Linux
Core Dump Analysis
Accelerated

Fourth Edition

Dmitry Vostokov
Software Diagnostics Services

Prerequisites

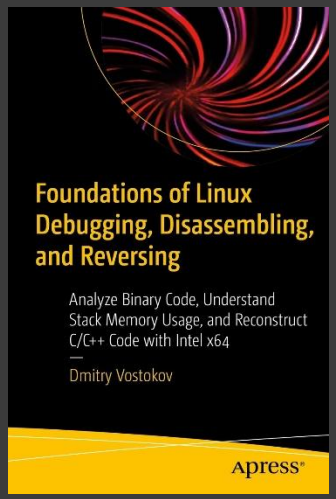
GDB Commands

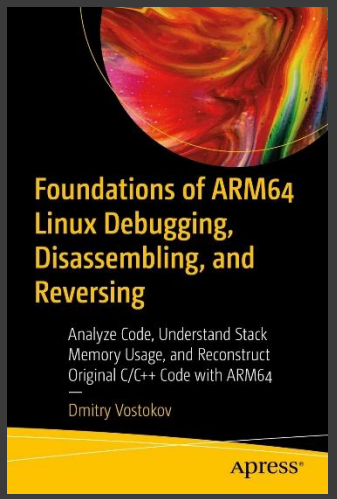
We use these boxes to introduce GDB commands used in practice exercises

WinDbg Commands

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

- Basic Linux troubleshooting
- Basics of assembly language (optional)





© 2025 Software Diagnostics Services

Training Goals

- Review fundamentals
- Learn how to collect core dumps
- Learn how to analyze core dumps

© 2025 Software Diagnostics Services

Training Principles

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

© 2025 Software Diagnostics Services

Schedule Overview

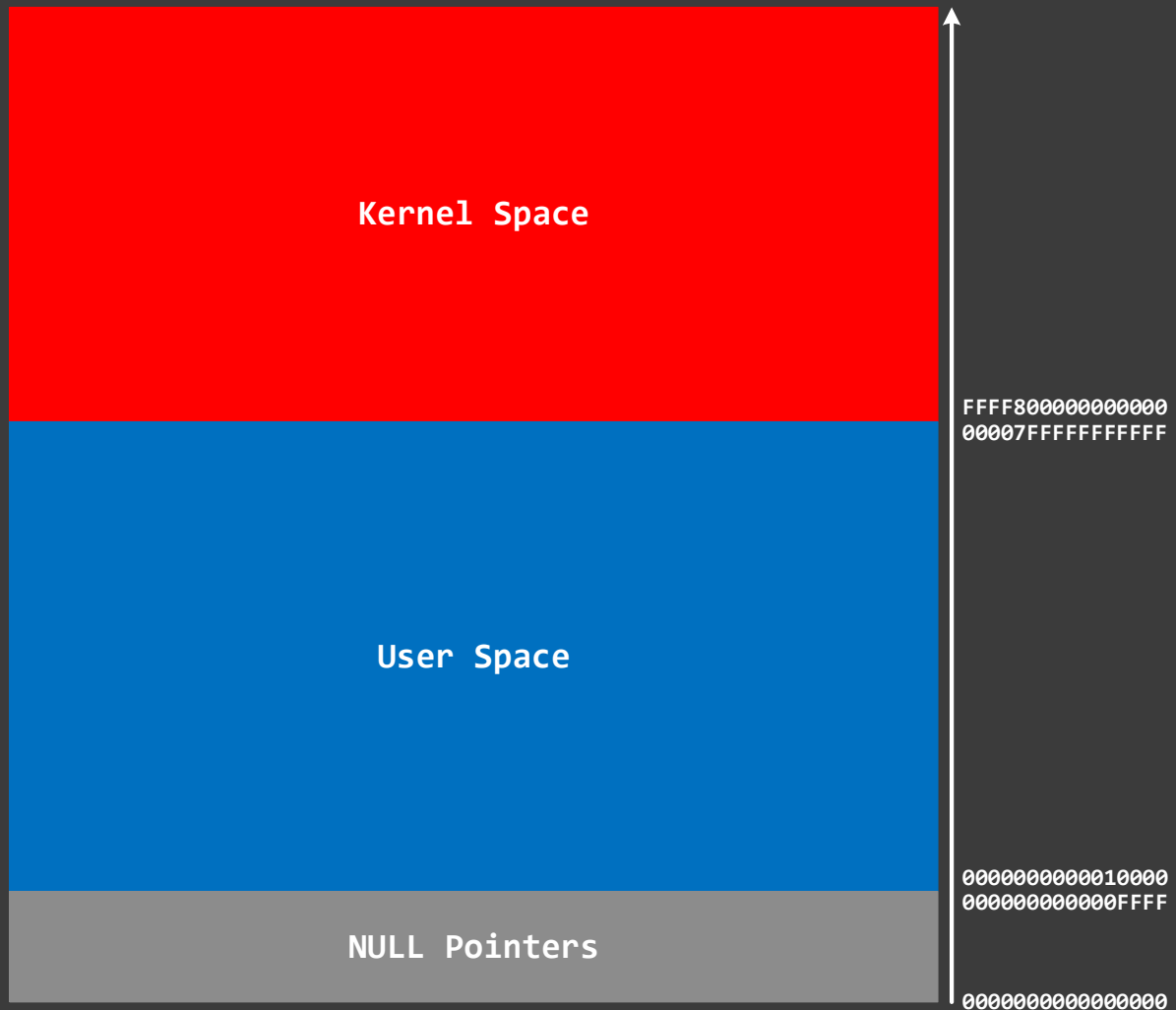
- Analysis fundamentals (x64 and ARM64)
- Process core dump collection
- Basic x64 assembly language review (GDB)
- Basic ARM64 assembly language review
- Process GDB core dump analysis (x64 and ARM64)
- Basic x64 assembly language review (WinDbg)
- Process WinDbg core dump analysis (x64 and ARM64)
- Kernel core dump collection
- Kernel core dump analysis (Crash tool, x64 and ARM64)
- Kernel core dump analysis (WinDbg)
- New analysis patterns
- Mechanisms

© 2025 Software Diagnostics Services

Part 1: Fundamentals

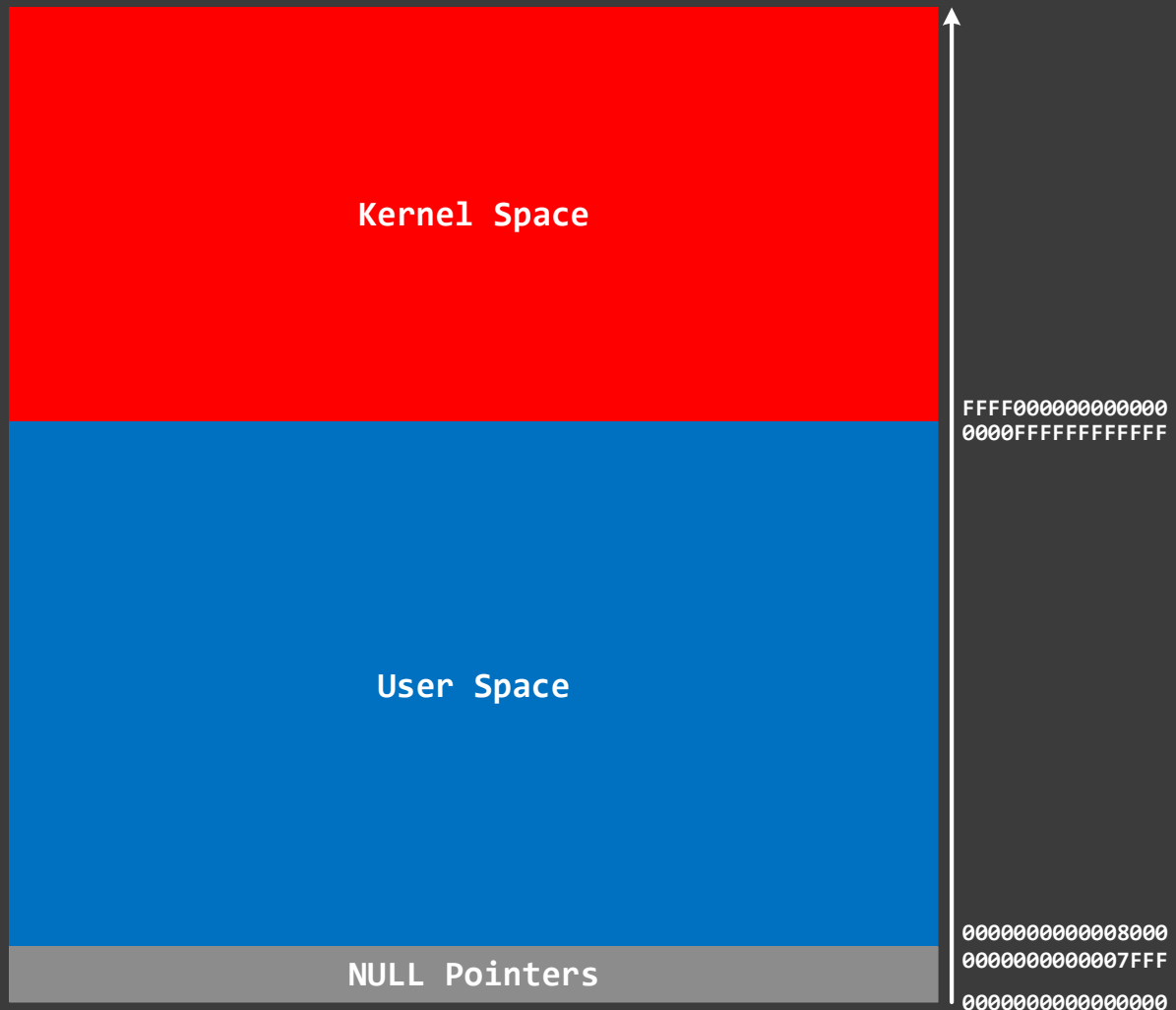
© 2025 Software Diagnostics Services

Memory/Kernel/User Space (x64)



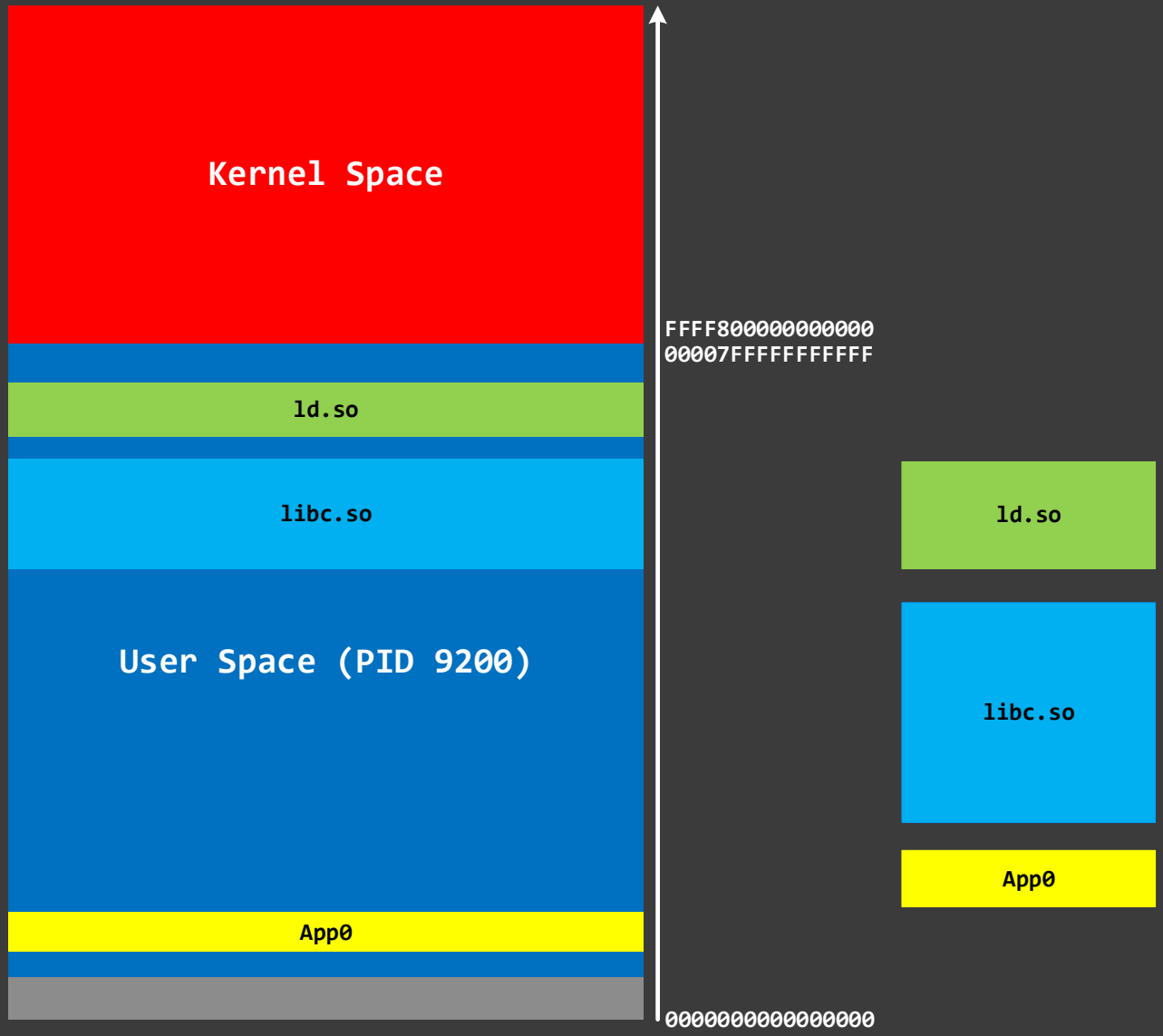
© 2025 Software Diagnostics Services

Memory/Kernel/User Space (ARM64)



