Parts 1-2

Rust

Windows Memory Dump Analysis

Accelerated

Dmitry Vostokov
Software Diagnostics Services

Prerequisites

WinDbg Commands

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

- Basic Windows troubleshooting
- Basic Rust knowledge

© 2024 Software Diagnostics Services

Training Goals

- Review fundamentals
- Review x64 disassembly
- Learn how to analyze process dumps
- Learn how to analyze complete (physical memory) dumps

© 2024 Software Diagnostics Services

Training Principles

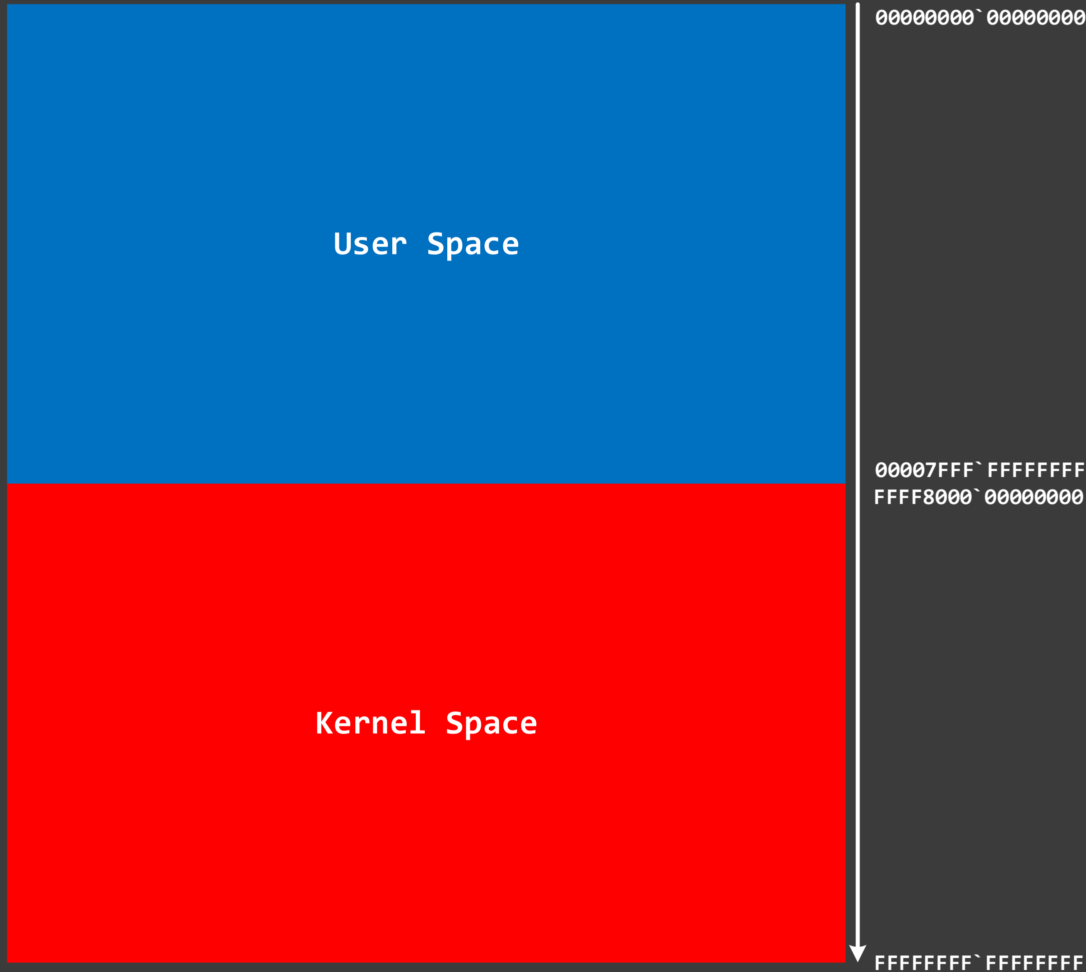
- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content and examples

