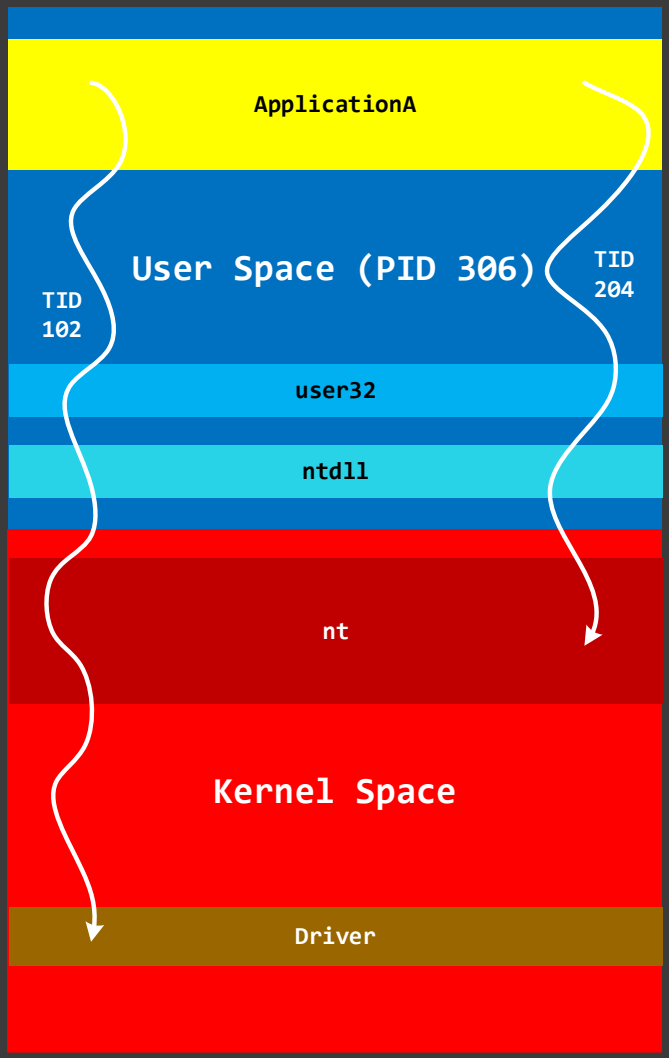
Process Memory Dump (WOW64)