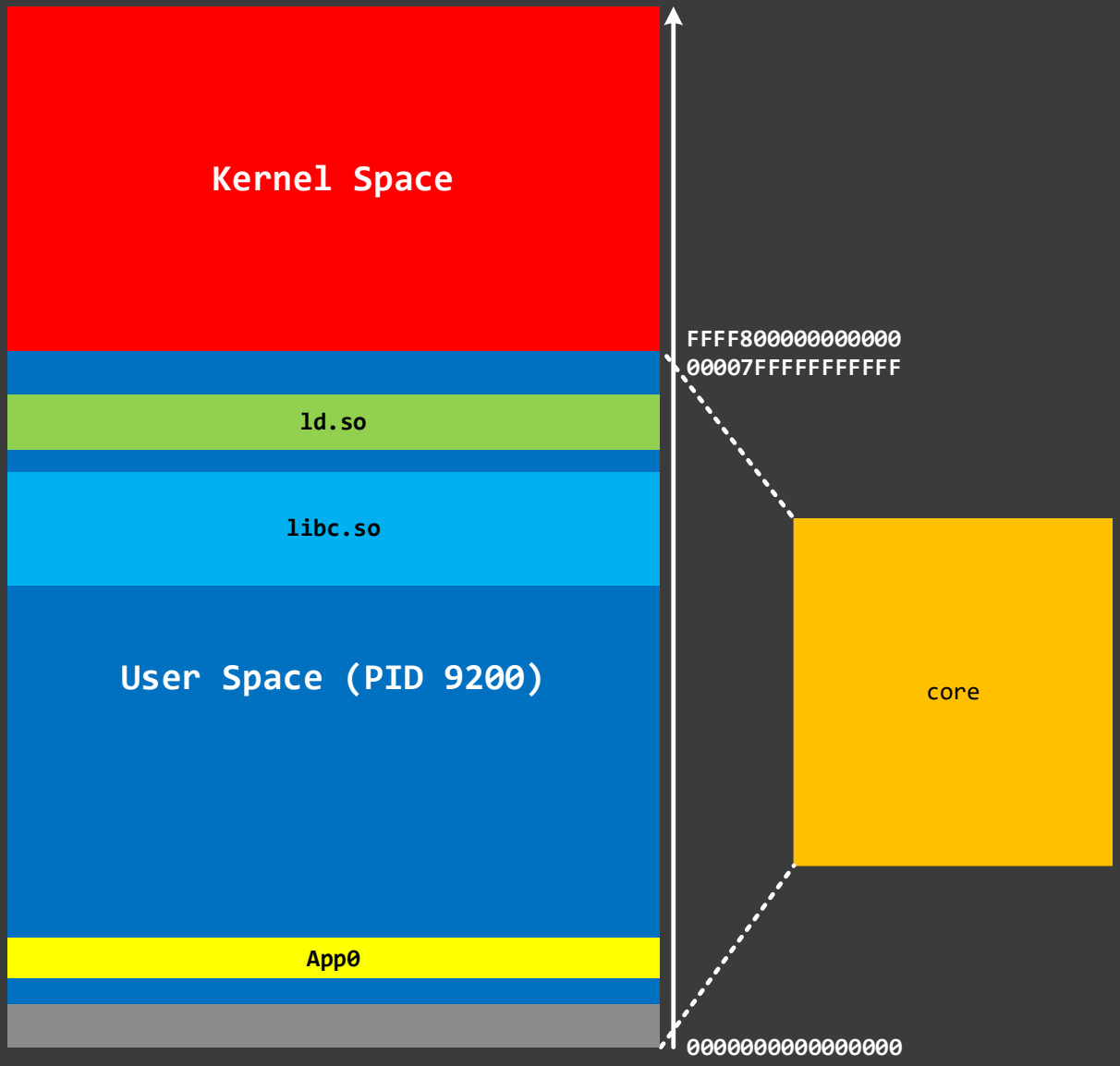
© 2025 Software Diagnostics Services

App/Process/Library (x64)



© 2025 Software Diagnostics Services

Process Memory Dump (x64)



GDB Commands

`info sharedlibrary`
Lists dynamic libraries

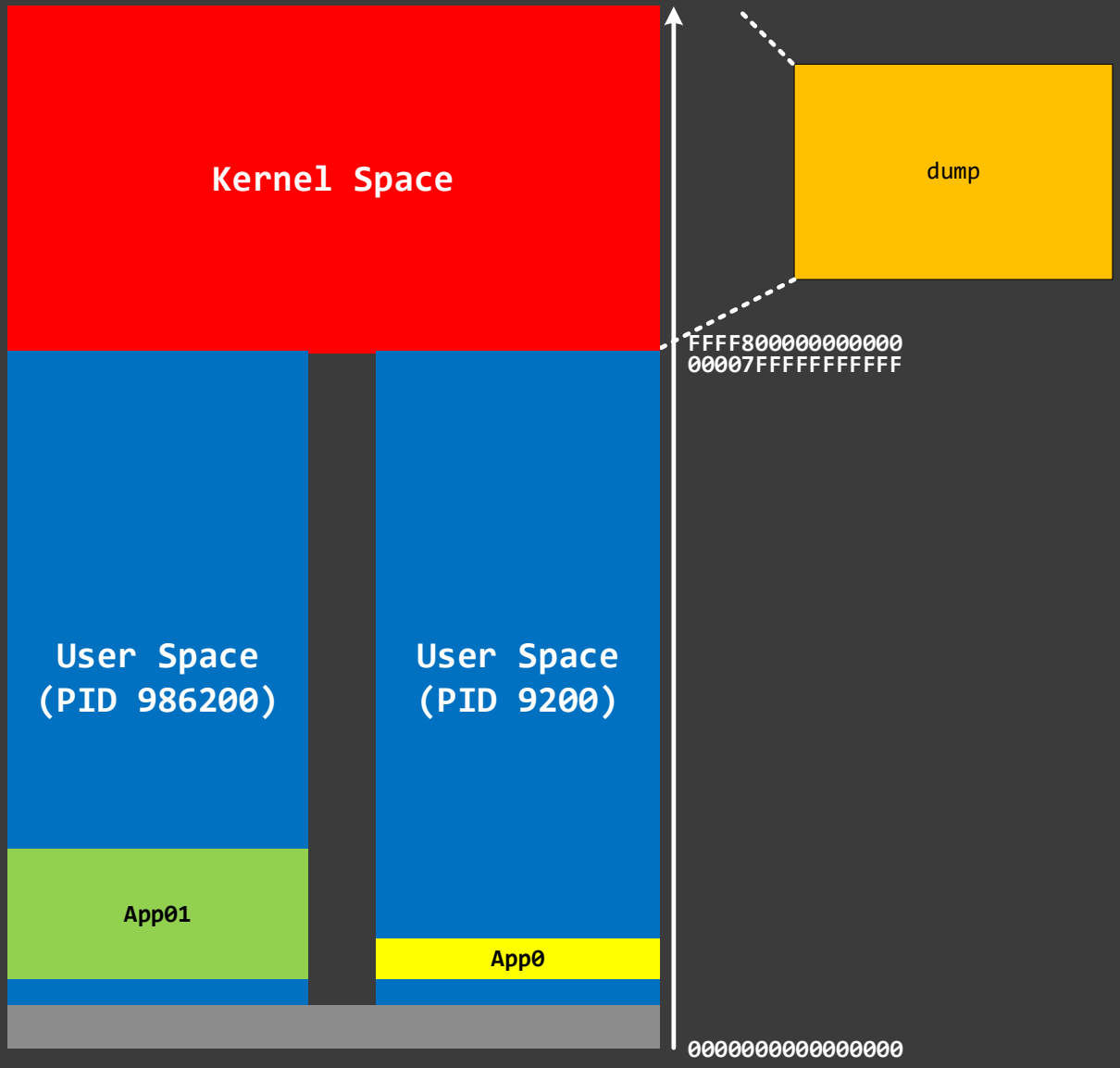
`maintenance info sections`
Lists memory regions

WinDbg Commands

`!address`
Lists memory regions

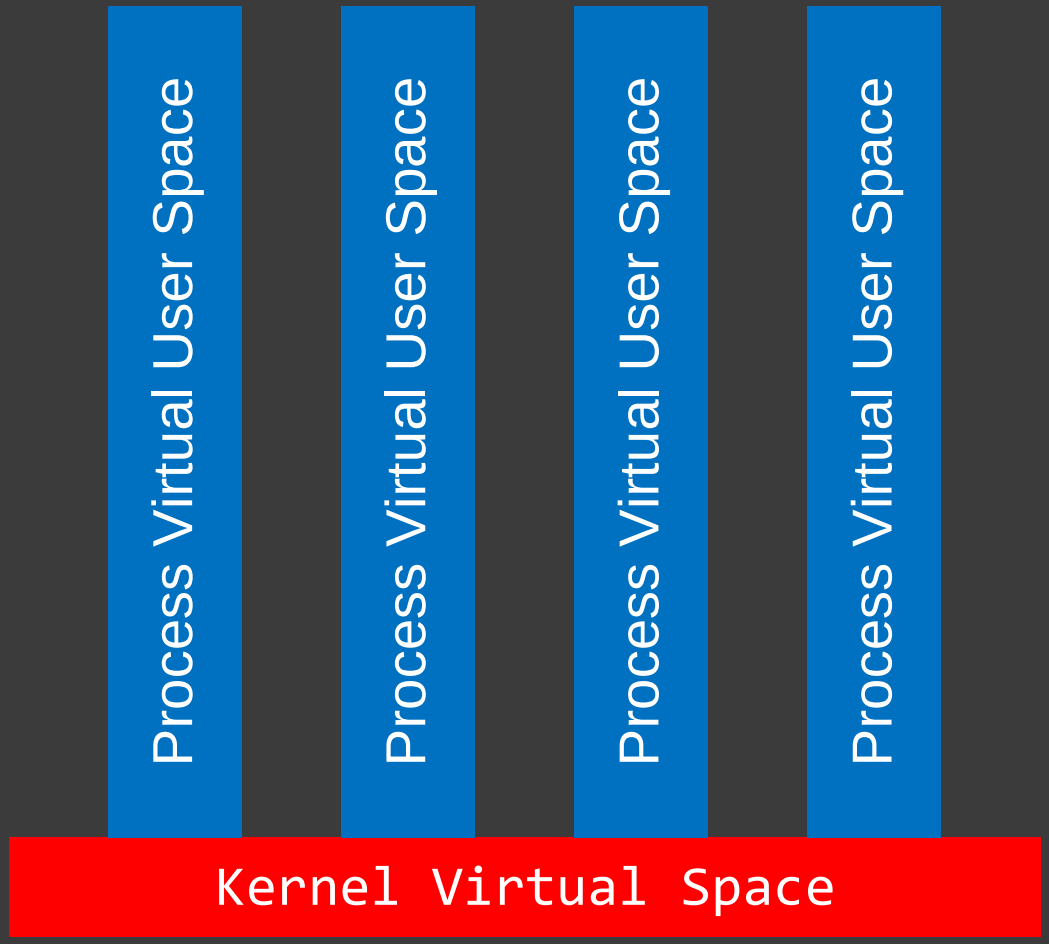
© 2025 Software Diagnostics Services

Kernel Memory Dump (x64)



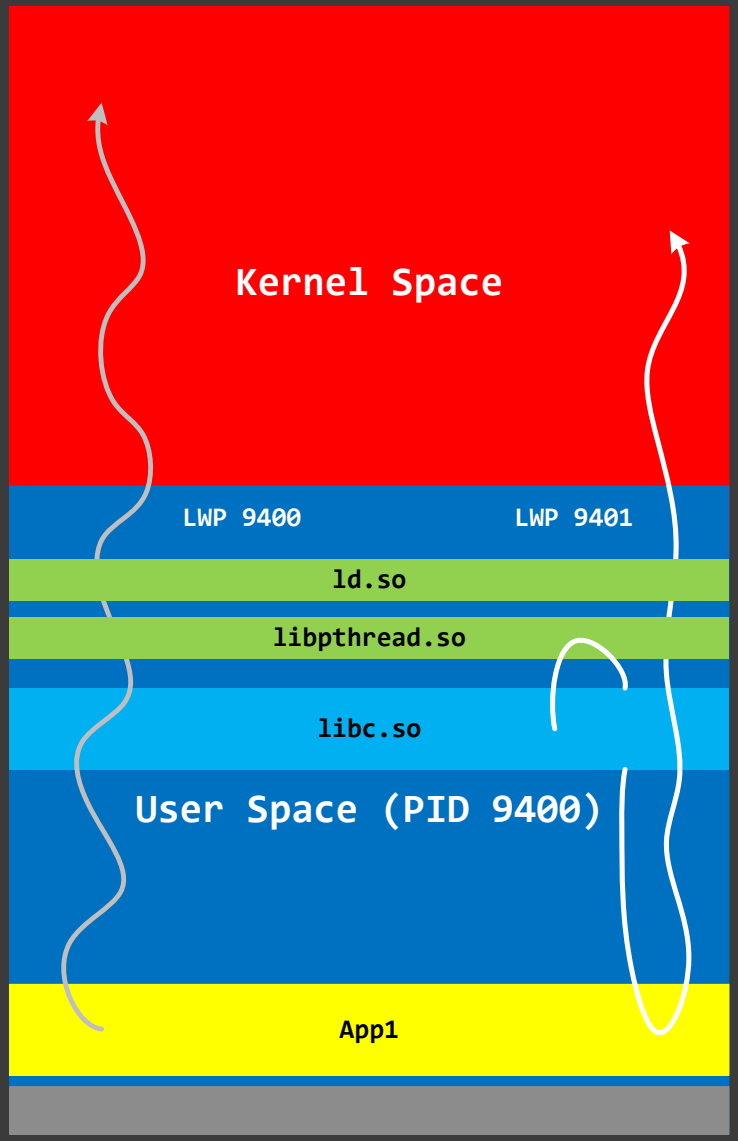
© 2025 Software Diagnostics Services

Fiber Bundle Memory Dump



© 2025 Software Diagnostics Services

Lightweight Processes (Threads)