© 2024 Software Diagnostics Services

Fundamentals

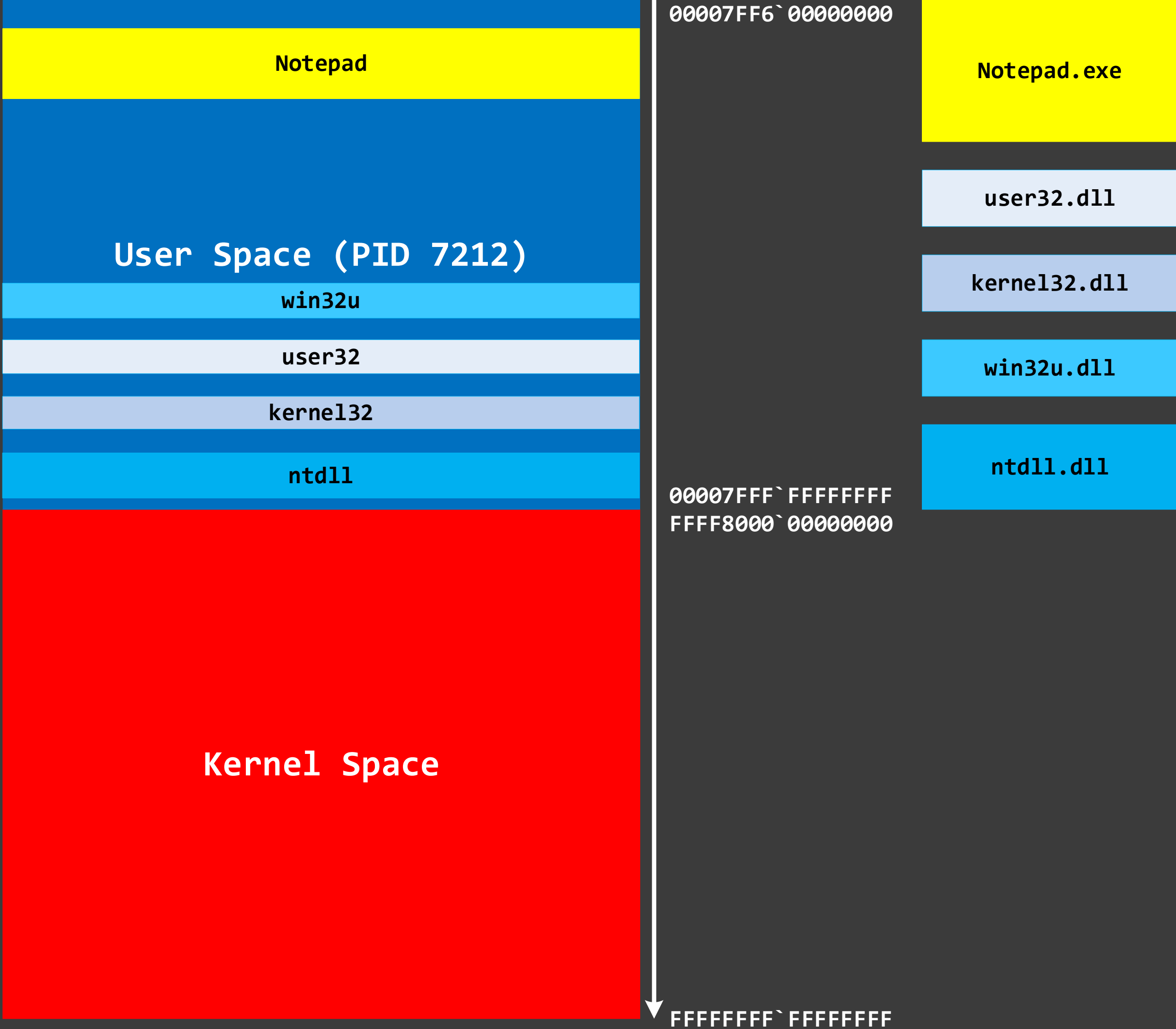
© 2024 Software Diagnostics Services

Process Space (x64)



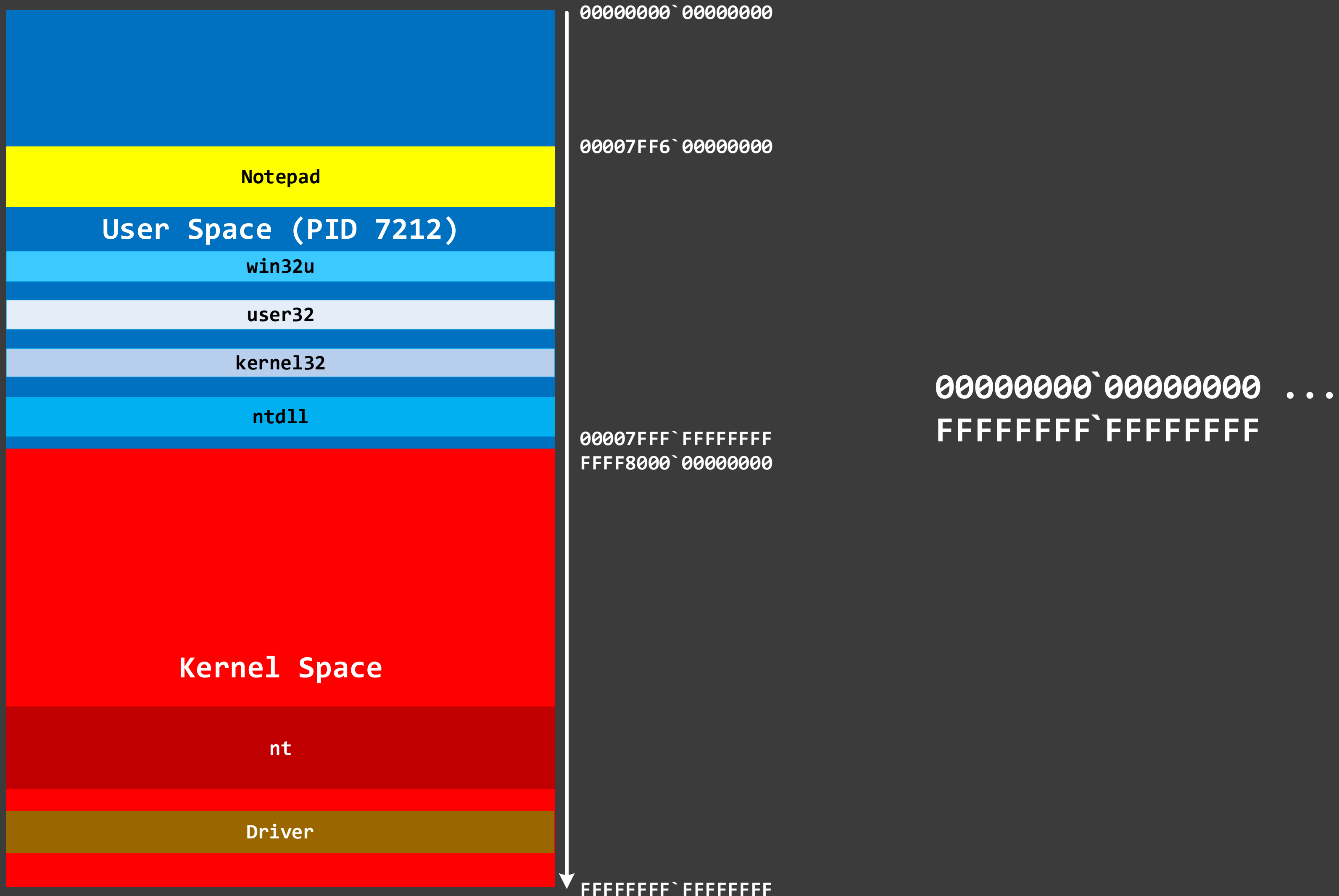
© 2024 Software Diagnostics Services

Application/Process/Module (x64)