WinDbg Commands

lmv command lists modules and their description

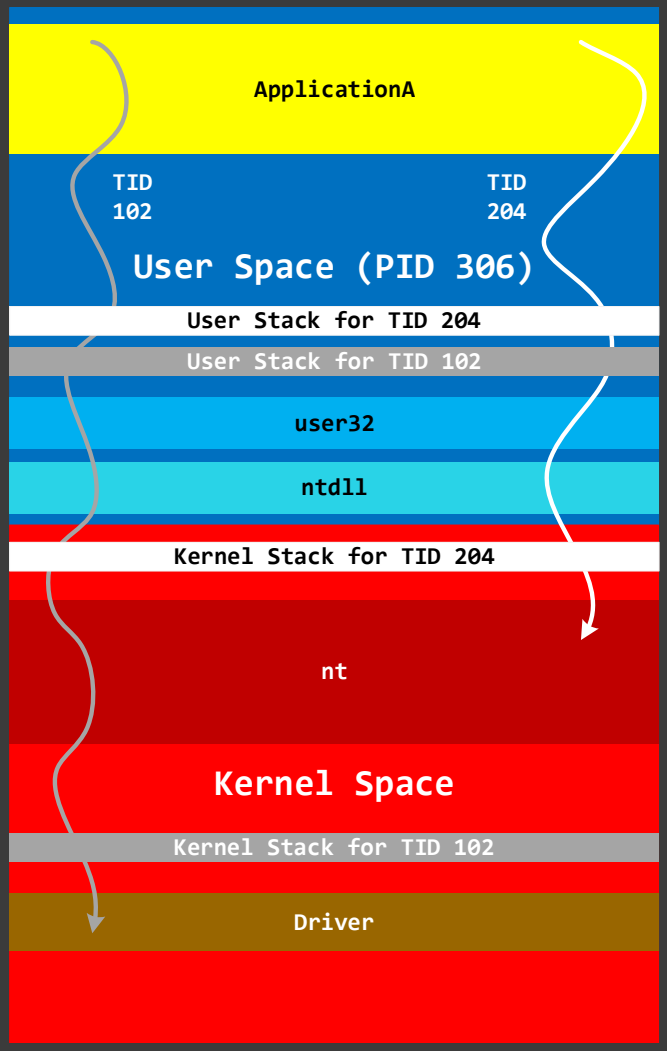
Process Threads



WinDbg Commands

Process dumps:
~<n>s switches between threads

Thread Stack Raw Data

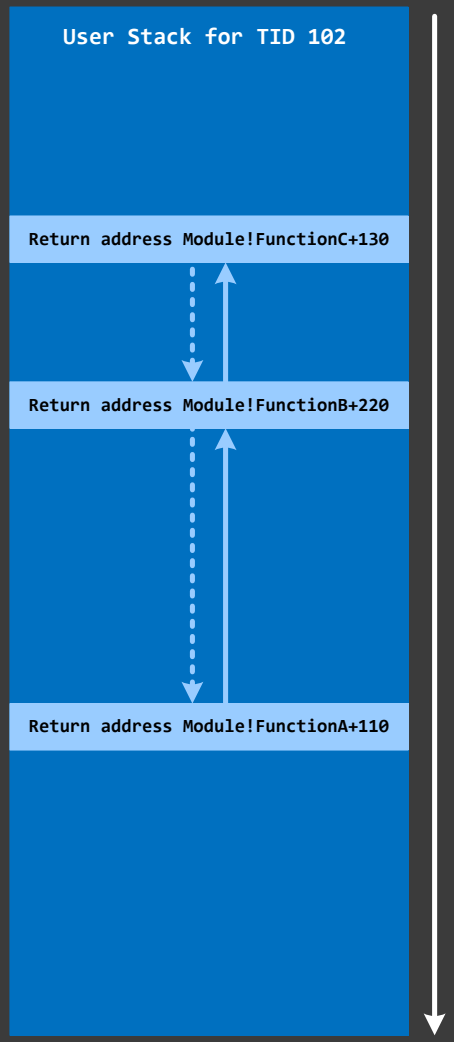


WinDbg Commands

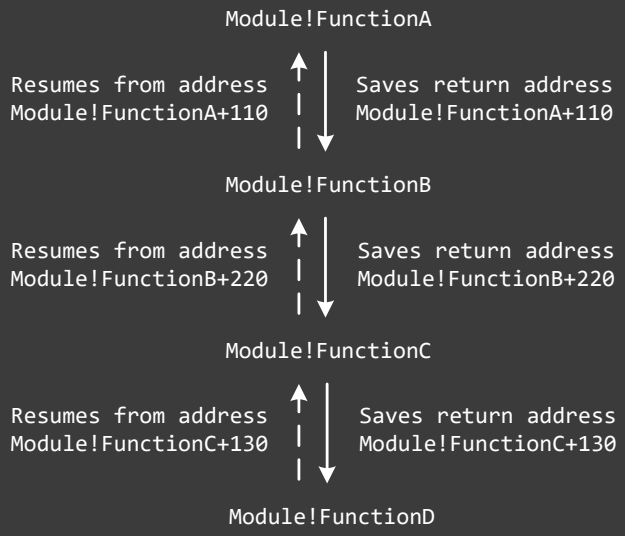
Process dumps:
!teb

Data:
dc / **dps** / **dpp** / **dpa** / **dpu**

Thread Stack Trace



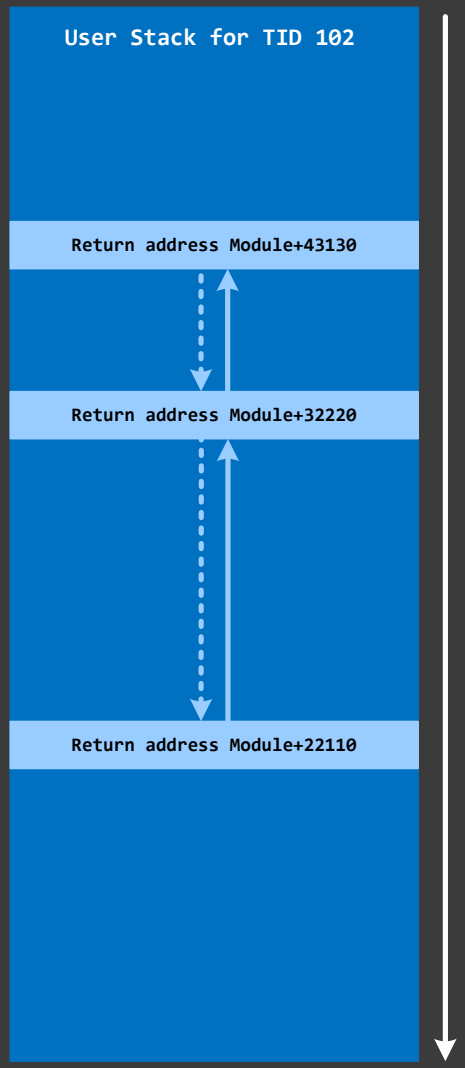

```
FunctionA()
{
  ...
  FunctionB();
  ...
}

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)

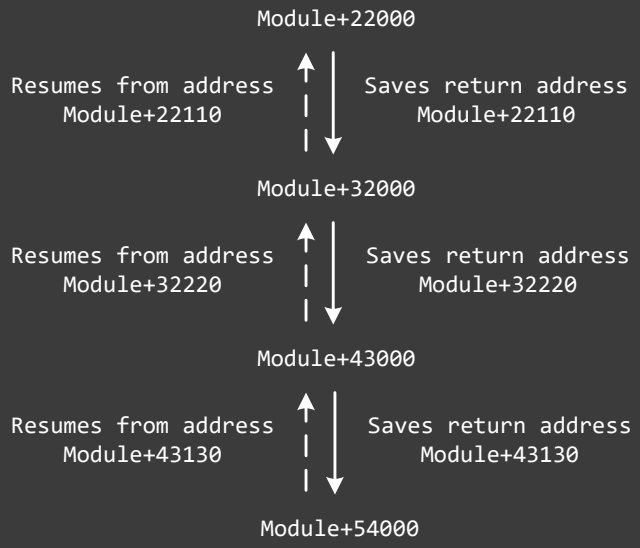