GDB Commands

`info threads`
Lists threads

`thread <n>`
Switches between threads

`thread apply all bt`
Lists stack traces from all threads

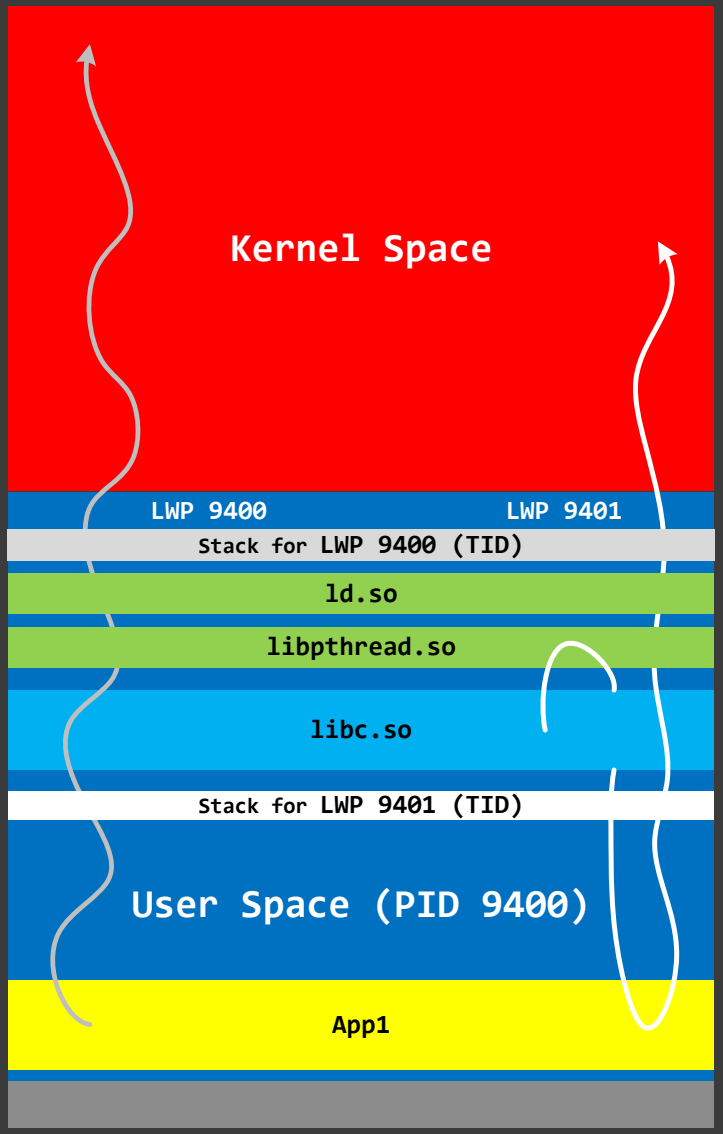
WinDbg Commands

`~*k`
Lists stack traces from all threads

`~<n>s`
Switches between threads

© 2025 Software Diagnostics Services

Thread Stack Raw Data



GDB Commands

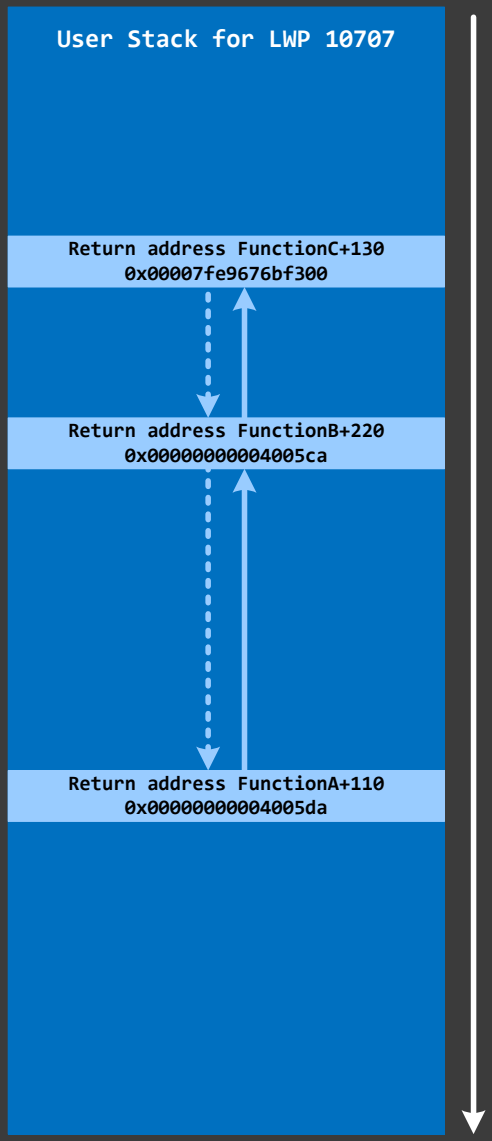
`x/<n>a <address>`
Prints n addresses with corresponding symbol mappings if any

WinDbg Commands

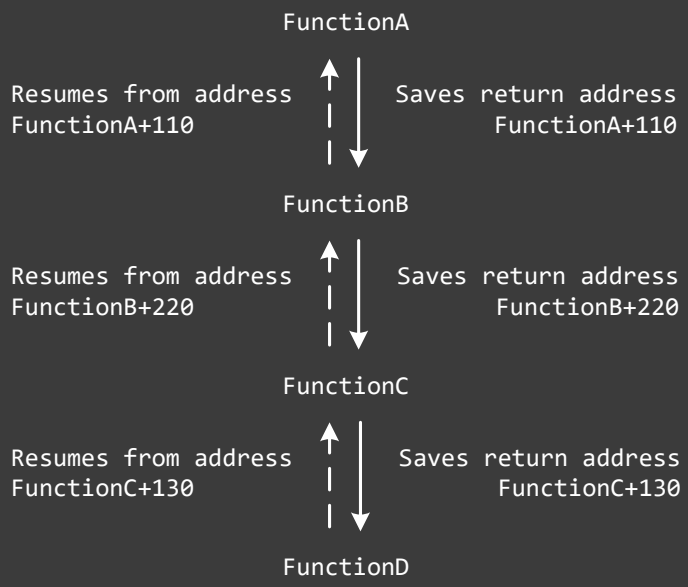
`dps <address> L<n>`
Prints n addresses with corresponding symbol mappings if any

© 2025 Software Diagnostics Services

Thread Stack Trace




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

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

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



GDB Commands

```
(gdb) bt
#0 0x00007fe9676bf48d in FunctionD ()
#1 0x00007fe9676bf300 in FunctionC ()
#2 0x00000000004005ca in FunctionB ()
#3 0x00000000004005da in FunctionA ()
```

© 2025 Software Diagnostics Services

GDB vs. WinDbg vs. LLDB

GDB Commands

```
(gdb) bt
#0 0x00007fe9676bf48d in FunctionD ()
#1 0x00007fe9676bf300 in FunctionC ()
#2 0x00000000004005ca in FunctionB ()
#3 0x00000000004005da in FunctionA ()
```

WinDbg Commands

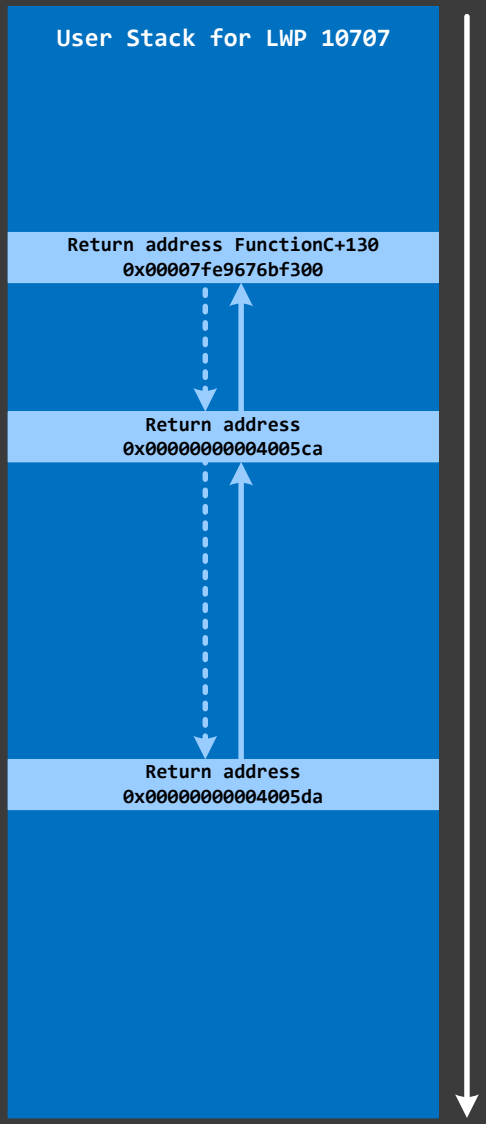
```
0:000> k
00 00007fe9676bf300 Module!FunctionD+offset
01 00000000004005ca Module!FunctionC+130
02 00000000004005da AppA!FunctionB+220
03 0000000000000000 AppA!FunctionA+110
```

LLDB Commands

```
(lldb) bt
frame #0: 0x000000020328982a Module`FunctionD + offset
frame #1: 0x0000000203288a9c Module`FunctionC + 130
frame #2: 0x0000000104da3ea9 AppA`FunctionB + 220
frame #3: 0x0000000104da3edb AppA`FunctionA + 110
```

© 2025 Software Diagnostics Services

Thread Stack Trace (no symbols)



Symbol file App.sym

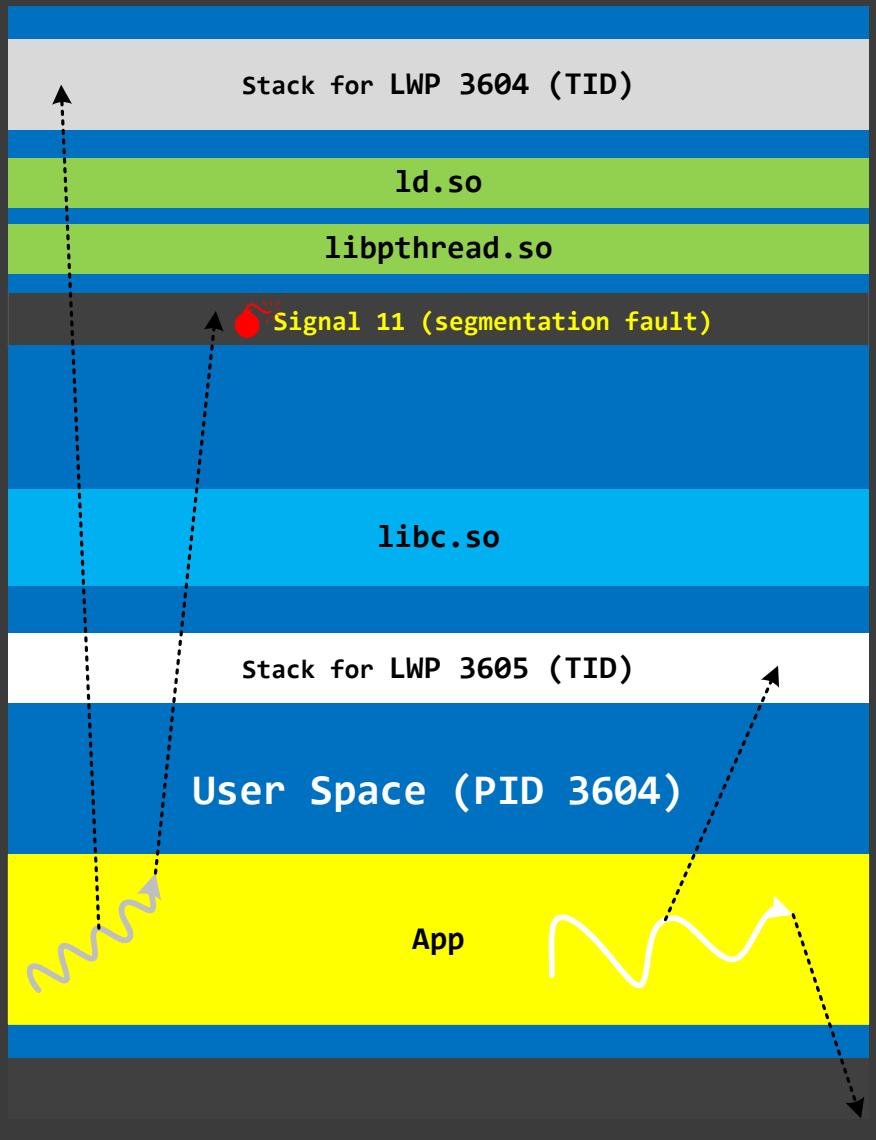
FunctionA 22000 - 23000
FunctionB 32000 - 33000

GDB Commands

```
(gdb) bt
#0 0x00007fe9676bf48d in FunctionD ()
#1 0x00007fe9676bf300 in FunctionC ()
#2 0x00000000004005ca in ?? ()
#3 0x00000000004005da in ?? ()
```

© 2025 Software Diagnostics Services

Exceptions (Access Violation)



GDB Commands

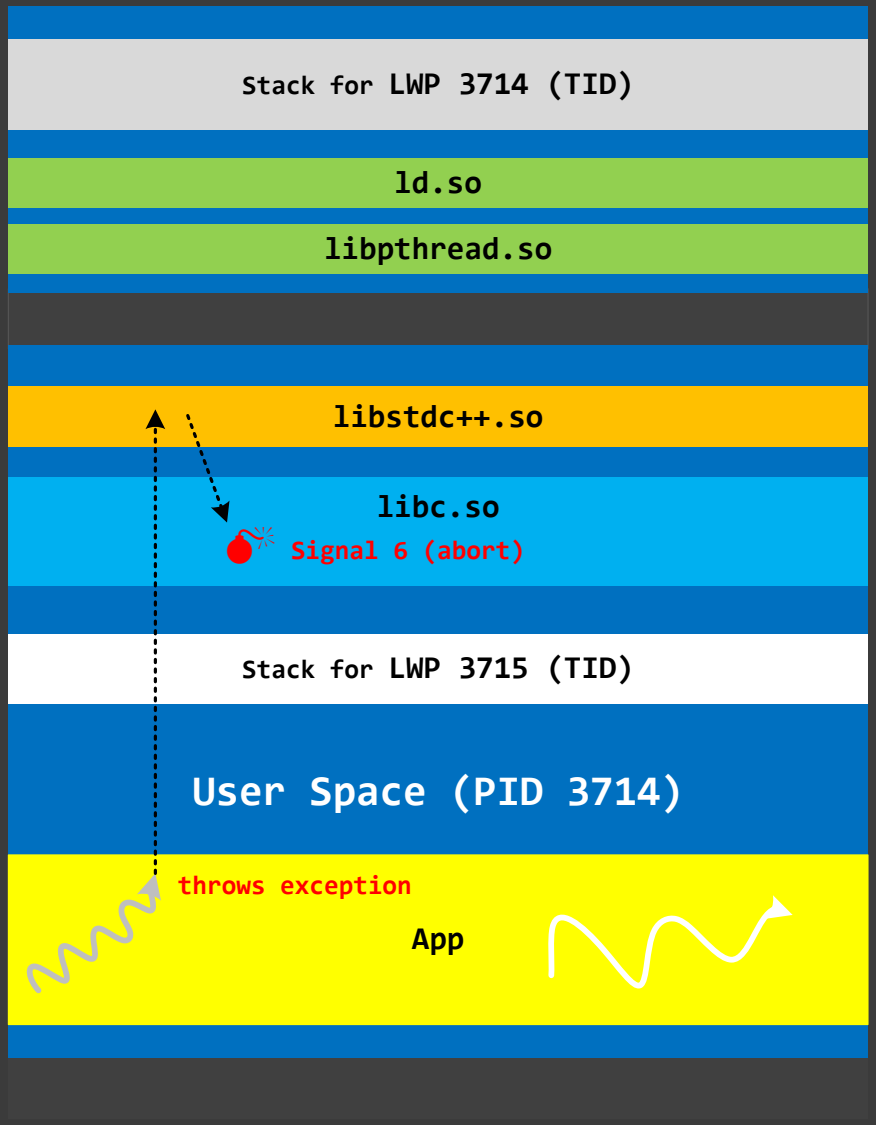
```
(gdb) x <address>
0x<address>: Cannot access
memory at address 0x<address>
```

WinDbg Commands

```
0:000> dp <address> L1
<address>  ????????`????????
```

© 2025 Software Diagnostics Services

Exceptions (Runtime)



© 2025 Software Diagnostics Services

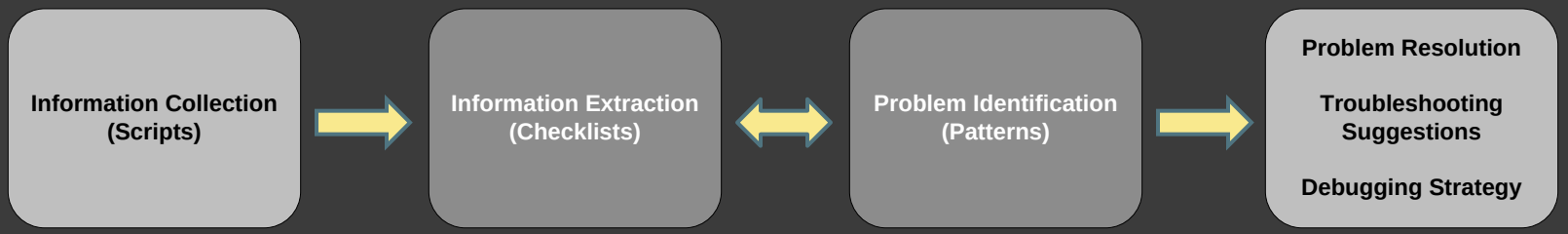
Pattern-Oriented Diagnostic Analysis

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, macOS, Linux, ...