© 2024 Software Diagnostics Services

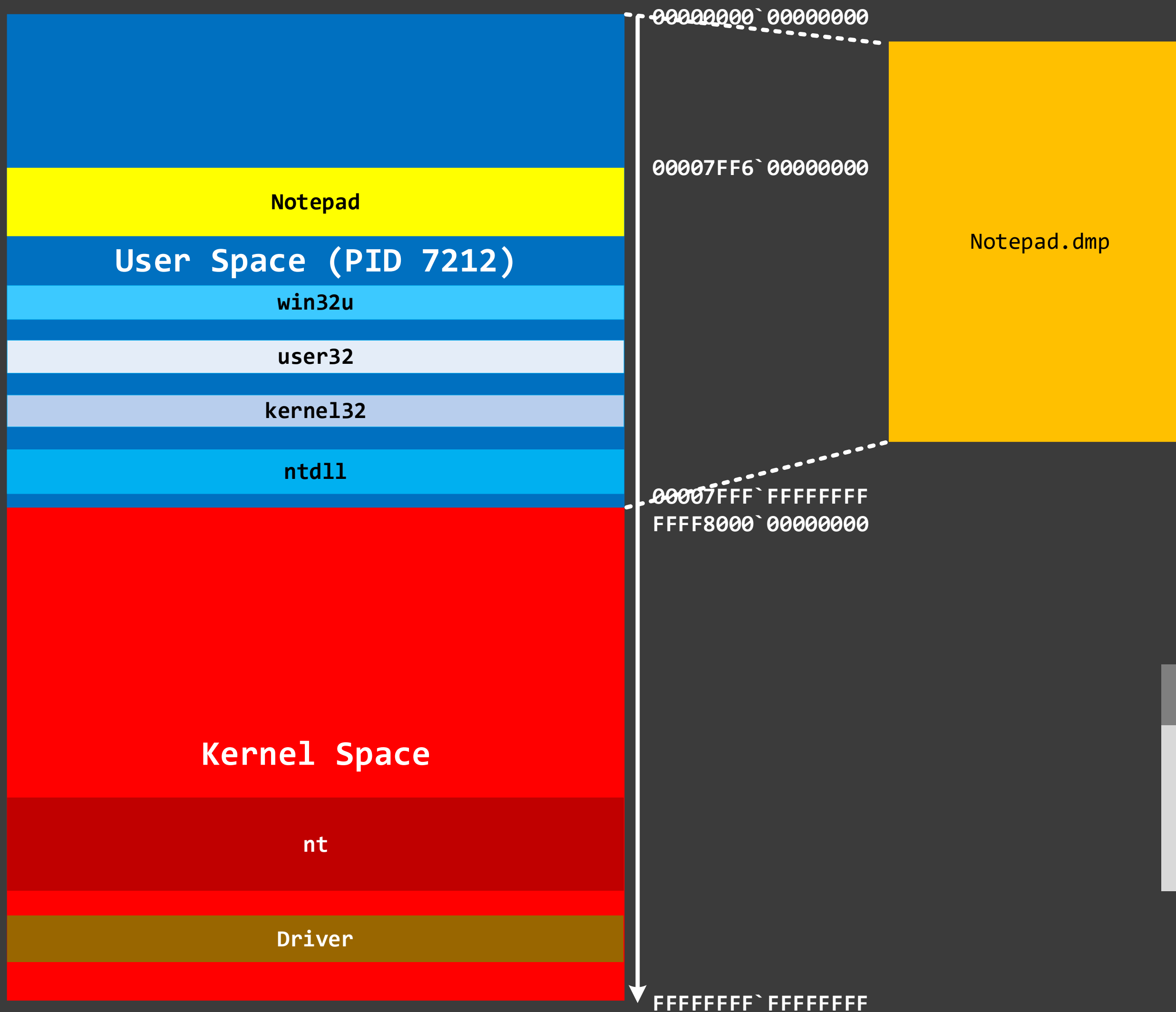
Process Virtual Space (x64)



00000000`00000000 ...
FFFFFFFF`FFFFFFFF

© 2024 Software Diagnostics Services

Process Memory Dump (x64)

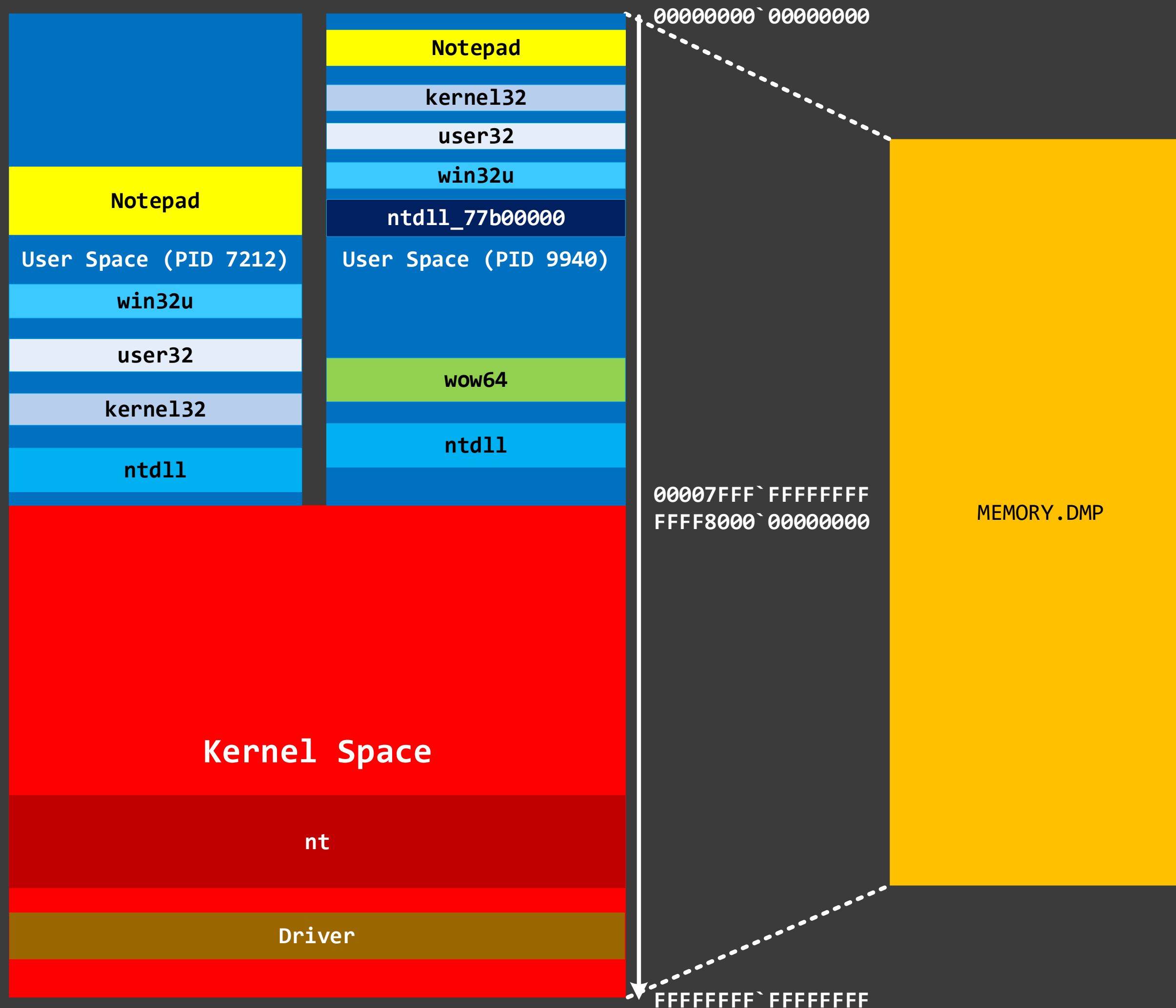


WinDbg Commands

lmv command lists modules and their description

© 2024 Software Diagnostics Services

Complete Memory Dump (x64)