```
FunctionA()
{
  ...
  FunctionB();
  ...
}
FunctionB()
{
  ...
  FunctionC();
  ...
}
FunctionC()
{
  ...
  FunctionD();
  ...
}
```

Symbol file Module.pdb

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

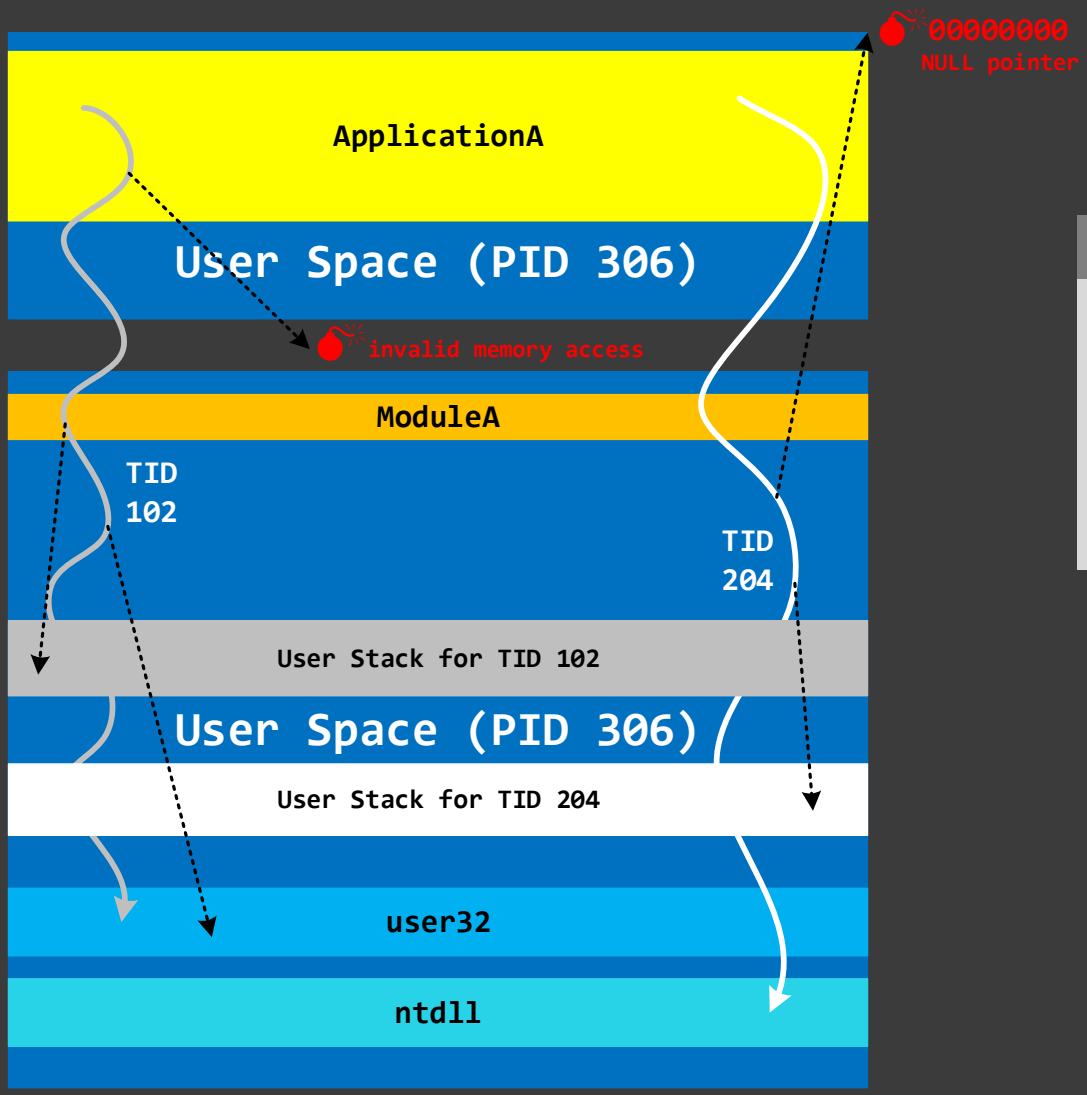
No symbols for Module



WinDbg Commands

```
0:000> k
Module+0
Module+43130
Module+32220
Module+22110
```

Exceptions (Access Violation)

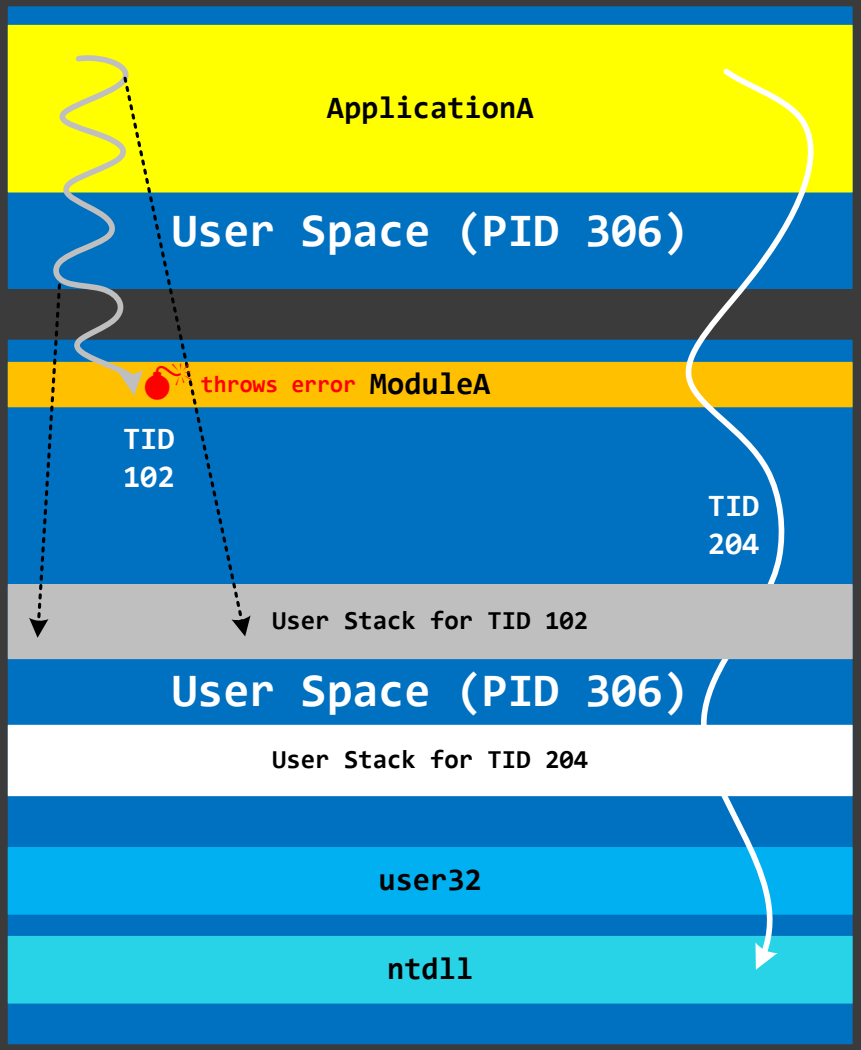


WinDbg Commands