© 2025 Software Diagnostics Services

Part 2: Core Dump Collection

© 2025 Software Diagnostics Services

Enabling Collection (Processes)

- Temporary for the current user

```
$ ulimit -c unlimited
```

- Permanent for every user except root

Edit the file: `/etc/security/limits.conf`

Add or uncomment the line:

```
*      soft   core   unlimited
```

To limit root to 1GB, add or uncomment this line:

```
*      hard   core   1000000
```

© 2025 Software Diagnostics Services

Fine-tuning Collection (Processes)

- Include shared libraries code

```
$ echo 0x3F > /proc/<pid>/coredump_filter
```

- You still need symbol files (GDB)
- You may not need symbol files (WinDbg)

- Further information

https://man7.org/linux/man-pages/man5/core.5.html

© 2025 Software Diagnostics Services

Generation Methods (Processes)

- kill (requires ulimit)

```
$ kill -s SIGQUIT PID
$ kill -s SIGABRT PID
```

- gcore

```
$ gcore [-o filename] PID
```

- procdump

https://github.com/Sysinternals/ProcDump-for-Linux

© 2025 Software Diagnostics Services

Finding Core Dumps (Processes)

- Check the current core dump directory and naming pattern

```
$ cat /proc/sys/kernel/core_pattern
```

- Search

```
$ sudo find / -name core.*
```

- Further information

https://man7.org/linux/man-pages/man5/core.5.html

© 2025 Software Diagnostics Services

Enabling Collection (Kernel)

- Uncompressed kernel image with symbols:

Debian: `$ sudo apt install linux-image-$(uname -r)-dbg`

Ubuntu: https://wiki.ubuntu.com/Kernel/Systemtap (Where to get debug symbols for kernel X?)

- Kdump (and kexec):

```
$ sudo apt install kdump-tools kexec-tools
```

```
$ kdump-config show
```

© 2025 Software Diagnostics Services

Configuring Collection (Kernel)

- General:

```
$ sudo nano /etc/default/kdump-tools
```

```
$ man makedumpfile
```

- Preprocessing (for WinDbg):

```
makedumpfile –c ... (ZLIB compression)
```

```
$ sudo makedumpfile -R date.kdump < dump.date
```

© 2025 Software Diagnostics Services

Generation Methods (Kernel)

- Manual (you may need to be root, not sudo)

```
$ sudo passwd root
$ su root
# echo 1 > /proc/sys/kernel/sysrq
# echo c > /proc/sysrq-trigger
```

- Kernel modules

© 2025 Software Diagnostics Services

Finding Core Dumps (Kernel)

- Core dumps

`/var/crash`

- `vmlinux`

`/usr/lib/debug`

© 2025 Software Diagnostics Services

Enabling Analysis (Kernel)

- Install crash tool (depends on distribution)

```
$ sudo apt install crash
```

- Compile crash tool from source

```
$ git clone https://github.com/crash-utility/crash.git
$ sudo apt install build-essential wget libgmp-dev
   libmpfr-dev libmpc-dev flex bison texinfo zlib1g-dev
   libncurses-dev
$ cd crash
$ make
$ sudo make install
```

© 2025 Software Diagnostics Services

Part 3: x64 Disassembly

(AT&T GDB Flavor)

© 2025 Software Diagnostics Services

CPU Registers (x64)

- **RAX** ⊃ **EAX** ⊃ **AX** ⊇ {**AH**, **AL**}
- ALU: **RAX**, **RDX**
- Counter: **RCX**
- Memory copy: **RSI** (src), **RDI** (dst)
- Stack: **RSP**, **RBP**
- Next instruction: **RIP**
- New: **R8** – **R15**, **Rx**(**D**|**W**|**L**)

GDB Commands

```
info registers
```

© 2025 Software Diagnostics Services

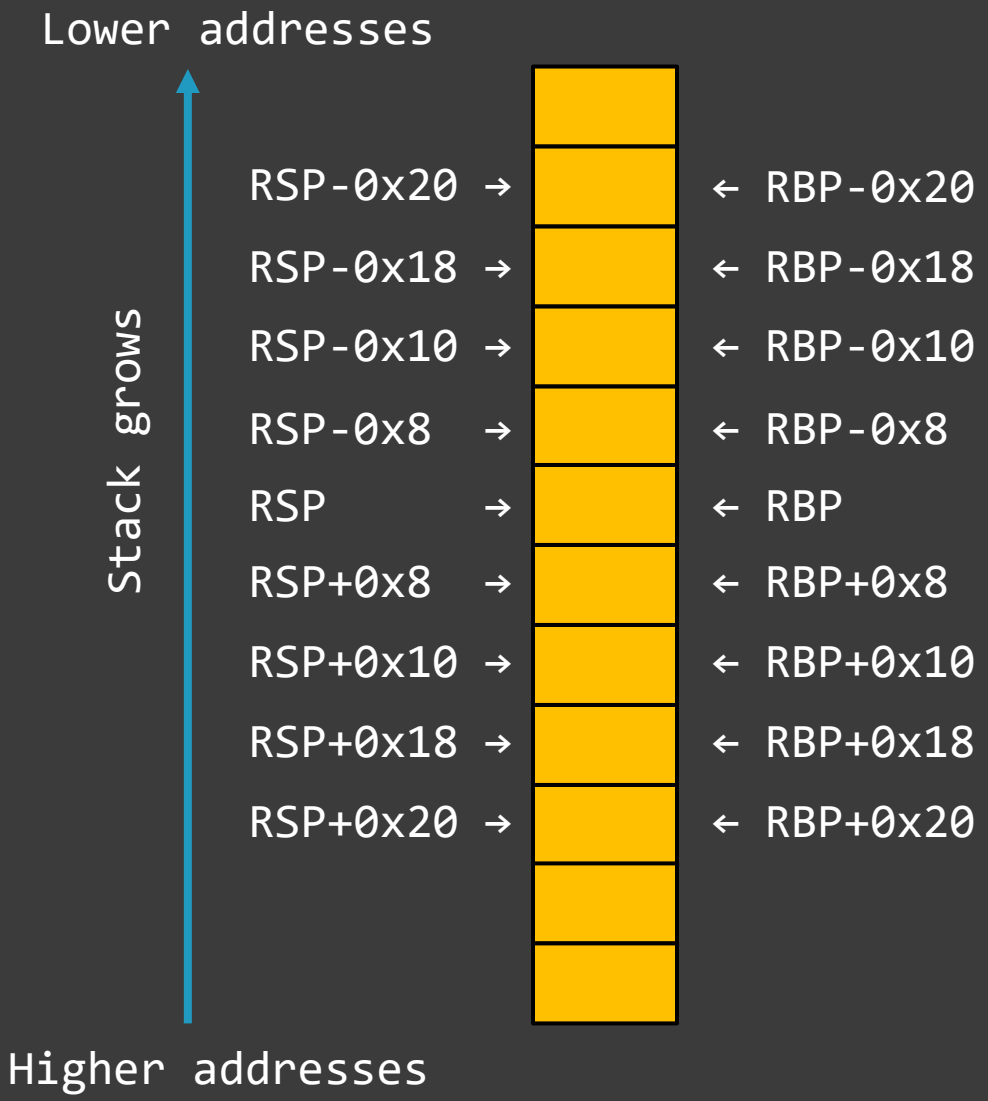
Instructions: registers (x64)

- `Opcode SRC, DST # default AT&T flavour`
- Examples:

```
mov   $0x10, %rax          # 0x10 → RAX
mov   %rsp, %rbp           # RSP → RBP
add   $0x10, %r10          # R10 + 0x10 → R10
imul  %ecx, %edx           # ECX * EDX → EDX
callq *%rdx                # RDX already contains
                           #    the address of func (&func)
                           # PUSH RIP; &func → RIP
sub   $0x30, %rsp          # RSP-0x30 → RSP
                           # make a room for local variables
```

© 2025 Software Diagnostics Services

Memory and Stack Addressing



© 2025 Software Diagnostics Services

Instructions: memory load (x64)

- `Opcode Offset(SRC), DST`
- `Opcode DST`
- Examples:

```
mov   0x10(%rsp), %rax       # value at address RSP+0x10 → RAX
mov   -0x10(%rbp), %rcx      # value at address RBP-0x10 → RCX
add   (%rax), %rdx           # RDX + value at address RAX → RDX
pop   %rdi                   # value at address RSP → RDI
                             # RSP + 8 → RSP
lea   0x20(%rbp), %r8        # address RBP+0x20 → R8
```

© 2025 Software Diagnostics Services

Instructions: memory store (x64)

- `Opcode SRC, Offset(DST)`
- `Opcode SRC|DST`
- Examples:

```
mov   %rcx, -0x20(%rbp)      # RCX → value at address RBP-0x20
addl  $1, (%rax)             # 1 + 32-bit value at address RAX →
                             #     32-bit value at address RAX
push  %rsi                   # RSP - 8 → RSP
                             # RSI → value at address RSP
inc   (%rcx)                 # 1 + value at address RCX →
                             #     value at address RCX
```

© 2025 Software Diagnostics Services

Instructions: indexing (x64)

- Opcode Offset(SRC_{base}, SRC_{index}, Scale), DST
- Opcode SRC, Offset(DST_{base}, DST_{index}, Scale)
- Examples:

```
mov   (%rdx, %rax, 1), %rax  # value at address RDX+RAX*1 → RAX

movq  $0x0, (%rdx, %rax, 8)  # 0 → value at address RDX+RAX*8

lea   0x0(, %rax, 8), %rdx   # address RAX*8+0 → RDX
```

© 2025 Software Diagnostics Services

Instructions: flow (x64)

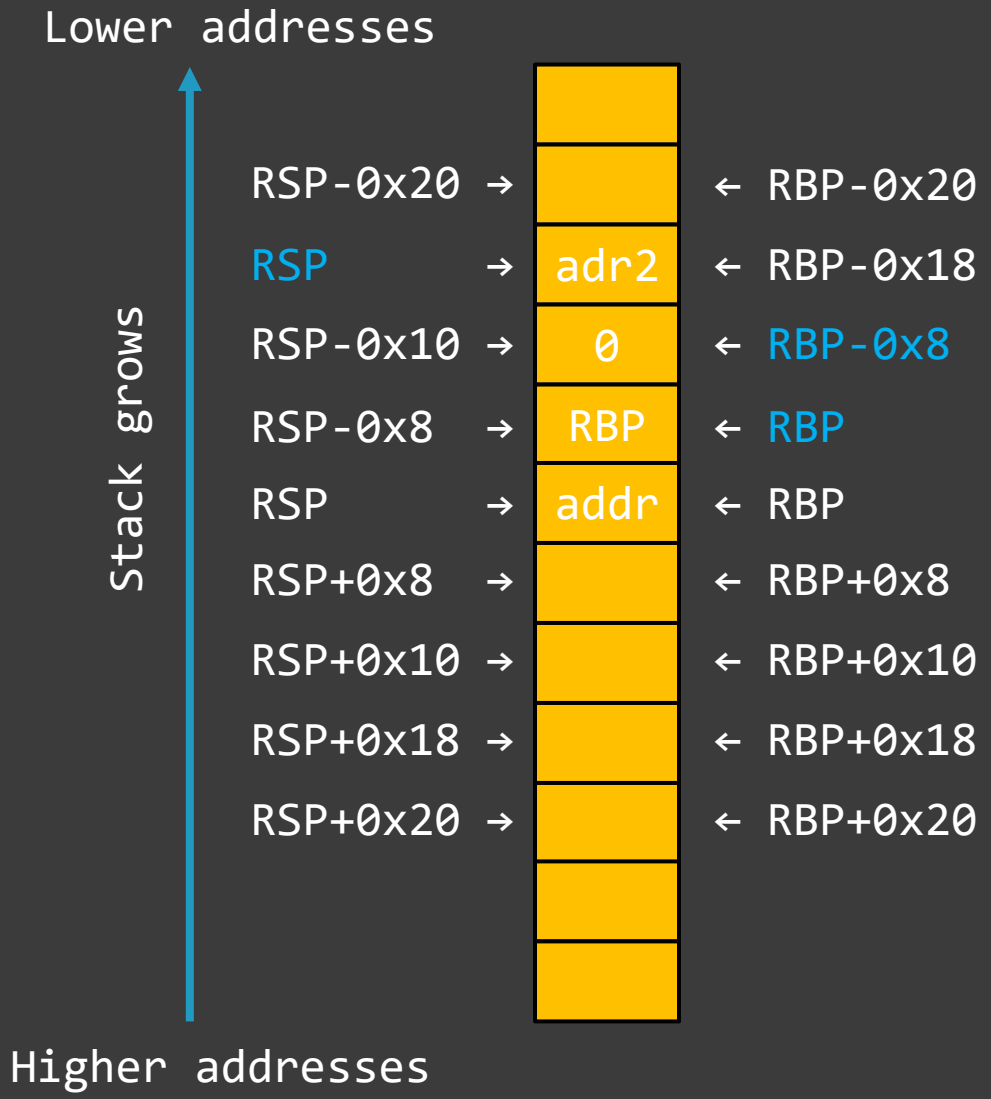
- Opcode DST
- Examples:

```
jmp    0x10493fc1c        # 0x10493fc1c → RIP
                          # (goto 0x10493fc1c)

call   0x10493ff74        # RSP – 8 → RSP
0x10493fc14:              # 0x10493fc14 → value at address RSP
                          # 0x10493ff74 → RIP
                          # (goto 0x10493ff74)
```