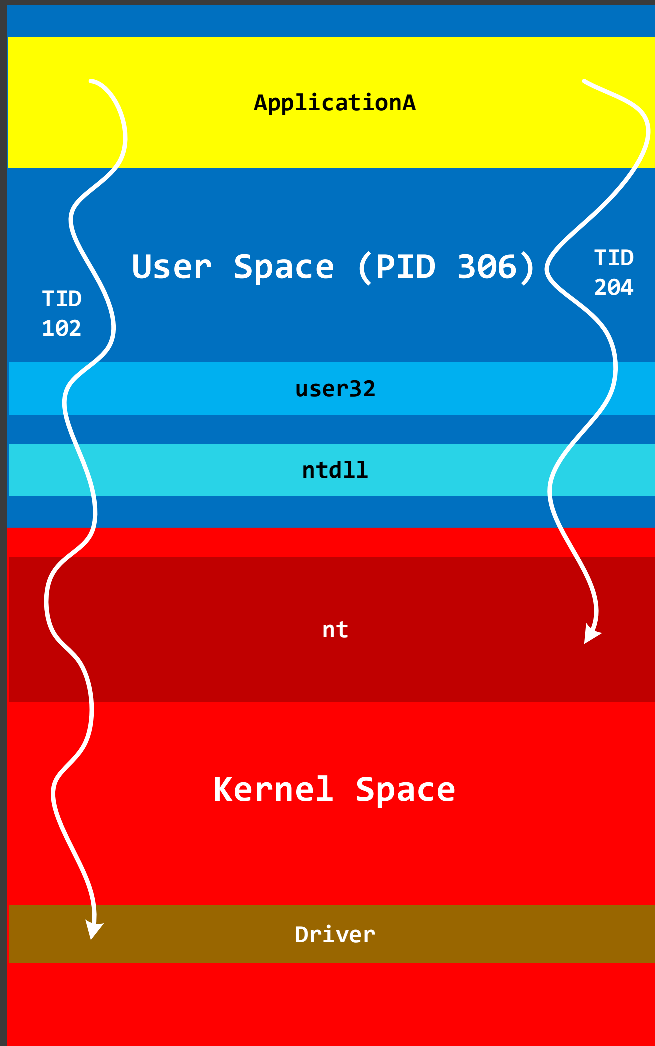


WinDbg Commands

.process switches between process virtual spaces (kernel space part remains the same)