```
address=????????
```

Set exception context (process dump):
.cxr

Exceptions (Runtime)



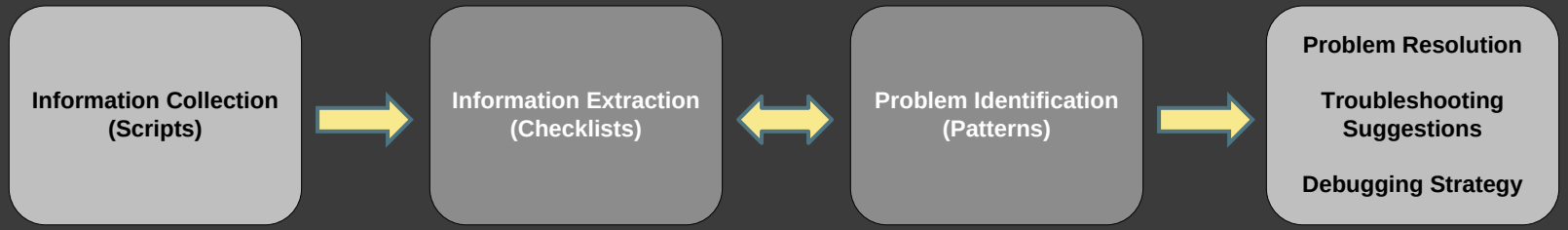
Pattern-Oriented Diagnostic Analysis

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

Patterns: http://www.dumpanalysis.org/blog/index.php/crash-dump-analysis-patterns/

Part 1B: x64 Disassembly

x64 CPU Registers