© 2025 Software Diagnostics Services

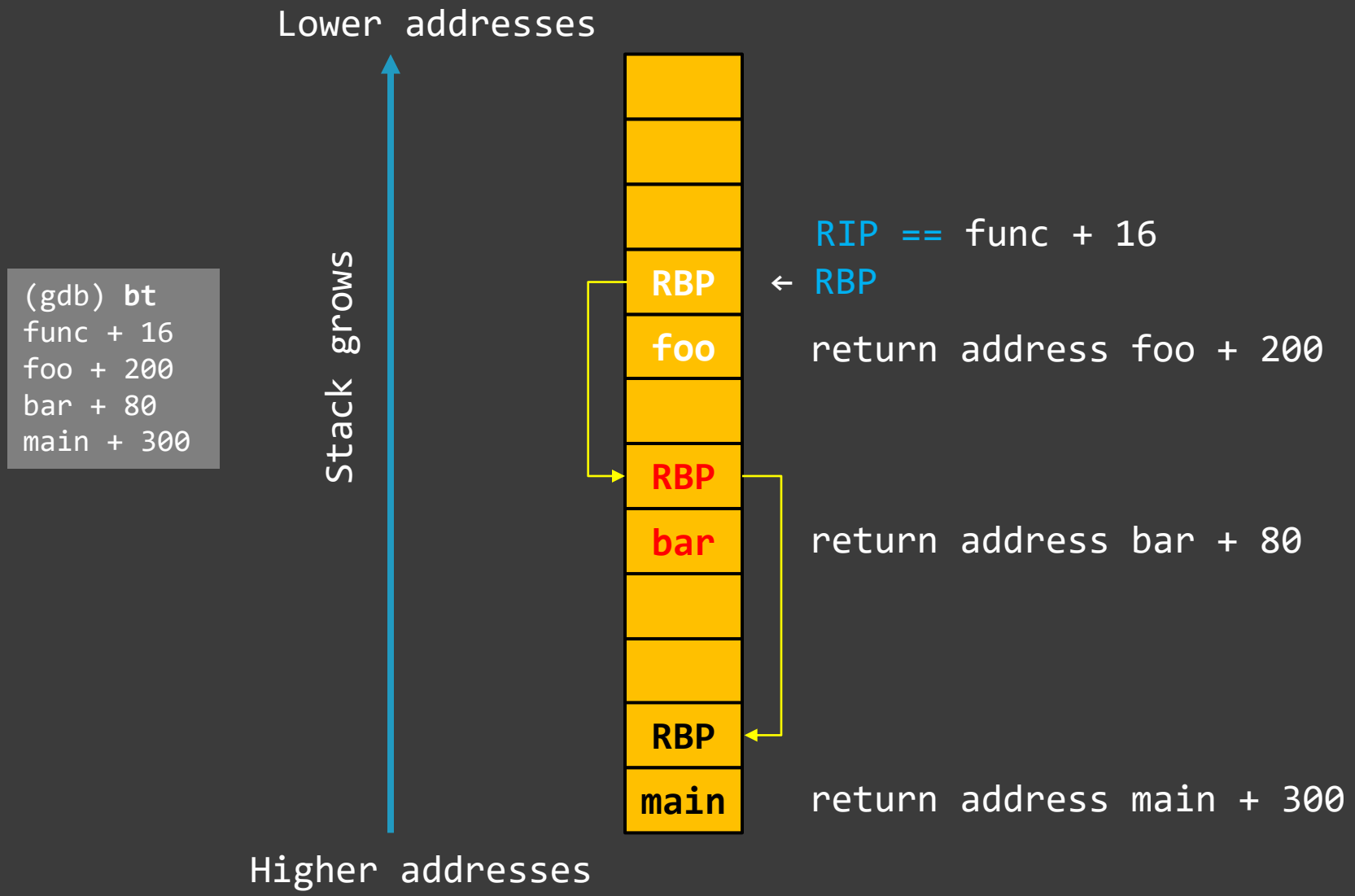
Function Call and Prolog (x64)

```
# void proc(int p1, long p2);
mov  $0x1, %edi
mov  $0x2, %rsi
call proc
addr:


# void proc2();
# void proc(int p1, long p2) {
#   long local = 0;
#   proc2();
# }
proc:
push %rbp
mov  %rsp, %rbp
sub  $0x8, %rsp
mov  $0, -0x8(%rbp)
call proc2
adr2:
...
```



© 2025 Software Diagnostics Services

Stack Trace Reconstruction (x64)



© 2025 Software Diagnostics Services

Function Parameters (x64)

- `foo(p1, p2, p3, …);`
- Left to right via `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9`, stack

© 2025 Software Diagnostics Services

Part 4: ARM64 Disassembly

© 2025 Software Diagnostics Services

CPU Registers (ARM64)

- **X0** – **X28**, **W0** – **W28**
- Stack: **SP**, **X29** (**FP**)
- Next instruction: **PC**
- Link register: **X30** (**LR**)
- Zero register: **XZR**, **WZR**
- 64-bit floating point registers **D0** – **D31**

GDB Commands

```
info registers
```

WinDbg Commands

```
r
```

© 2025 Software Diagnostics Services

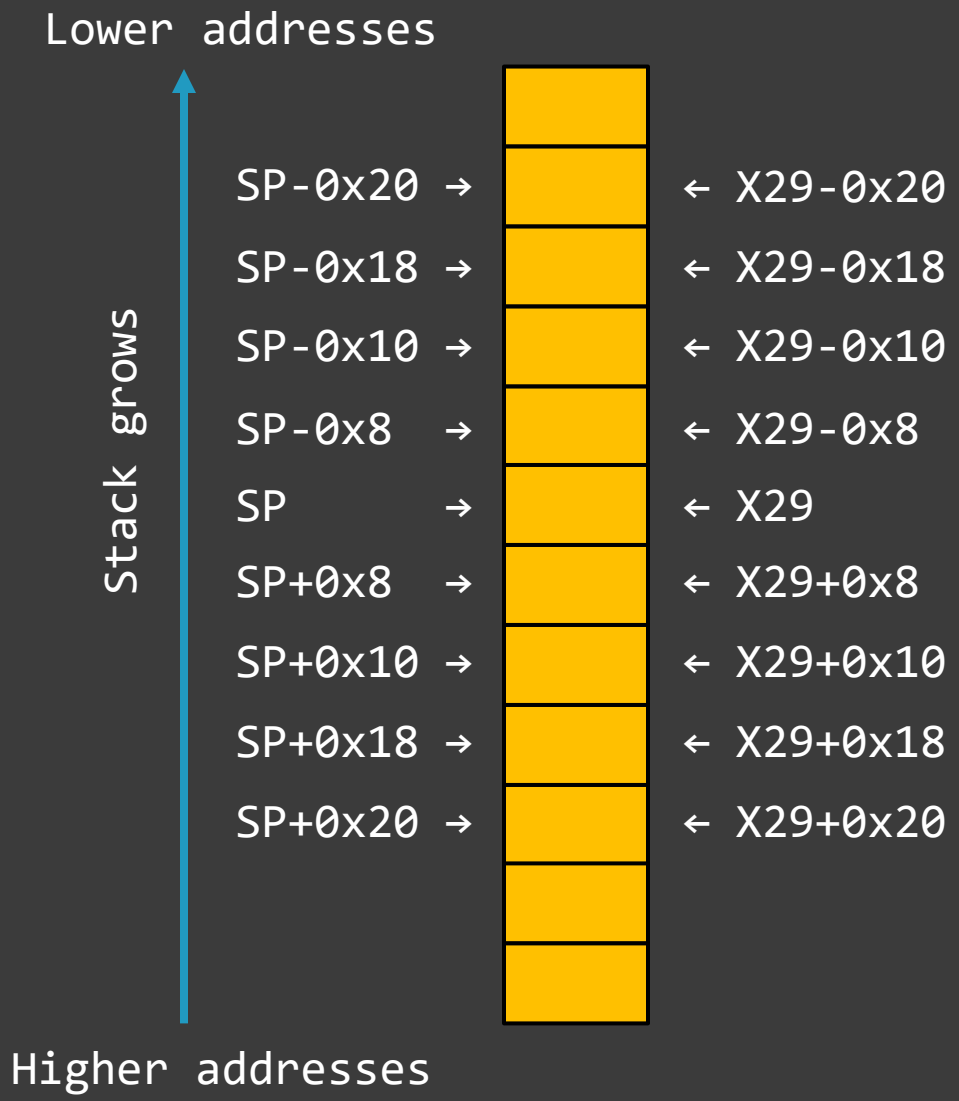
Instructions: registers (ARM64)

- `Opcode DST, SRC, SRC2`
- Examples:

```
mov   x0, #16                 // X0 ← 16 (0x10)
mov   x29, sp                 // X29 ← SP
add   x1, x2, #16             // X1 ← X2+16 (0x10)
mul   x1, x2, x3              // X1 ← X2*X3
blr   x8                      // X8 already contains
                              //    the address of func (&func)
                              // LR ← PC+4; PC ← &func

sub   sp, sp, #48             // SP ← SP-48 (-0x30)
                              // make a room for local variables
```

© 2025 Software Diagnostics Services

Memory and Stack Addressing



© 2025 Software Diagnostics Services

Instructions: memory load (ARM64)

- `Opcode DST, DST2, [SRC, Offset]`
- `Opcode DST, DST2, [SRC], Offset // Postincrement`
- Examples:

```
ldr   x0, [sp]              // X0 ← value at address SP+0
ldr   x0, [x29, #-8]        // X0 ← value at address X29-0x8
ldp   x29, x30, [sp, #32]   // X29 ← value at address SP+32 (0x20)
                            // X30 ← value at address SP+40 (0x28)
ldp   x29, x30, [sp], #16   // X29 ← value at address SP+0
                            // X30 ← value at address SP+8
                            // SP ← SP+16 (0x10)
```

© 2025 Software Diagnostics Services

Instructions: memory store (ARM64)

- `Opcode SRC, SRC2, [DST, Offset]`
- `Opcode SRC, SRC2, [DST, Offset]! // Preincrement`
- Examples:

```
str   x0, [sp, #16]          // x0 → value at address SP+16 (0x10)
str   x0, [x29, #-8]         // x0 → value at address X29-8
stp   x29, x30, [sp, #32]    // x29 → value at address SP+32 (0x20)
                             // x30 → value at address SP+40 (0x28)
stp   x29, x30, [sp, #-16]! // SP ← SP-16 (-0x10)
                             // x29 → set value at address SP
                             // x30 → set value at address SP+8
```

© 2025 Software Diagnostics Services

Instructions: indexing (ARM64)

- Opcode DST, [SRC, SRC_{offset}, Shift]
- Opcode SRC, [DST, DST_{Offset}, Shift]
- Example:

```
ldr     x0, [x0, x1, lsl #3]    // X0 ← value at address X0+X1<<3
                                //          (X0+X1*8)
```

© 2025 Software Diagnostics Services

Instructions: flow (ARM64)

- `Opcode DST, SRC`
- Examples:

```
adrp   x0, 0x420000       // x0 ← 0x420000

b      0x10493fc1c        // PC ← 0x10493fc1c
                          // (goto 0x10493fc1c)
br     x17                // PC ← the value of X17

0x10493fc14:              // PC == 0x10493fc14
bl     0x10493ff74        // LR ← PC+4 (0x10493fc18)
                          // PC ← 0x10493ff74
                          // (goto 0x10493ff74)
```

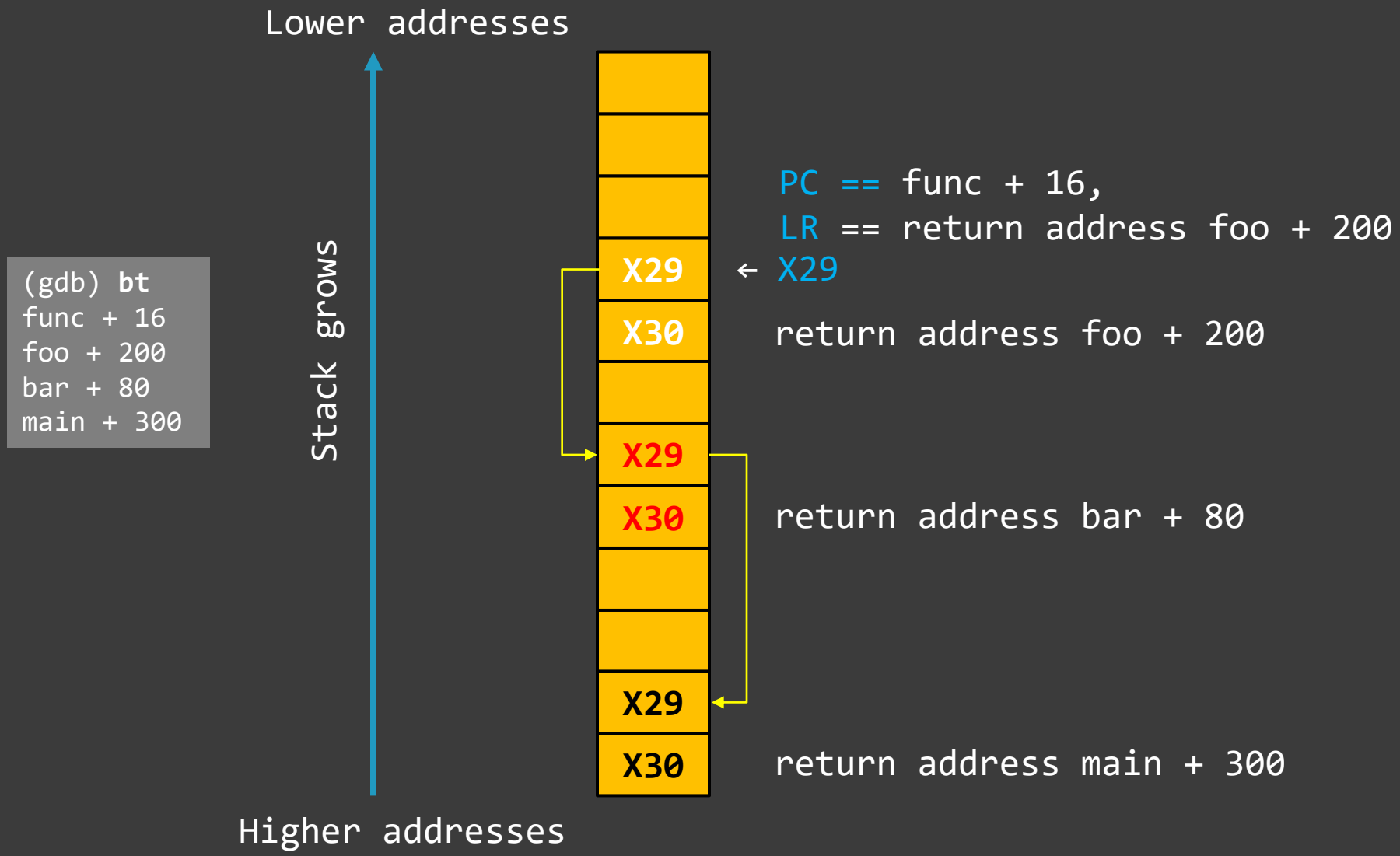
© 2025 Software Diagnostics Services

Function Call and Prolog (ARM64)

```
// void proc(int p1, long p2);
mov  w0, #0x1
mov  x1, #0x2
bl   proc


// void proc2();
// void proc(int p1, long p2) {
//   long local = 0;
//   proc2();
// }
proc:
stp  x29, x30, [sp, #-32]!
mov  x29, sp
str  zxr, [x29, #16]
bl   proc2
...
```