© 2024 Software Diagnostics Services

Process Threads

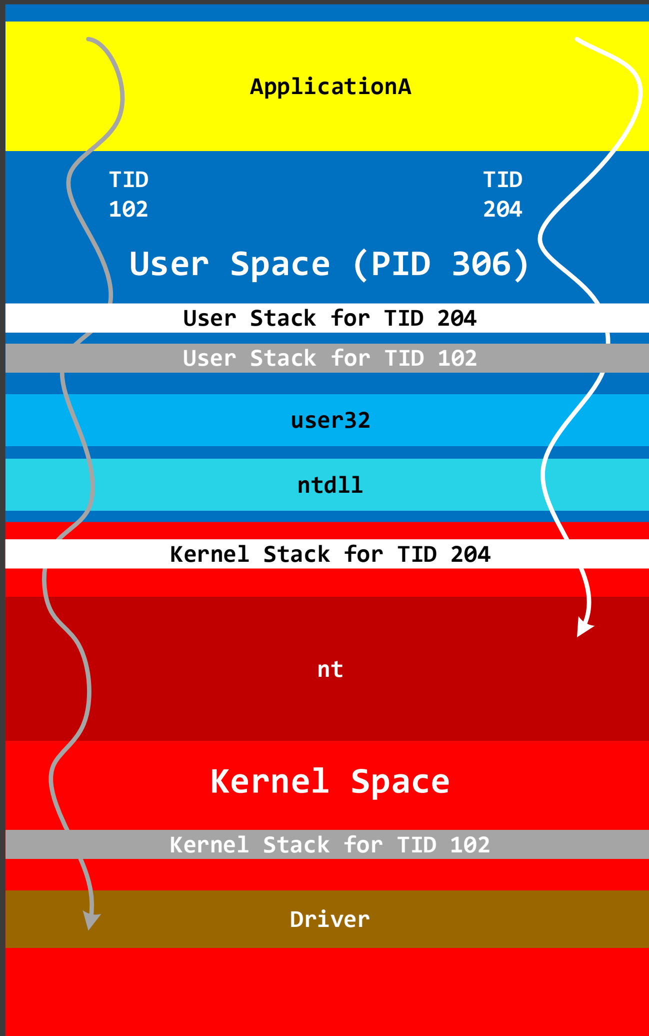


WinDbg Commands

Process dumps:
~<n>s switches between threads

© 2024 Software Diagnostics Services

Thread Stack Raw Data



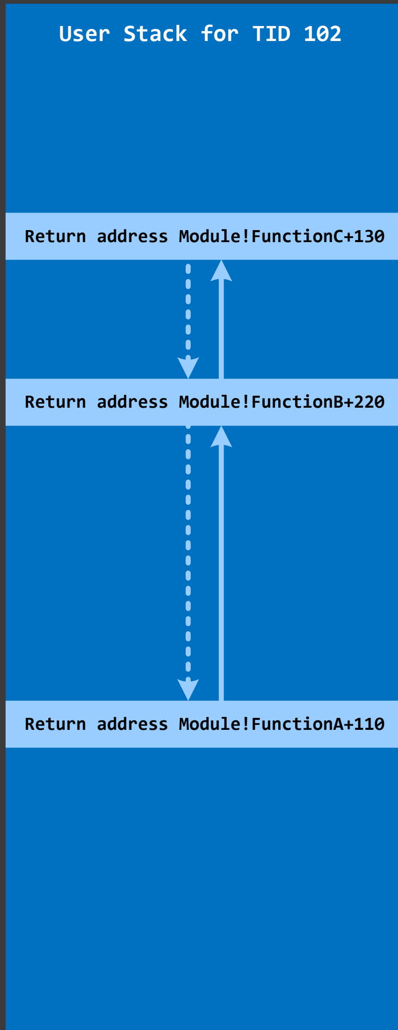
WinDbg Commands

Process dumps:
!teb

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

© 2024 Software Diagnostics Services

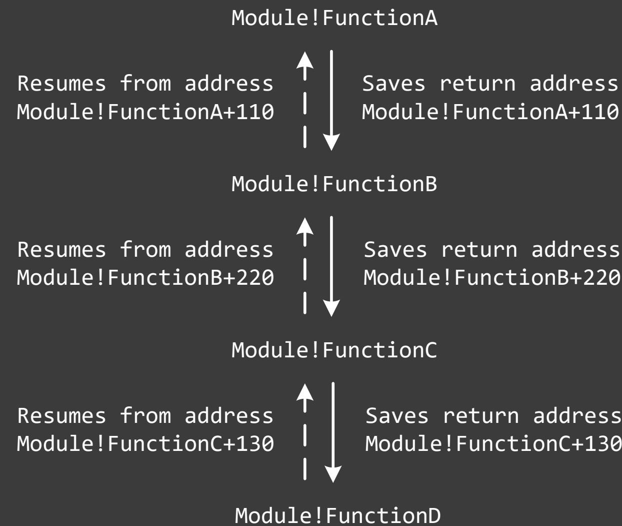
Thread Stack Trace




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

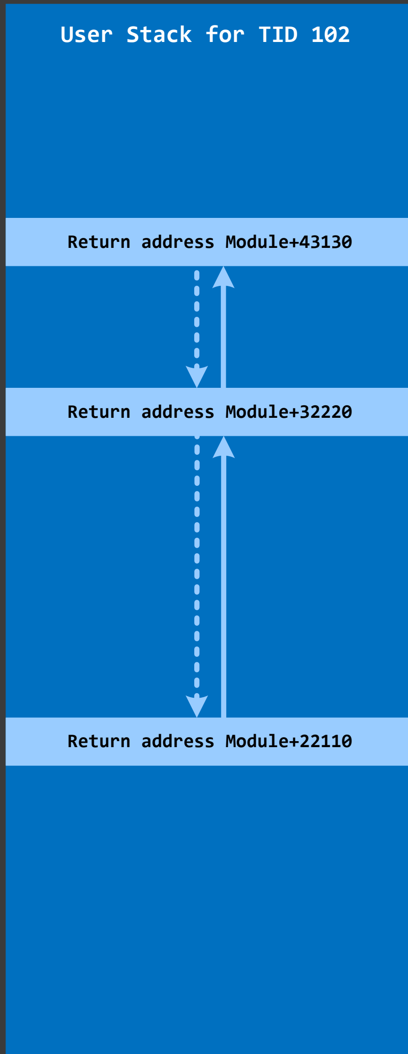
WinDbg Commands

```
0:000> k
Module!FunctionD
Module!FunctionC+130
Module!FunctionB+220
Module!FunctionA+110
```



© 2024 Software Diagnostics Services

Thread Stack Trace (no PDB)