- **RAX** ⊃ **EAX** ⊃ **AX** ⊇ {**AH**, **AL**}

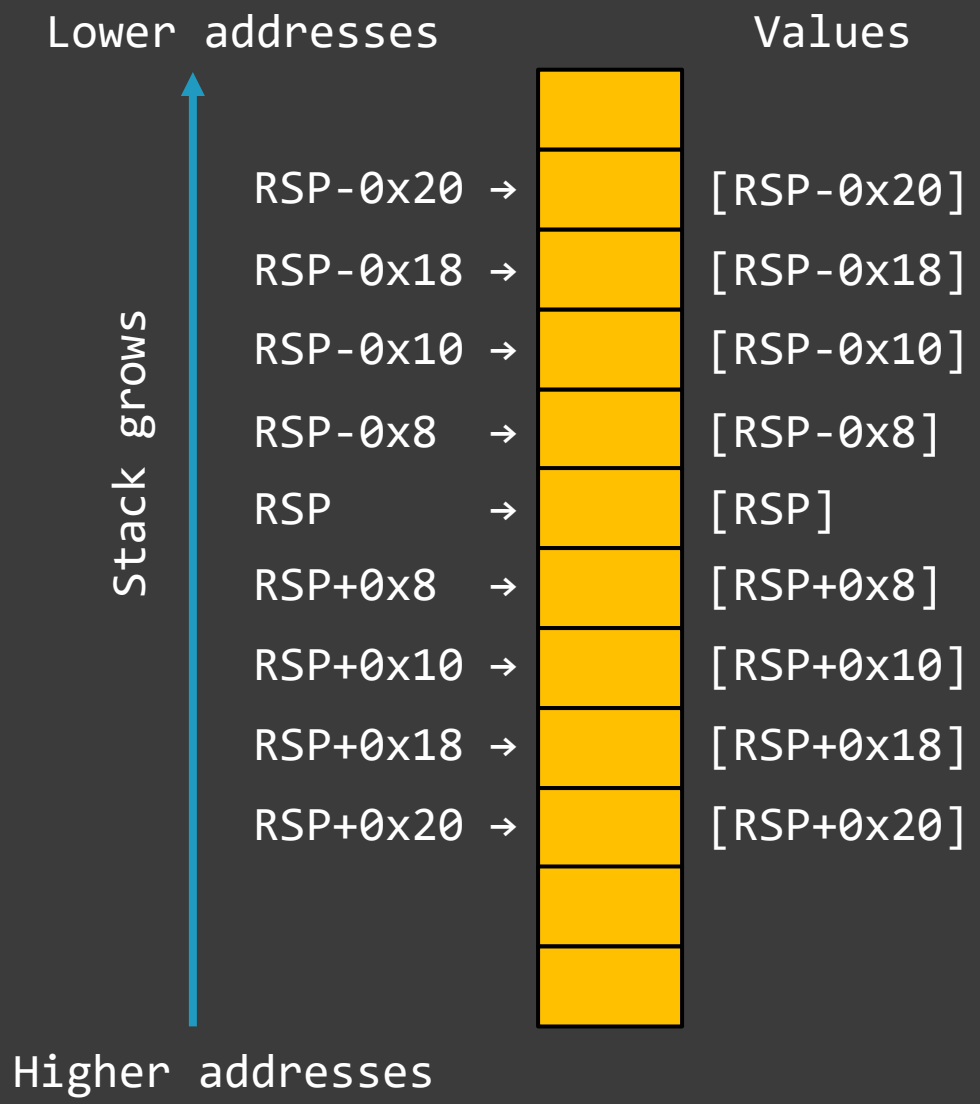
- ALU: **RAX**, **RDX**
- Counter: **RCX**
- Memory copy: **RSI** (src), **RDI** (dst)
- Stack: **RSP**
- Next instruction: **RIP**
- New: **R8** – **R15**, **Rx**(**D**|**W**|**B**)

Instructions and Registers

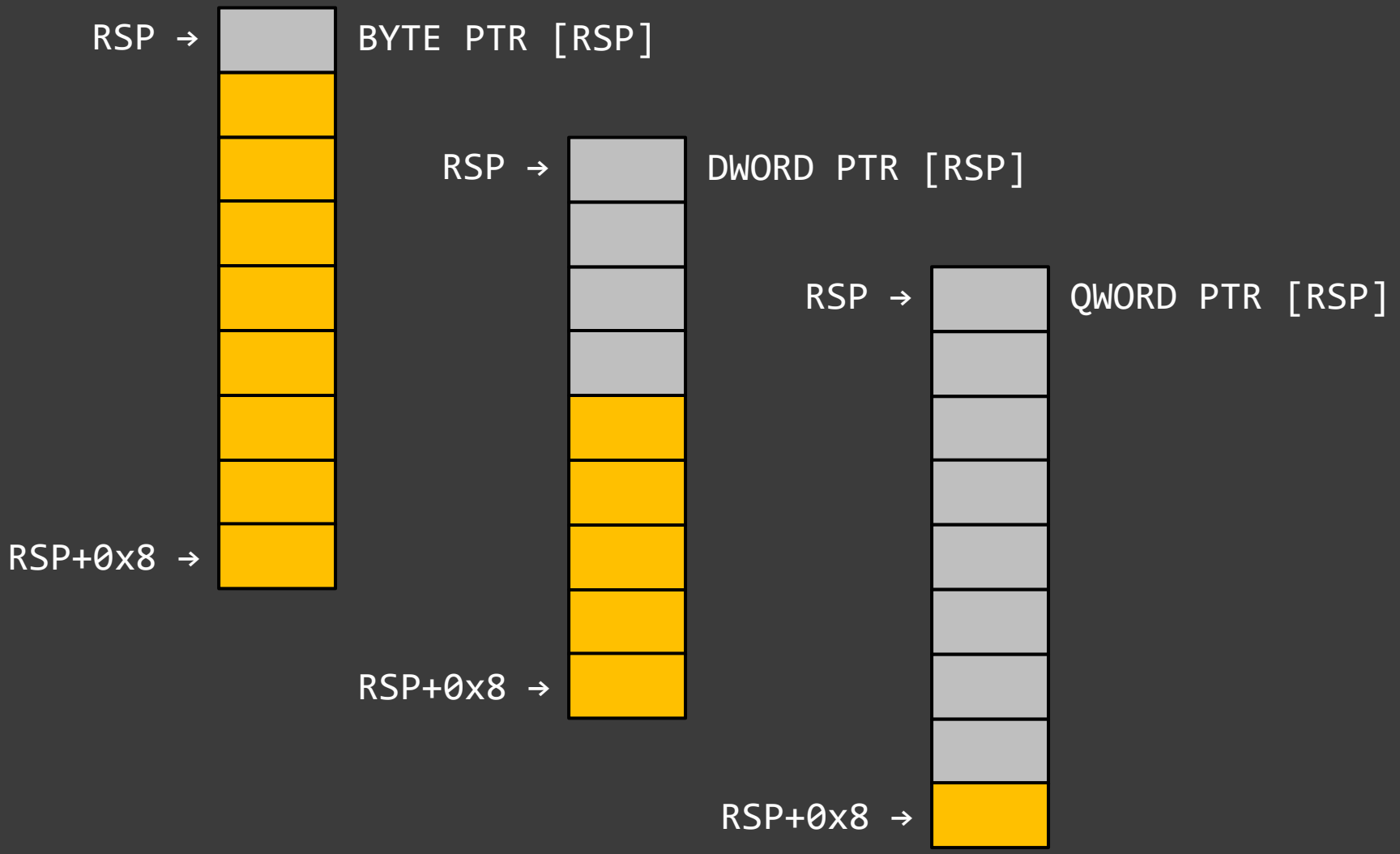
- `Opcode DST, SRC`
- Examples:

```
mov   rax, 10h          ; RAX ← 0x10
mov   r13, rdx          ; R13 ← RDX
add   r10, 10h          ; R10 ← R10 + 0x10
imul  edx, ecx          ; EDX ← EDX * ECX
call  rdx               ; RDX already contains
                        ;    the address of func (&func)
                        ; PUSH RIP; &func → RIP
sub   rsp, 30h          ; RSP ← RSP-0x30
                        ; make room for local variables
```

Memory and Stack Addressing



Memory Cell Sizes



Memory Load Instructions

- `Opcode DST, PTR [SRC+Offset]`
- `Opcode DST`
- Examples:

```
mov   rax, qword ptr [rsp+10h] ; RAX ←
                               ; 64-bit value at address RSP+0x10
mov   ecx, dword ptr [20]      ; ECX ←
                               ; 32-bit value at address 0x20
pop   rdi                      ; RDI ← value at address RSP
                               ; RSP ← RSP + 8
lea   r8, [rsp+20h]            ; R8 ← address RSP+0x20
```

Memory Store Instructions

- `Opcode PTR [DST+Offset], SRC`
- `Opcode DST|SRC`
- Examples:

```
mov   qword ptr [rbp-20h], rcx ; 64-bit value at address RBP-0x20
                               ;   ← RCX
mov   byte ptr [0], 1          ; 8-bit value at address 0 ← 1
push  rsi                      ; RSP ← RSP - 8
                               ; value at address RSP ← RSI
inc   dword ptr [rcx]          ; 32-bit value at address RCX ←
                               ;   1 + 32-bit value at address RCX
```

Flow Instructions

- `Opcode DST`
- `Opcode PTR [DST]`
- Examples:

```
jmp   00007ff6`9ef2f008   ; RIP ← 0x7ff69ef2f008
                          ; (goto 0x7ff69ef2f008)
jmp   qword ptr [rax+10h] ; RIP ← value at address RAX+0x10
call  00007ff6`9ef21400   ; RSP ← RSP – 8
00007ff6`9ef21057:        ; value at address RSP ← 0x7ff69ef21057
                          ; RIP ← 0x7ff69ef21400
                          ; (goto 0x7ff69ef21400)
```

Windows API Parameters

- x86: Right to left PUSH

  `Args to Child` are parameters

- x64: Left to right RCX, RDX, R8, R9, stack

  `Args to Child` are not parameters

WinDbg Commands