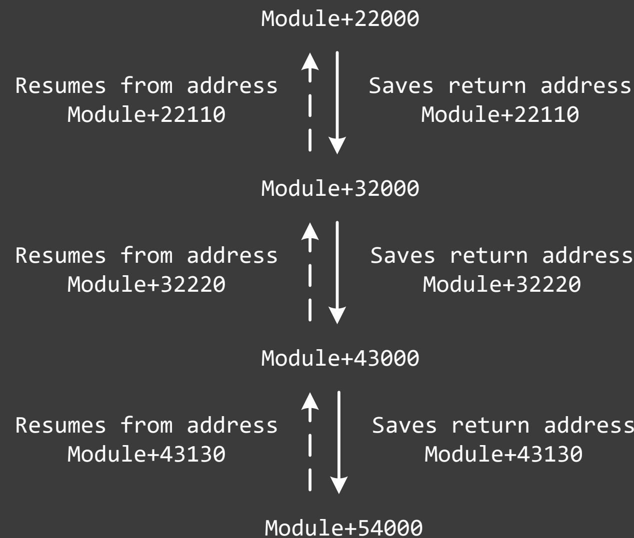

```
FunctionA()
{
  ...
  FunctionB();
  ...
}
FunctionB()
{
  ...
  FunctionC();
  ...
}

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

Symbol file Module.pdb

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

No symbols for Module

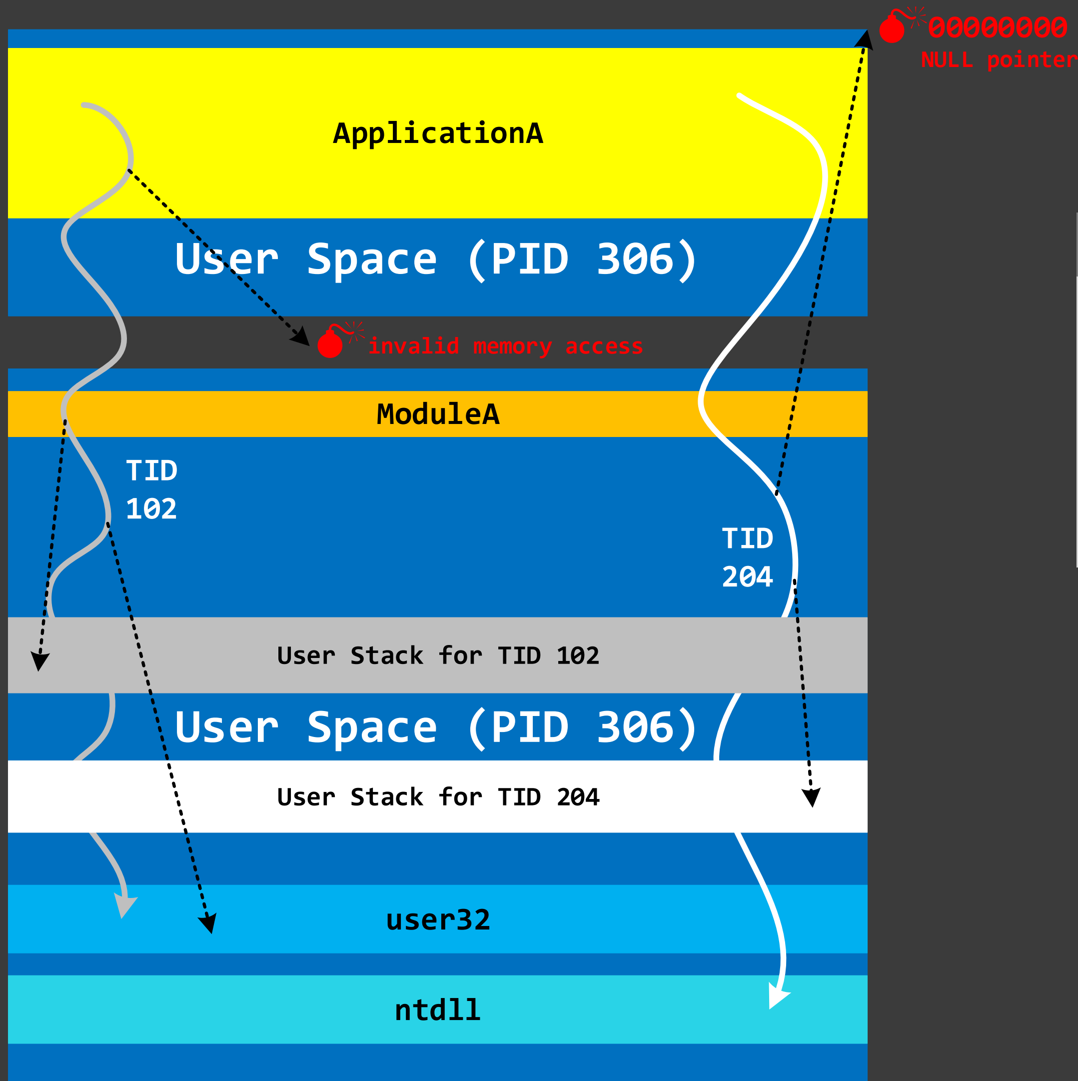


WinDbg Commands

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

© 2024 Software Diagnostics Services

Exceptions (Access Violation)



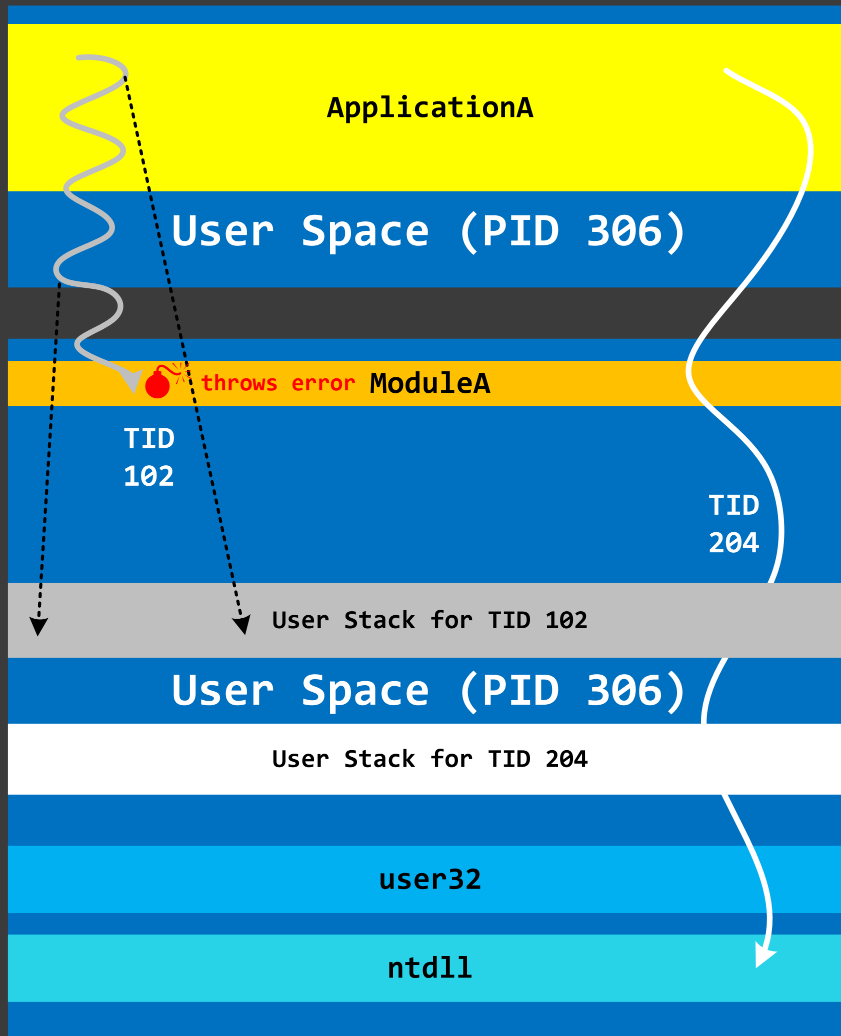
WinDbg Commands

address=????????

Set exception context (process dump):
.cxr

© 2024 Software Diagnostics Services

Exceptions (Runtime)



© 2024 Software Diagnostics Services

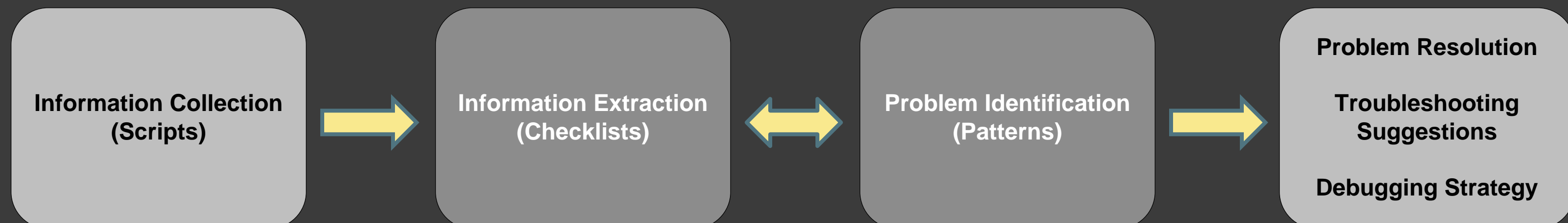
Pattern-Oriented Diagnostic Analysis

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

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

© 2024 Software Diagnostics Services

Memory Dump Collection

© 2024 Software Diagnostics Services

Process Dump Generation

- Crash or Hang, … ?

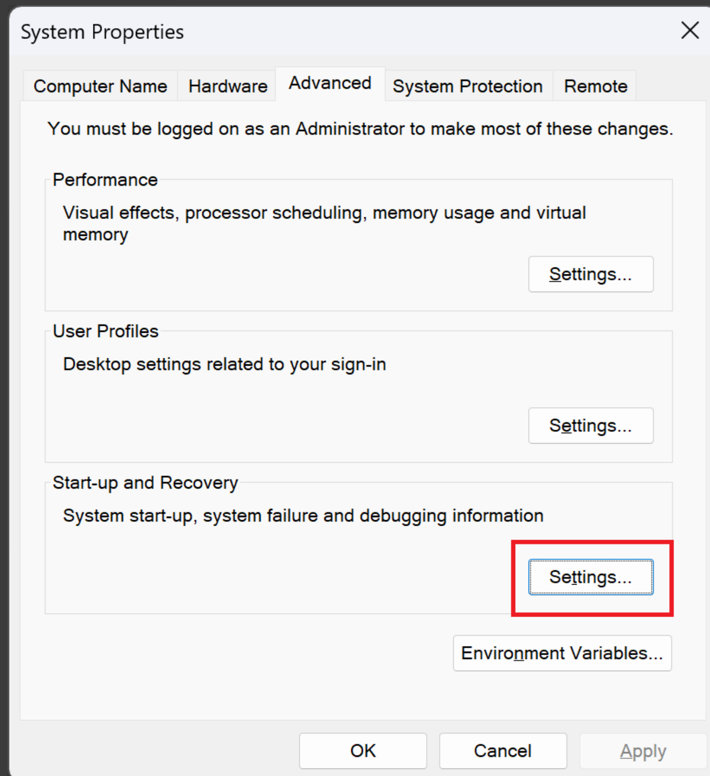
  PID in Task Manager

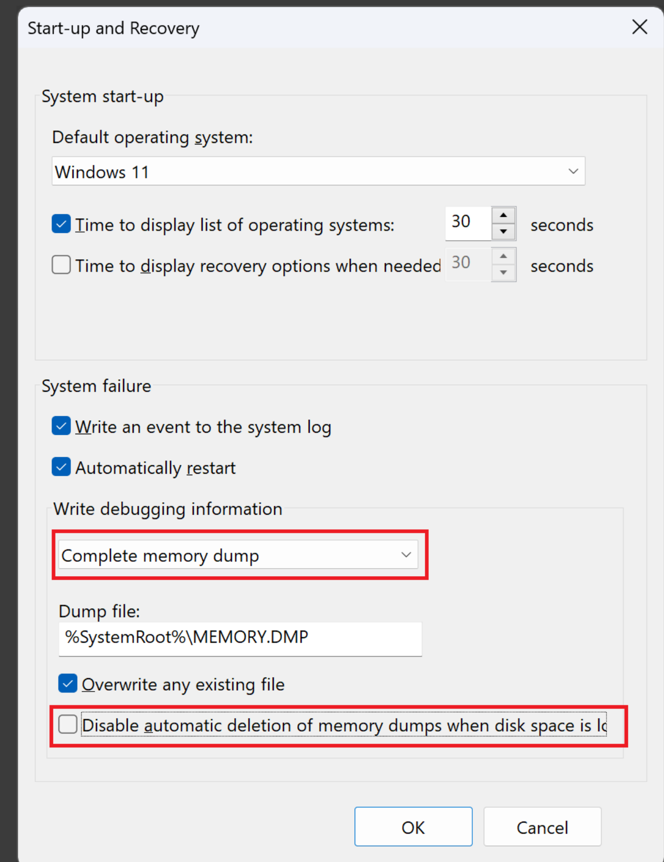
- Windows W10, W11