```
0:000> kv
 # Child-SP   RetAddr   : Args to Child   : Call Site
…
```

Part 1C: Practice Exercises

Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in this book

Exercise 0

- **Goal:** Install WinDbg or Debugging Tools for Windows, or pull Docker image, and check that symbols are set up correctly
- **Patterns:** Stack Trace; Incorrect Stack Trace
- \AWMDA-Dumps\Exercise-0-Download-Setup-WinDbg.pdf

Process Memory Dumps

Exercises P1 – P21

Exercise P1

- **Goal:** Learn how to see dump file type and version, get a stack trace, check its correctness, perform default analysis, list threads and modules, check module version information, dump module data, and check the process environment
- **Patterns:** Manual Dump (Process); Stack Trace; Not My Version (Software); Environment Hint; Unknown Component
- \AWMDA-Dumps\Exercise-P1-Analysis-normal-process-dump-wordpad-64.pdf

Exercise P2

- **Goal:** Repeat exercise P1 using 32-bit notepad process memory dump

- \AWMDA-Dumps\Exercise-P2-Analysis-normal-process-dump-wordpad-32.pdf

Exercise P3

- **Goal:** Learn how to list stack traces, check their correctness, perform default analysis, list modules, check their version information, and check thread age and CPU consumption
- **Patterns:** Stack Trace Collection (Unmanaged Space)
- \AWMDA-Dumps\Exercise-P3-Analysis-normal-process-dump-msedge-64.pdf

Exercise P4

- **Goal:** Learn to recognize exceptions in process memory dumps and get their context
- **Patterns:** Exception Stack Trace; Exception Module; Multiple Exceptions (User Mode); NULL Pointer (Data)
- \AWMDA-Dumps\Exercise-P4-Analysis-process-dump-AppK-64-no-symbols.pdf

Exercise P5

- **Goal:** Learn how to load application symbols
- \AWMDA-Dumps\Exercise-P5-Analysis-process-dump-AppK-64-with-symbols.pdf

Exercise P6

- **Goal:** Learn how to recognize heap corruption, dump memory contents, follow critical section wait chains, and check error and status codes
- **Patterns:** Dynamic Memory Corruption (Process Heap); Wait Chain (Critical Sections); Execution Residue (Unmanaged Space, User); Last Error Collection
- \AWMDA-Dumps\Exercise-P6-Analysis-process-dump-AppL-64.pdf

Exercise P7

- **Goal:** Learn how to debug heap corruption using page heap
- **Patterns:** Invalid Pointer (General); Instrumentation Information
- \AWMDA-Dumps\Exercise-P7-Analysis-process-dump-AppL2-64.pdf

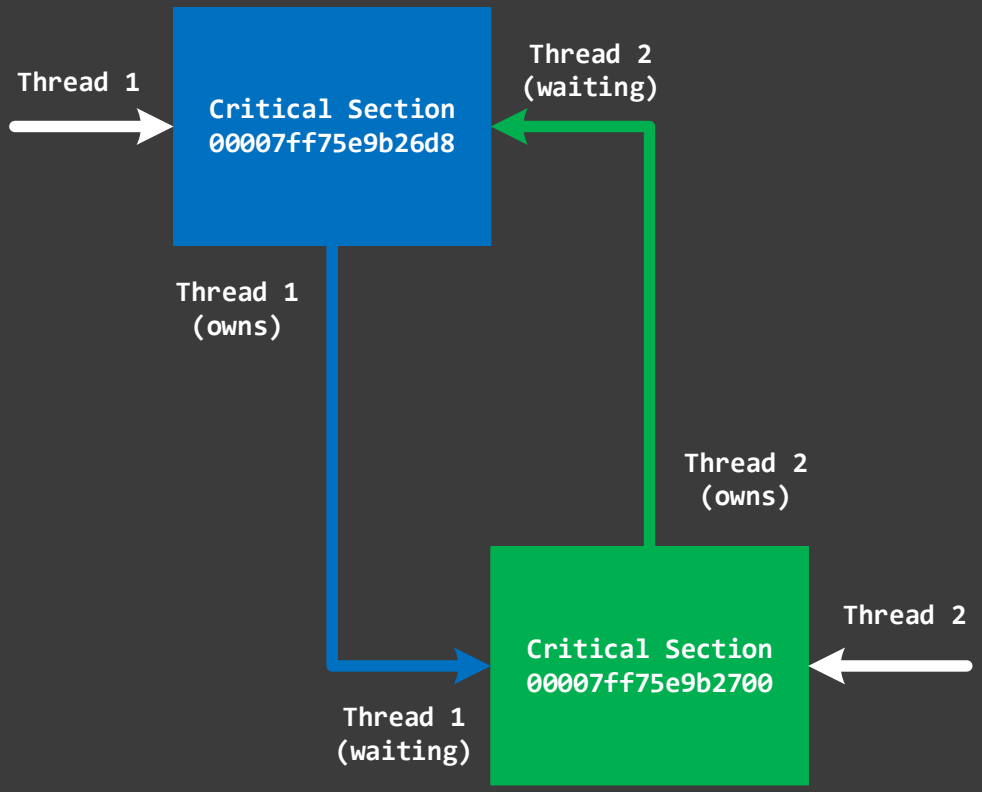
Exercise P8

- **Goal:** Learn how to recognize CPU spikes, invalid pointers, disassemble code, and reconstruct stack trace
- **Patterns:** Wild Code; Active Thread; Spiking Thread; NULL Pointer (Code); Truncated Stack Trace; Stored Exception
- \AWMDA-Dumps\Exercise-P8-Analysis-process-dump-AppM-64.pdf

Exercise P9

- **Goal:** Learn how to recognize critical section waits and deadlocks, dump raw stack data, and see hidden exceptions
- **Patterns:** Deadlock (Critical Sections); Hidden Exception (User Space)
- \AWMDA-Dumps\Exercise-P9-Analysis-process-dump-AppN-64.pdf

Deadlock



Exercise P10

- **Goal:** Learn how to recognize application heap problems, buffer and stack overflow patterns, and analyze raw stack data
- **Patterns:** Double Free (Process Heap); Local Buffer Overflow (User Space); Stack Overflow (User Mode)
- \AWMDA-Dumps\Exercise-P10-Analysis-process-dump-AppO-64.pdf

Exercise P11

- **Goal:** Learn how to analyze exception patterns, raw stacks, and execution residue
- **Patterns:** Divide by Zero (User Mode); C++ Exception; Execution Residue (Unmanaged Space, User)
- \AWMDA-Dumps\Exercise-P11-Analysis-process-dump-AppP-64.pdf

Exercise P12

- **Goal:** Learn how to analyze managed space
- **Patterns:** Platform-Specific Debugger; CLR Thread; JIT Code (.NET); Managed Code Exception; Managed Stack Trace
- \AWMDA-Dumps\Exercise-P12-Analysis-process-dump-AppR2-64.pdf

Exercise P13

- **Goal:** Learn how to analyze the 32-bit process saved as a 64-bit process memory dump
- **Patterns:** Virtualized Process (WOW64); Message Box; Debugger Bug; Rough Stack Trace (Unmanaged Space)
- \AWMDA-Dumps\Exercise-P13-Analysis-process-dump-AppA-WOW64.pdf

Exercise P14

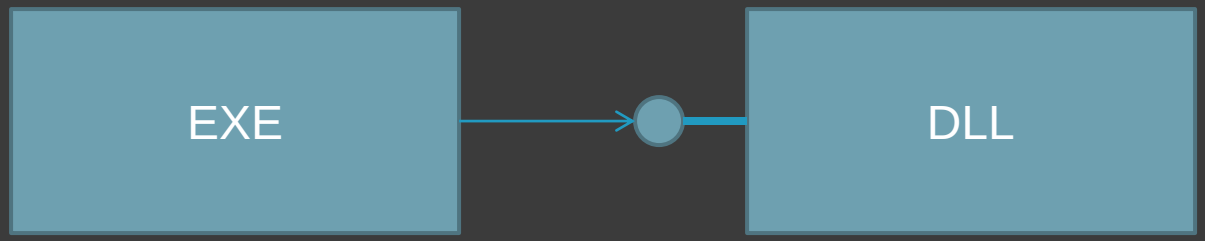
- **Goal:** Learn how to analyze process memory leaks
- **Patterns:** Thread Age; Memory Leak (Process Heap)
- \AWMDA-Dumps\Exercise-P14-Analysis-process-dump-AppS-64.pdf

Parameters and Locals

Debugging TV Frames episode 0x18

Symbol Types

- Exported and imported names



- Function and variable names
- Data types

Exercise P15

- **Goal:** Learn how to navigate function parameters in cases of reduced symbolic information in 32-bit process memory dumps
- **Patterns:** Reduced Symbolic Information
- \AWMDA-Dumps\Exercise-P15-Analysis-process-dump-notepad-32.pdf

Exercise P16

- **Goal:** Learn how to navigate function parameters in x64 process memory dumps
- **Patterns:** False Function Parameters; Injected Symbols
- \AWMDA-Dumps\Exercise-P16-Analysis-process-dump-notepad-64.pdf

Exercise P17

- **Goal:** Learn how to navigate object wait chains in 32-bit memory dumps saved with ProcDump
- **Patterns:** Embedded Comments; Wait Chain (General); No Data Types; Deadlock (Mixed Objects, User Space)
- \AWMDA-Dumps\Exercise-P17-Analysis-process-dump-AppQ-32.pdf

Exercise P18

- **Goal:** Learn how to navigate object wait chains in 64-bit memory dumps saved with ProcDump
- **Patterns:** Not My Thread; Blocked Thread (Software); Main Thread; Passive Thread (User Space); Coincidental Symbolic Information
- \AWMDA-Dumps\Exercise-P18-Analysis-process-dump-AppQ-64.pdf

Exercise P19

- **Goal:** Learn how to analyze process handle leaks
- **Patterns:** Active Space; Handle Leak
- \AWMDA-Dumps\Exercise-P19-Analysis-process-dump-AppT-64.pdf

Exercise P20

- **Goal:** Learn how to analyze service memory dumps
- **Patterns:** Input Thread; Blocking Module
- \AWMDA-Dumps\Exercise-P20-Analysis-process-dump-ServiceA-64.pdf

Exercise P21

- **Goal:** Learn how to analyze memory dumps from Rust processes
- **Patterns:** Language-Specific Subtrace (Rust)
- \AWMDA-Dumps\Exercise-P21-Analysis-process-dump-rusty.pdf

Pattern Links

Spiking Thread
C++ Exception
Divide by Zero (User Mode)
Dynamic Memory Corruption (Process Heap)
Execution Residue (Unmanaged Space, User)
Invalid Pointer (General)
Manual Dump (Process)
Managed Stack Trace
Not My Version (Software)
NULL Pointer (Code)
Stack Trace Collection (Unmanaged Space)
Environment Hint
Unknown Component
Virtualized Process (WOW64)
False Function Parameters
Reduced Symbolic Information
Stored Exception
Instrumentation Information
JIT Code (.NET)
Embedded Comment
Deadlock (Mixed Object, User Space)
Blocked Thread (Software)
Passive Thread (User Space)
Rough Stack Trace (Unmanaged Space)
Memory Leak (Process Heap)

CLR Thread
Deadlock (Critical Sections)
Double Free (Process Heap)
Exception Stack Trace
Hidden Exception (User Space)
Local Buffer Overflow (User Space)
Managed Code Exception
Multiple Exceptions (User Mode)
NULL Pointer (Data)
Stack Trace
Stack Overflow (User Mode)
Wild Code
Wait Chain (Critical Sections)
Message Box
Injected Symbols
Truncated Stack Trace
Incorrect Stack Trace
Active Thread
Thread Age
Wait Chain (General)
Not My Thread
Main Thread
Coincidental Symbolic Information
Platform-Specific Debugger
Language-Specific Subtrace (Rust)

Active Space
Debugger Bug
Exception Module
Blocking Module
Last Error Collection
Handle Leak
Input Thread

Pattern Classification

Space/Mode
Hooksware
DLL Link Patterns
Contention Patterns
Stack Trace Patterns
Exception Patterns
Module Patterns
Thread Patterns
Dynamic Memory Corruption Patterns
.NET / CLR / Managed Space Patterns
Falsity and Coincidence Patterns
Hidden Artifact Patterns
Frame Patterns

Memory dump type
Wait Chain Patterns
Insufficient Memory Patterns
Stack Overflow Patterns
Symbol Patterns
Meta-Memory Dump Patterns
Optimization Patterns
Process Patterns
Deadlock and Livelock Patterns
Executive Resource Patterns
RPC, LPC and ALPC Patterns
Pointer Patterns
CPU Consumption Patterns

Pattern Case Studies

More than 70 multiple pattern case studies:

http://www.dumpanalysis.org/blog/index.php/pattern-cooperation/

Pattern Interaction chapters in
Memory Dump Analysis Anthology

Additional Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- Windows Internals, 6th ed. (Chapter 14. Crash Dump Analysis), 7th ed.
- Advanced Windows Debugging
- Inside Windows Debugging
- Principles of Memory Dump Analysis
- Windows Debugging Notebook: Essential User Space WinDbg Commands
- Encyclopedia of Crash Dump Analysis Patterns, 3rd edition
- Memory Dump Analysis Anthology (Diagnomicon)

Further Training Courses

- Accelerated Windows Memory Dump Analysis, 6th Edition, Part 2
- Practical Foundations of Windows Debugging, Disassembling, Reversing, 2nd Edition
- Advanced Windows Memory Dump Analysis with Data Structures, 4th Edition, Revised
- Accelerated .NET Core Memory Dump Analysis, Revised Edition
- Accelerated Windows Malware Analysis with Memory Dumps, 3rd Edition
- Accelerated Disassembly, Reconstruction and Reversing, 2nd Revised Edition
- Accelerated Windows Debugging4, 3rd Edition
- Extended Windows Memory Dump Analysis
- Accelerated Windows API for Software Diagnostics

Q&A

Please send your feedback using the contact form on PatternDiagnostics.com

Thank you for attendance!