  - Crash: LocalDumps
  - Hang / Leak / Spike: Task Manager, procdump -ma

© 2024 Software Diagnostics Services

Complete Dump Setup

- View Advanced System Settings (Control Panel)
- Page file size > physical memory + 100 MB





© 2024 Software Diagnostics Services

Complete Dump Generation

- Keyboard (KB972110, Step 6), NMI button
- Tools: NotMyFault
- VMware memory snapshot + vmss2core

© 2024 Software Diagnostics Services

Common Issues

- Truncated complete dumps

- No “Complete memory dump” option

  HKLM \ SYSTEM \ CurrentControlSet \ Control \ CrashControl
  CrashDumpEnabled = 1 (DWORD)

© 2024 Software Diagnostics Services

x64 Disassembly

© 2024 Software Diagnostics Services

x64 CPU Registers

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

RAX 64-bit	EAX 32-bit

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

© 2024 Software Diagnostics Services

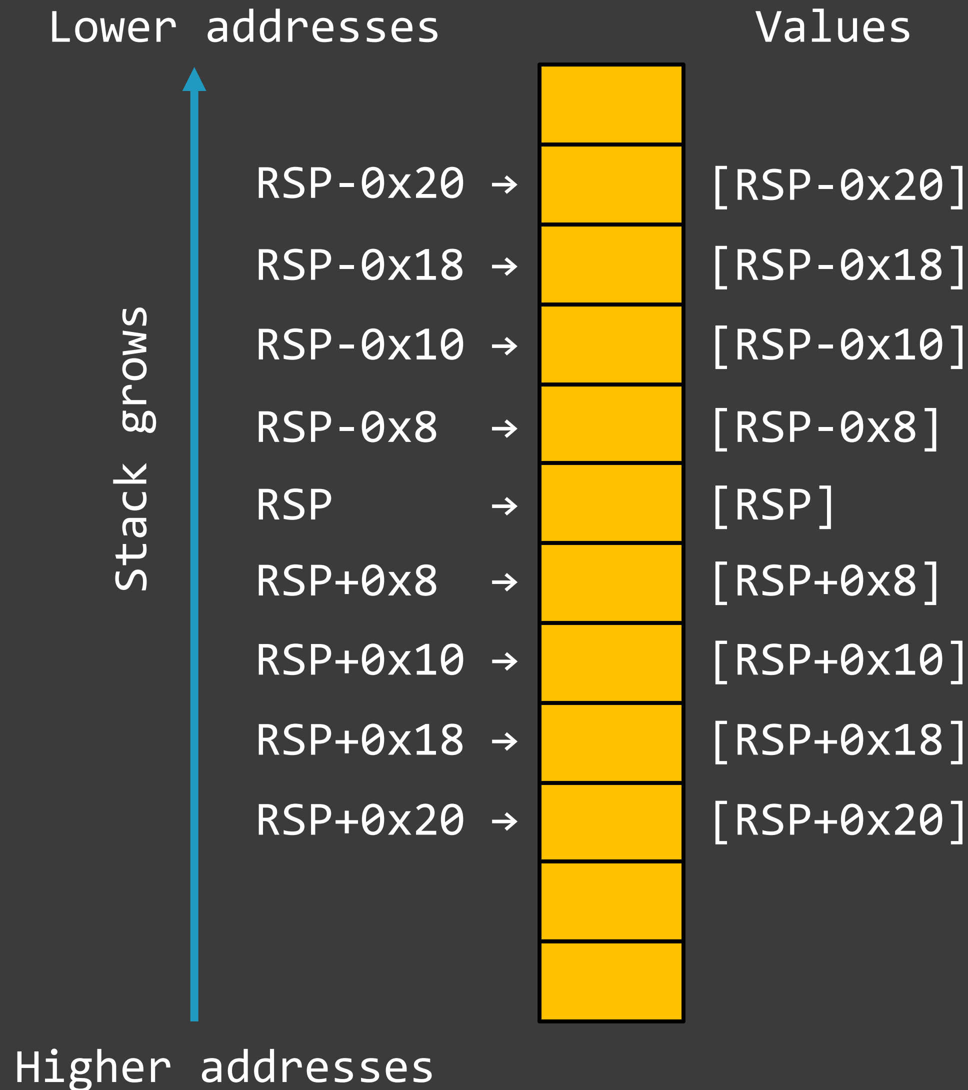
Instructions and Registers

- `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; RIP ← &func
sub   rsp, 30h               ; RSP ← RSP-0x30
                             ; make room for local variables
```

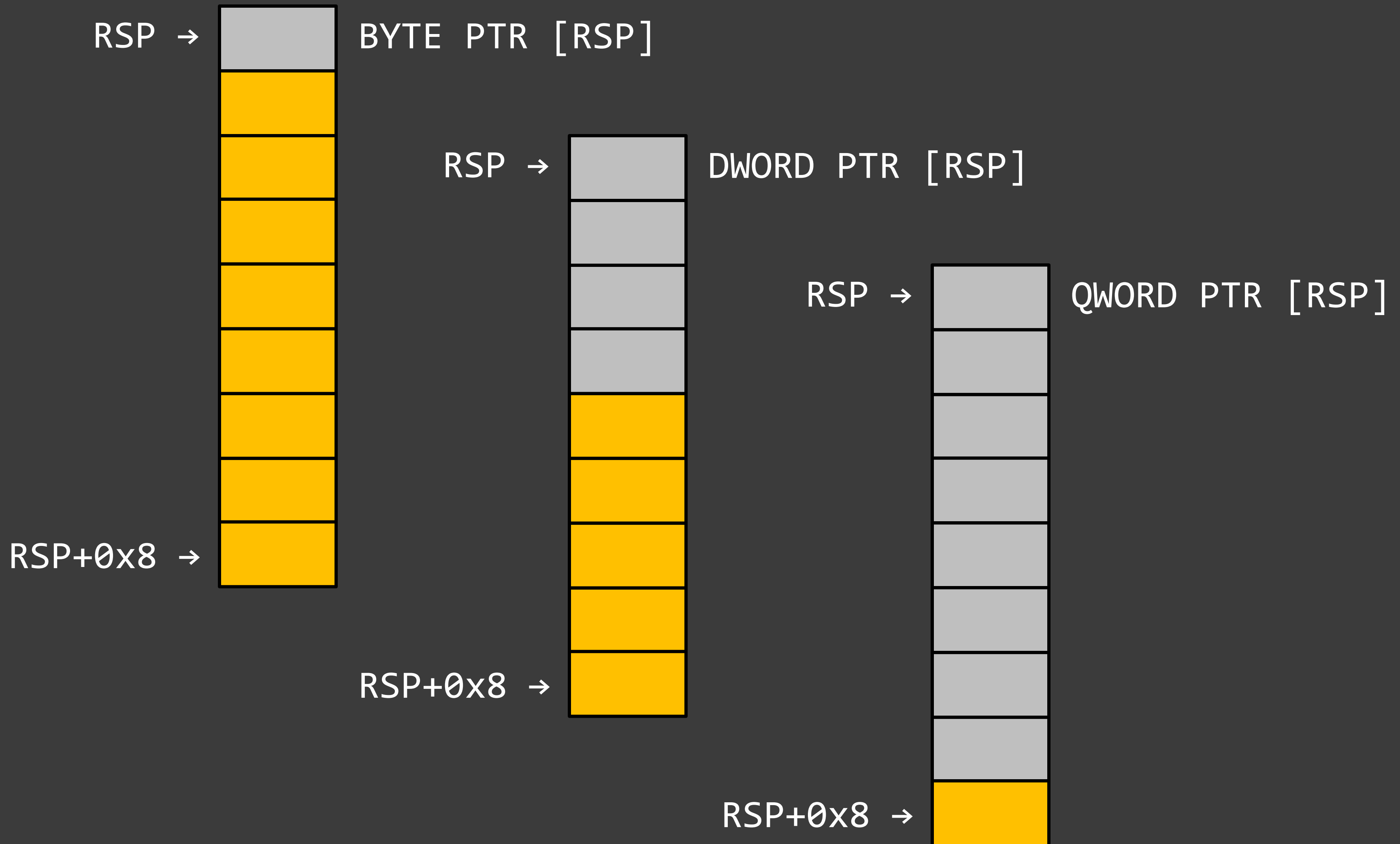
© 2024 Software Diagnostics Services

Memory and Stack Addressing



© 2024 Software Diagnostics Services

Memory Cell Sizes



© 2024 Software Diagnostics Services

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
```

© 2024 Software Diagnostics Services

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
```

© 2024 Software Diagnostics Services

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)
```

© 2024 Software Diagnostics Services

Function Parameters

- `fn func(...);`
- Left to right via `RCX, RDX, R8, R9,` stack
- stack: `[RSP+20], [RSP+28], [RSP+30], ...`

© 2024 Software Diagnostics Services

Struct Function Parameters

- `RCX`

  Implicit struct object memory address (`&myStruct`)

  ```
  let myStruct: Struct = ...;
  myStruct.func(...);
  ```

- `RDX, R8, R9, stack`

  The rest of the struct function parameters

  ```
  impl Struct {
    fn func(&self, ...);
  }
  ```

© 2024 Software Diagnostics Services

Windows API 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
…
```

© 2024 Software Diagnostics Services

Practice Exercises

© 2024 Software Diagnostics Services

Links

- Memory Dumps:

Included in Exercise RW1

- Exercise Transcripts:

Included in this book

© 2024 Software Diagnostics Services

Process Memory Dumps

Exercises RW1 – RW5

© 2024 Software Diagnostics Services

Exercise RW1

- **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); Incorrect Stack Trace; Stack Trace; Stack Trace Collection (Unmanaged Space); Main Thread; System Call; Not My Version (Software); Environment Hint; Unknown Component

- \ARWMDA-Dumps\Exercise-RW1.pdf

© 2024 Software Diagnostics Services

Supportability Best Practice

Keep PDB files from each release

© 2024 Software Diagnostics Services

Exercise RW2

- **Goal:** Learn how to analyze stack traces from debug versions
- **Patterns:** Technology-Specific Subtrace (Rust)
- \ARWMDA-Dumps\Exercise-RW2.pdf

© 2024 Software Diagnostics Services

Exercise RW3

- **Goal:** Learn how to analyze stack traces from release versions
- **Patterns:** Adjoint Stack Trace; Hidden Frame; Inline Function Optimization (Unmanaged Code)
- \ARWMDA-Dumps\Exercise-RW3.pdf

© 2024 Software Diagnostics Services