© 2025 Software Diagnostics Services

Stack Trace Reconstruction (ARM64)



© 2025 Software Diagnostics Services

Function Parameters (ARM64)

- `foo(p1, p2, p3, …);`
- Left to right via `X0 – X7, [SP], [SP+8], [SP+16], ...`

© 2025 Software Diagnostics Services

Part 5: x64 Disassembly

(Intel WinDbg Flavor)

© 2025 Software Diagnostics Services

CPU Registers (x64)

- **RAX** ⊃ **EAX** ⊃ **AX** ⊇ {**AH**, **AL**}
- ALU: **RAX**, **RDX**
- Counter: **RCX**
- Memory copy: **RSI** (src), **RDI** (dst)
- Stack: **RSP**, **RBP**
- Next instruction: **RIP**
- New: **R8** – **R15**, **Rx**(**D**|**W**|**B**)

WinDbg Commands

```
r
```

© 2025 Software Diagnostics Services

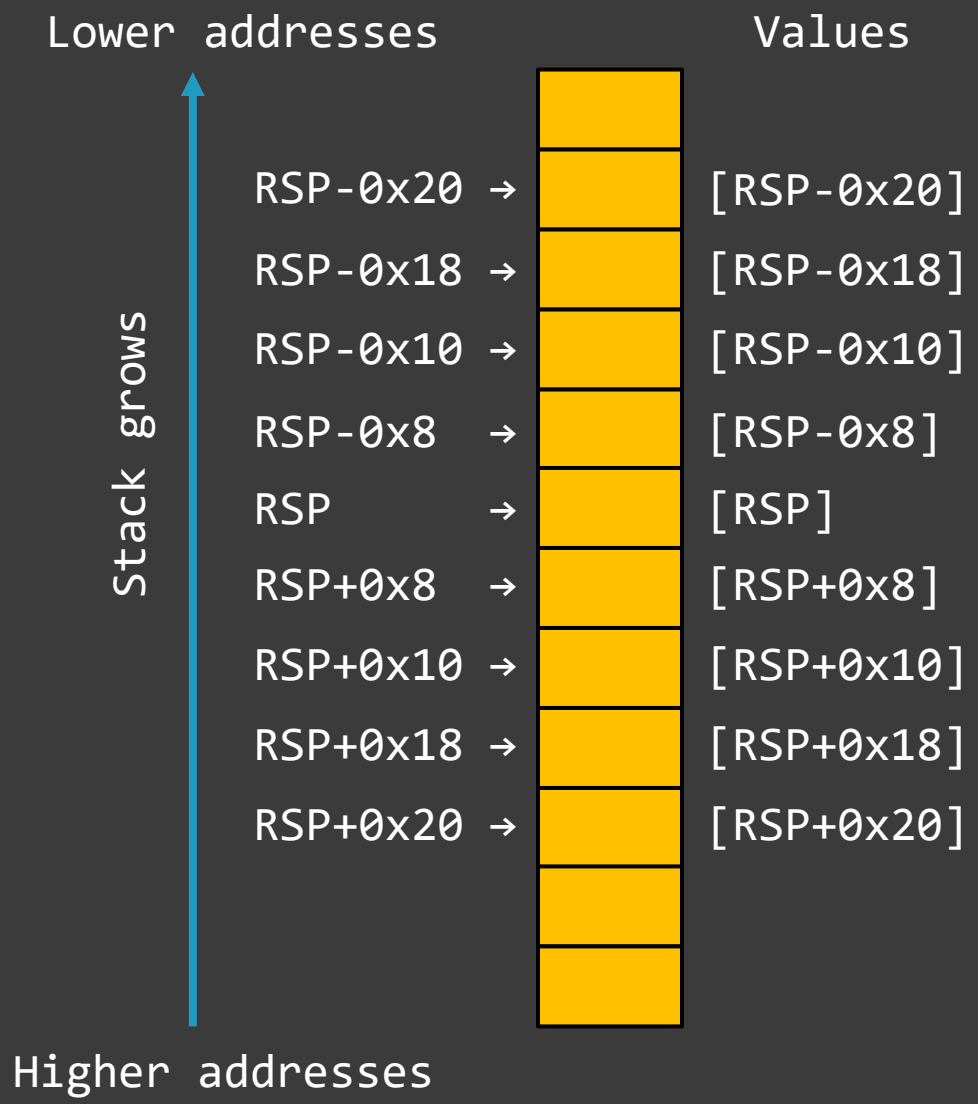
Instructions and Registers (x64)

- `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
```

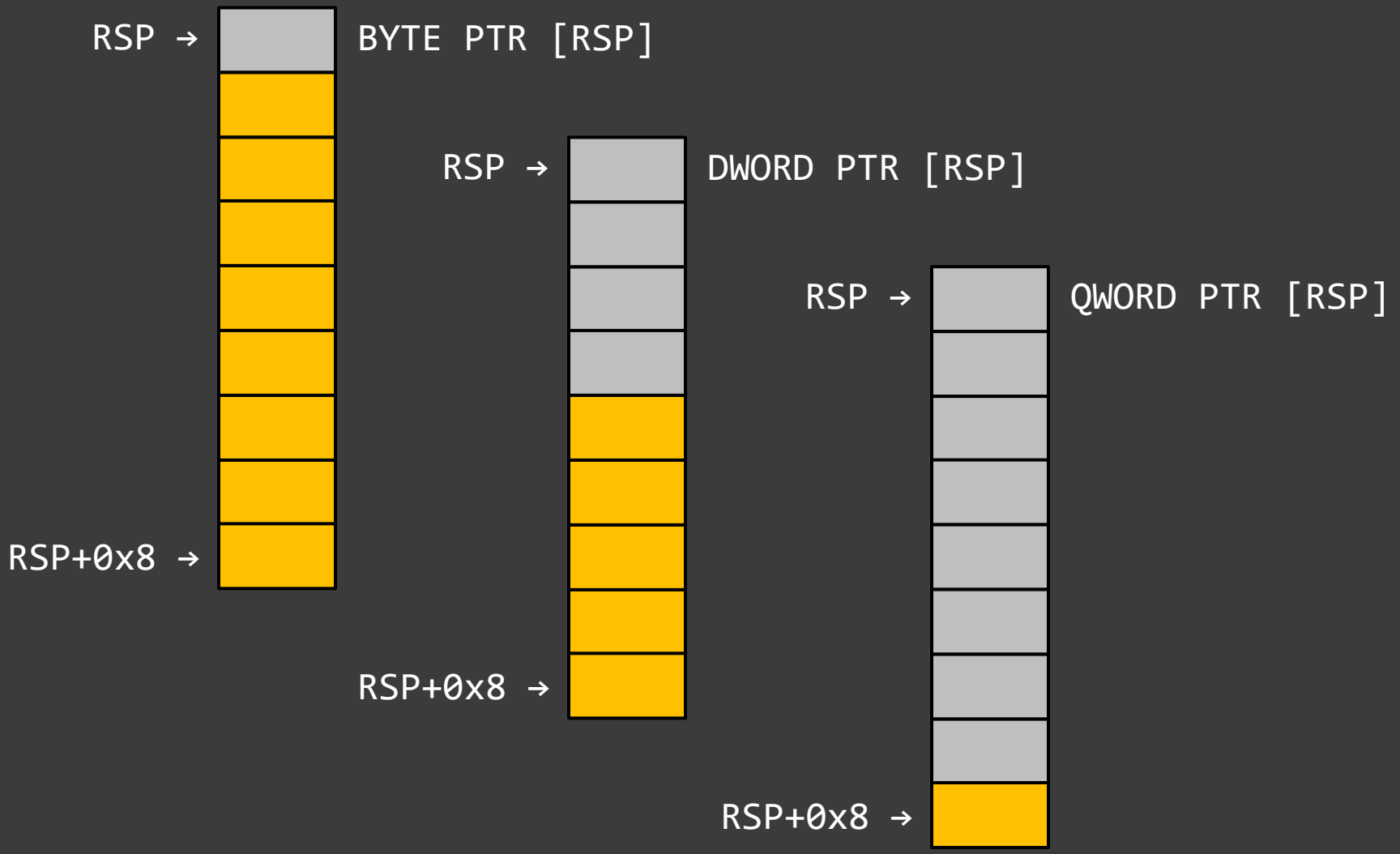
© 2025 Software Diagnostics Services

Memory and Stack Addressing



© 2025 Software Diagnostics Services

Memory Cell Sizes



© 2025 Software Diagnostics Services

Memory Load Instructions (x64)

- `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
```

© 2025 Software Diagnostics Services

Memory Store Instructions (x64)

- `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
```

© 2025 Software Diagnostics Services

Instructions: indexing (x64)

- Opcode DST, PTR [SRC_{base} + SRC_{index}*Scale + Offset]
- Opcode PTR [DST_{base} + DST_{index}*Scale + Offset], SRC
- Examples:

```
mov   rax, qword ptr [rdx+rax]  # RAX ← value at address RDX+RAX*1

mov   qword ptr [rdx+rax*8], 0  # value at address RDX+RAX*8 ← 0

lea   rdx, [rax*8]              # RDX ← address RAX*8
```

© 2025 Software Diagnostics Services

Flow Instructions (x64)

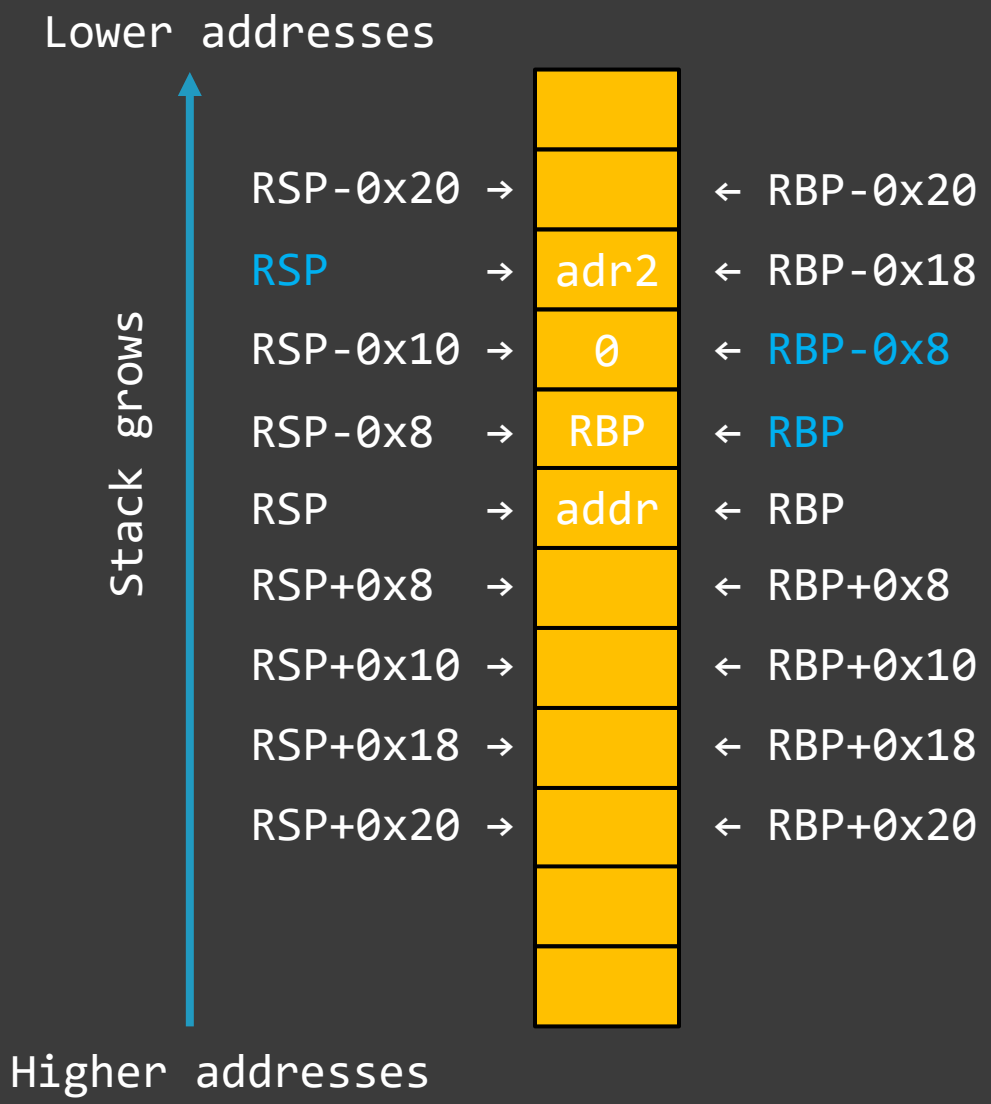
- `Opcode DST`
- `Opcode PTR [DST]`
- Examples:

```
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)
```

© 2025 Software Diagnostics Services

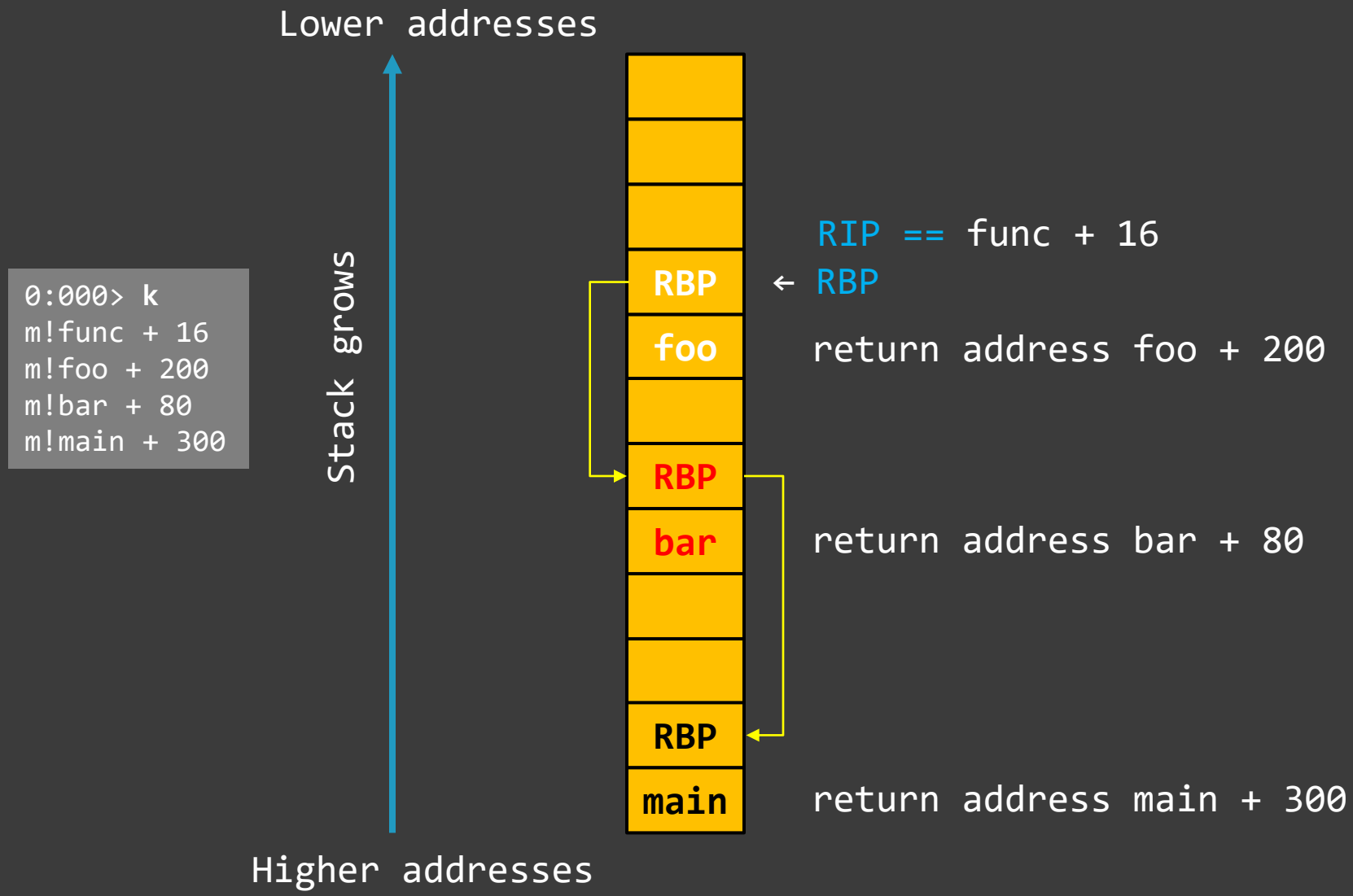
Function Call and Prolog (x64)

```
# void proc(int p1, long p2);
mov  edi, 1
mov  rsi, 2
call proc
addr:


# void proc2();
# void proc(int p1, long p2) {
#   long local = 0;
#   proc2();
# }
proc:
push rbp
mov  rbp, rsp
sub  rsp, 8
mov  qword ptr [rbp-8], 0
call proc2
adr2:
...
```



© 2025 Software Diagnostics Services

Stack Trace Reconstruction (x64)



© 2025 Software Diagnostics Services

Function Parameters (x64)

- `foo(p1, p2, p3, …);`
- Left to right via `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9`, stack

© 2025 Software Diagnostics Services

Part 6: Practice Exercises

© 2025 Software Diagnostics Services

Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in this book

© 2025 Software Diagnostics Services

Exercise 0

- **Goal:** Install GDB and check if GDB loads a core dump correctly
- **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
- \ALCDA-Dumps\Exercise-A0-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A0-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A0-A64-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Process Core Dumps

Exercises A1 – A12

© 2025 Software Diagnostics Services

Exercise A1

- **Goal:** Learn how to list stack traces, disassemble functions, check their correctness, dump data, get environment

- **Patterns:** Manual Dump (Process); Stack Trace; Incorrect Stack Trace; Unrecognizable Symbolic Information; Truncated Stack Trace; Stack Trace Collection; Annotated Disassembly; Paratext; Not My Version; Environment Hint

- \ALCDA-Dumps\Exercise-A1-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A1-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A1-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A1-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise A2D

- **Goal:** Learn how to identify exceptions, find problem threads and CPU instructions

- **Patterns:** Crash Signature; NULL Pointer (Data); Active Thread (x64, GDB)

- \ALCDA-Dumps\Exercise-A2D-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A2D-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A2D-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A2D-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

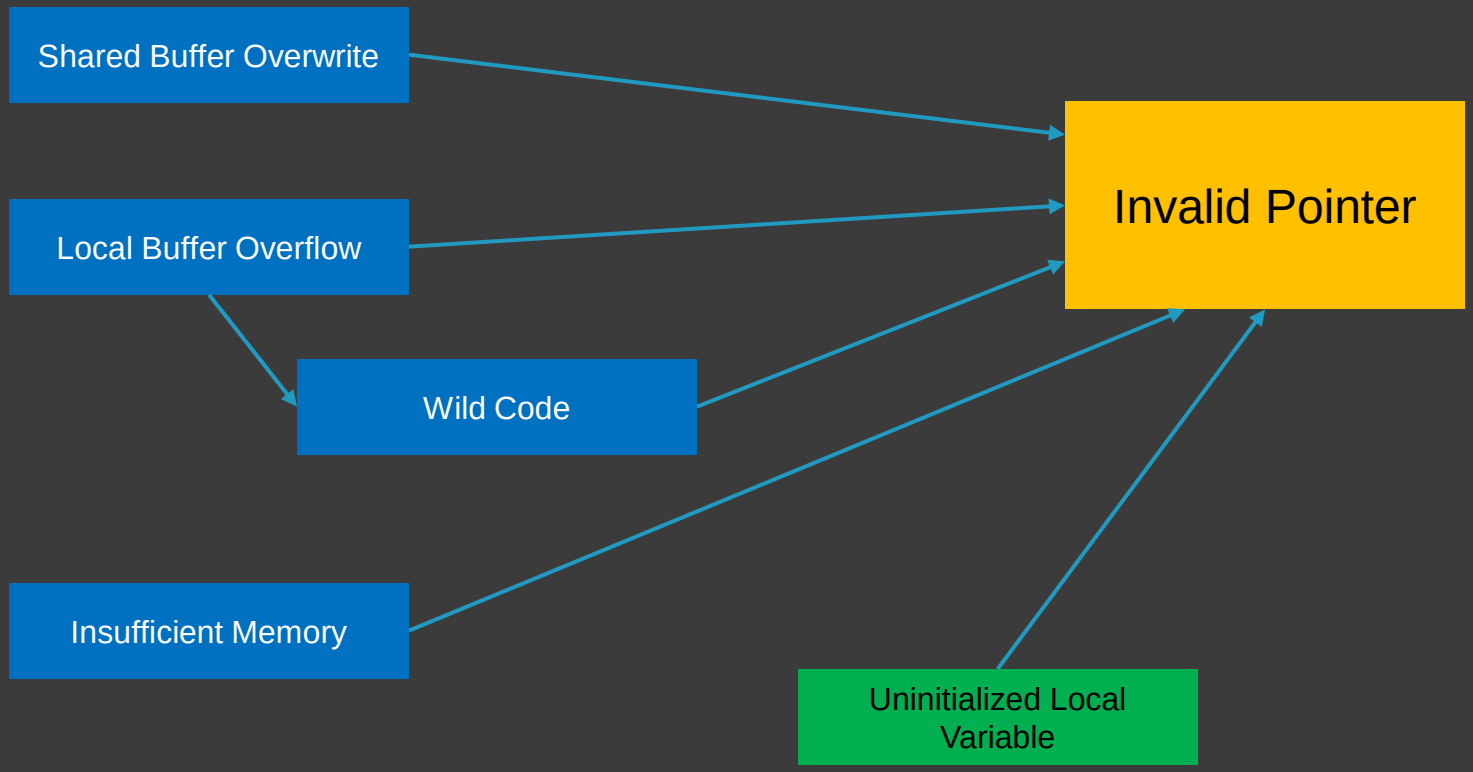
Exercise A2C

- **Goal:** Learn how to identify exceptions, find problem threads and CPU instructions
- **Patterns:** NULL Pointer (Code); Disassembly Ambiguity (WinDbg)
- \ALCDA-Dumps\Exercise-A2C-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A2C-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A2C-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A2C-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Mechanisms (Invalid Pointer)

Patterns-Based Root Cause Analysis Methodology



© 2025 Software Diagnostics Services

Exercise A2S

- **Goal:** Learn how to use external debugging information

- \ALCDA-Dumps\Exercise-A2S-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A2S-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A2S-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A2S-x64-WinDbg.pdf

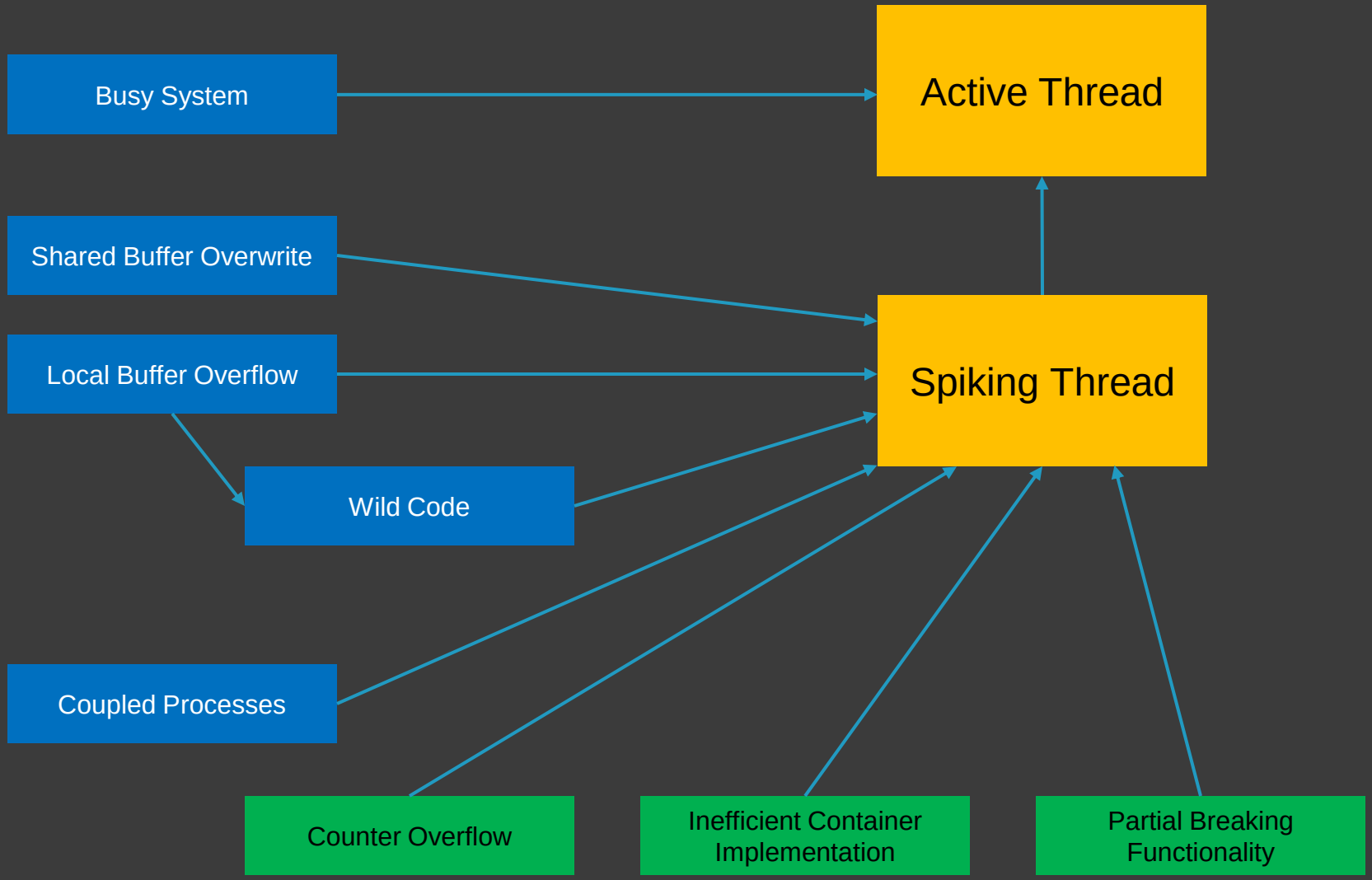
© 2025 Software Diagnostics Services

Exercise A3

- **Goal:** Learn how to identify spiking threads
- **Patterns:** Active Thread; Spiking Thread; Well-Tested Function
- \ALCDA-Dumps\Exercise-A3-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A3-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A2C-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A2C-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Mechanisms (Active Thread)



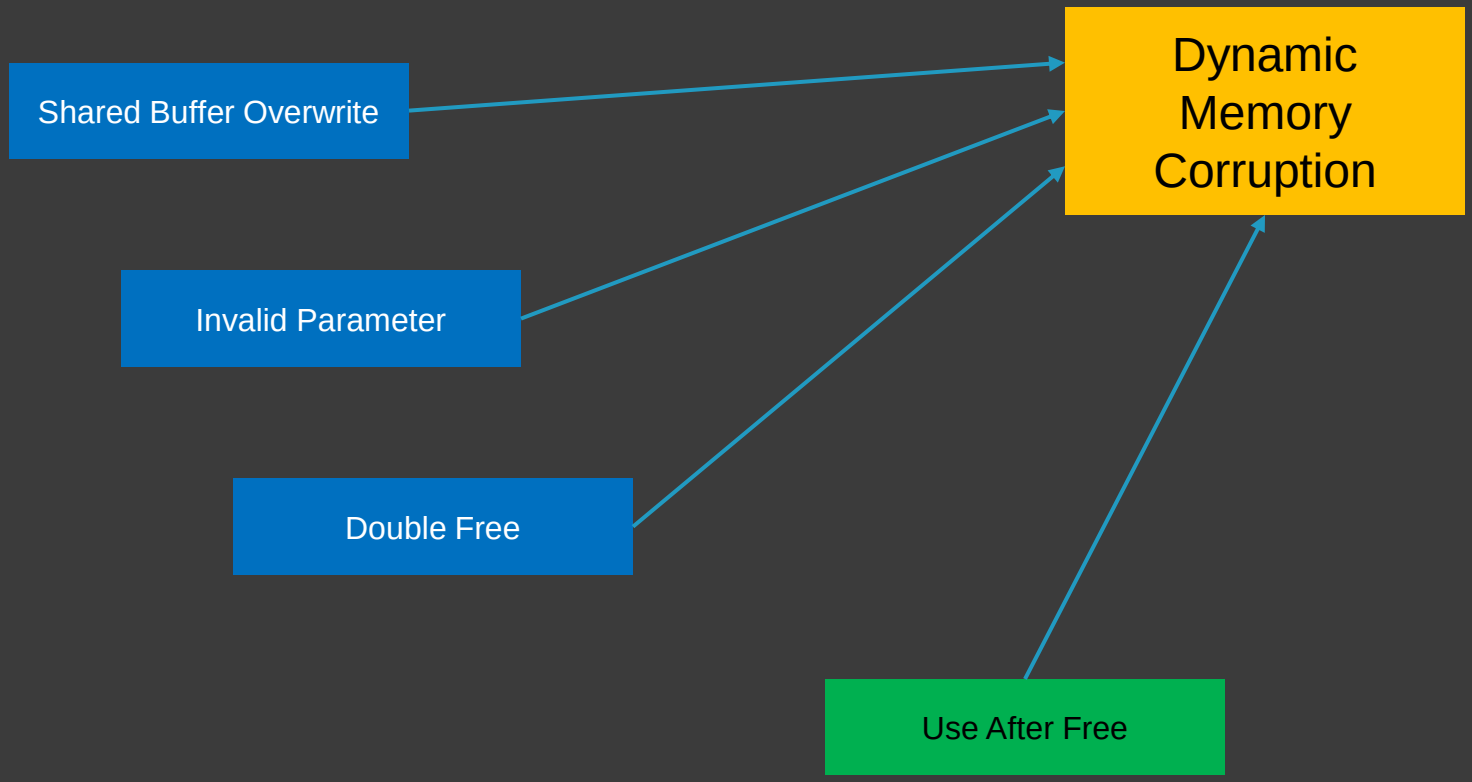
© 2025 Software Diagnostics Services

Exercise A4

- **Goal:** Learn how to identify heap regions and heap corruption
- **Patterns:** Dynamic Memory Corruption (Process Heap); Regular Data; Wild Pointer
- \ALCDA-Dumps\Exercise-A4-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A4-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A4-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A4-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Mechanisms (Heap Corruption)



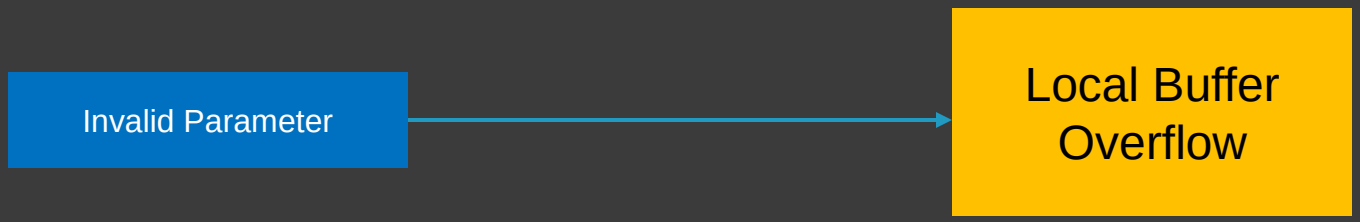
© 2025 Software Diagnostics Services

Exercise A5

- **Goal:** Learn how to identify stack corruption
- **Patterns:** Local Buffer Overflow (User Space); Execution Residue (User Space); Past Stack Trace
- \ALCDA-Dumps\Exercise-A5-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A5-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A5-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A5-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Mechanisms (Stack Corruption)



© 2025 Software Diagnostics Services

Exercise A6

- **Goal:** Learn how to identify stack overflow, stack boundaries, reconstruct stack trace

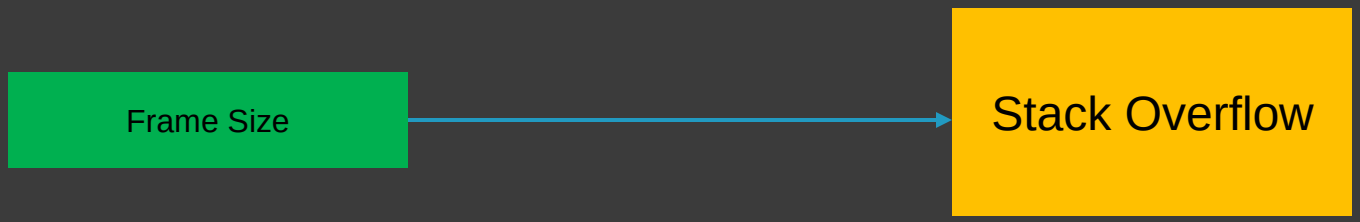
- **Patterns:** Stack Overflow (User Mode)

- \ALCDA-Dumps\Exercise-A6-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A6-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A6-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A6-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Mechanisms (Stack Overflow)



© 2025 Software Diagnostics Services

Exercise A7

- **Goal:** Learn how to identify active threads
- **Patterns:** Divide by Zero (User Mode, x64); Invalid Pointer (General); Multiple Exceptions (User Mode); Near Exception
- \ALCDA-Dumps\Exercise-A7-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A7-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise A8

- **Goal:** Learn how to identify runtime exceptions, past execution residue and stack traces, identify handled exceptions

- **Patterns:** C++ Exception; Coincidental Symbolic Information; Handled Exception (User Space)

- \ALCDA-Dumps\Exercise-A8-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A8-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A8-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A8-x64-WinDbg.pdf

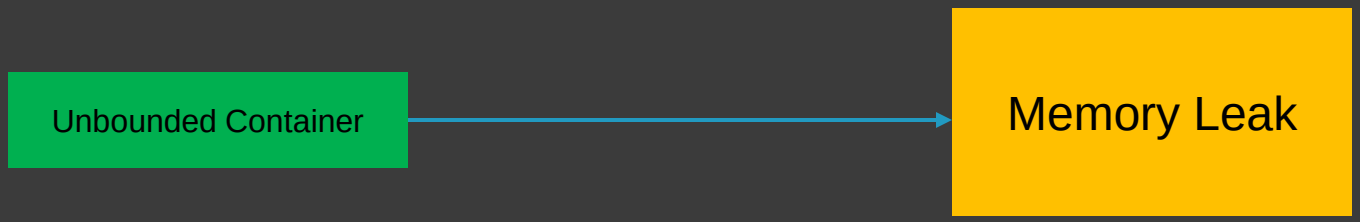
© 2025 Software Diagnostics Services

Exercise A9

- **Goal:** Learn how to identify heap leaks
- **Patterns:** Memory Leak (Process Heap); Module Hint; Historical Information
- \ALCDA-Dumps\Exercise-A9-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A9-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A9-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A9-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Mechanisms (Memory Leak)



© 2025 Software Diagnostics Services

Exercise A10

- **Goal:** Learn how to identify heap contention wait chains, synchronization issues, advanced disassembly, dump arrays
- **Patterns:** Double Free (Process Heap); High Contention (Process Heap); Wait Chain (General); Critical Region; Self-Diagnosis (User Mode)
- \ALCDA-Dumps\Exercise-A10-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A10-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-A10-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A10-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise A11

- **Goal:** Learn how to identify synchronization wait chains, deadlocks, hidden and handled exceptions

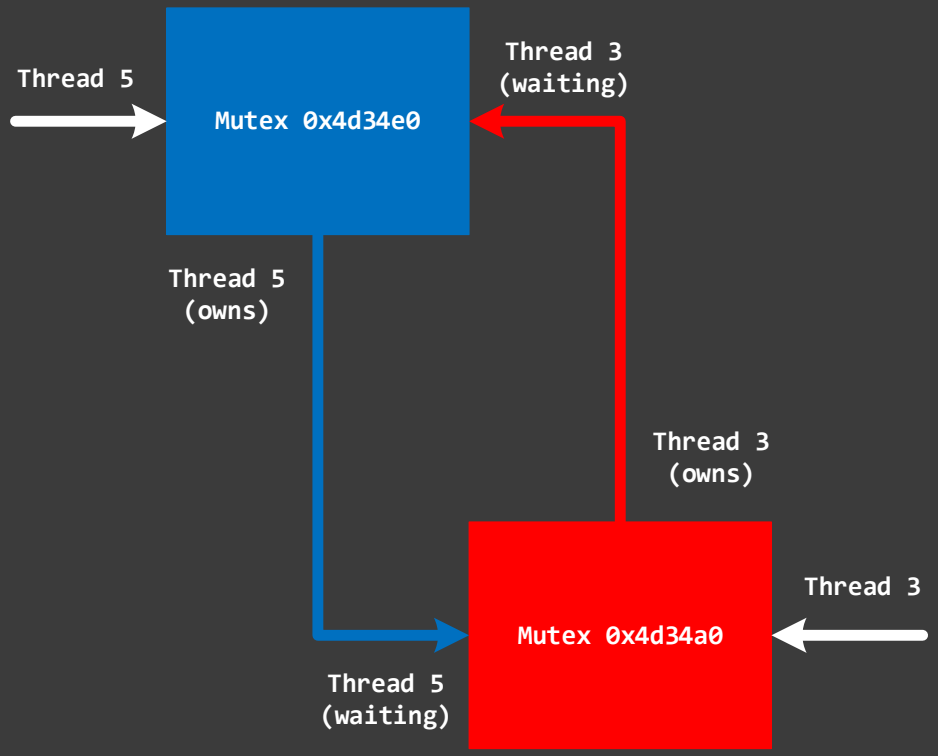
- **Patterns:** Wait Chain (Mutex Objects); Deadlock (Mutex Objects, User Space); Disassembly Hole (WinDbg)

- \ALCDA-Dumps\Exercise-A11-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A11-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A11-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A11-x64-WinDbg.pdf

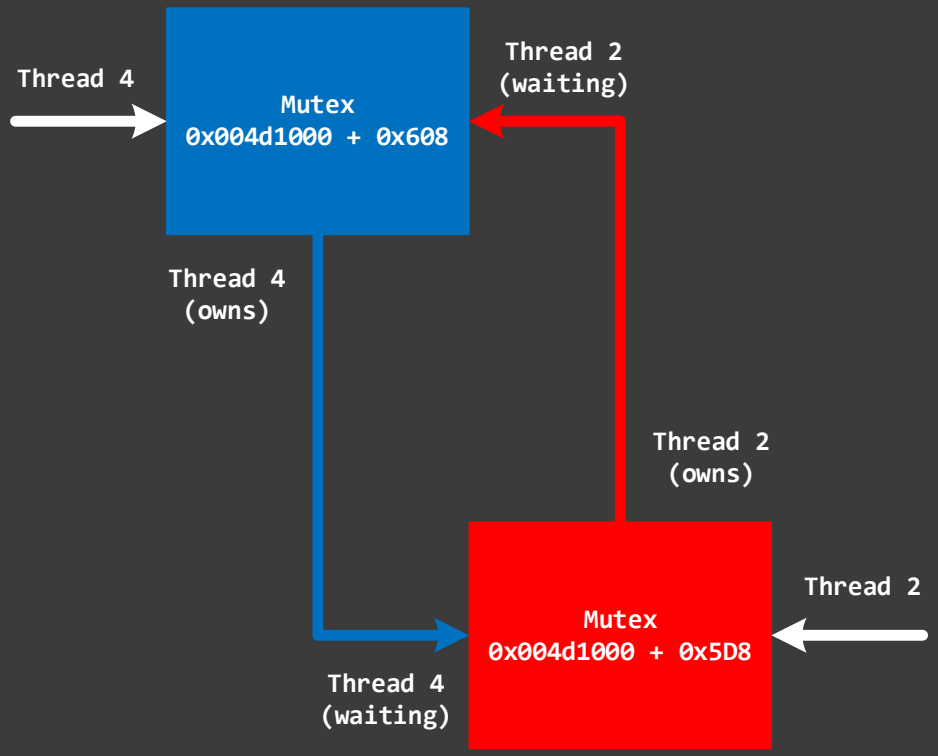
© 2025 Software Diagnostics Services

Deadlock (x64, GDB)



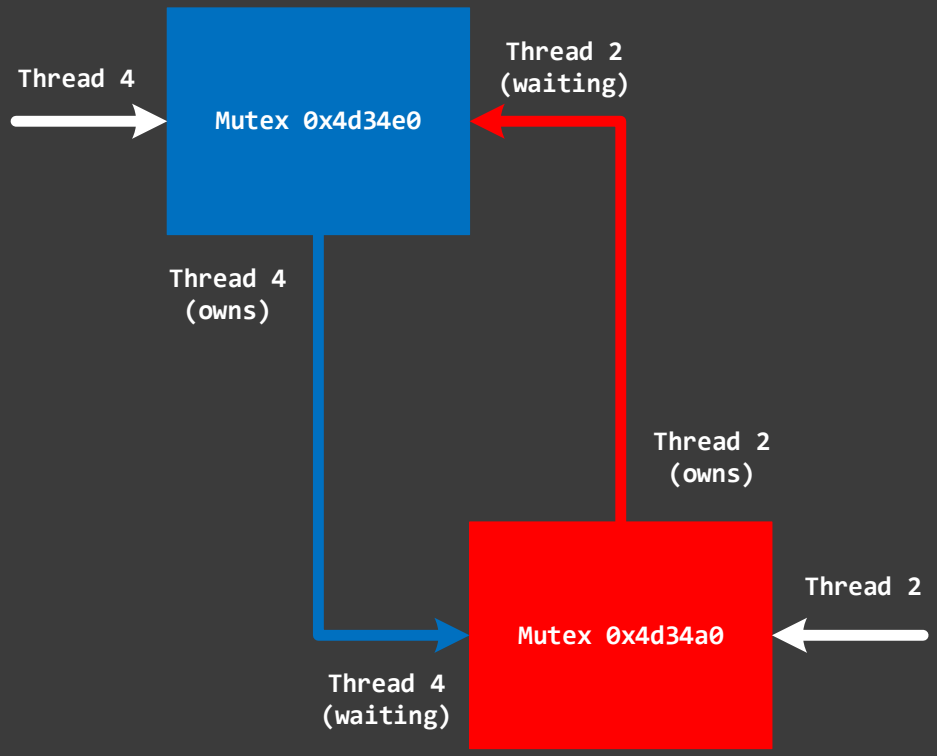
© 2025 Software Diagnostics Services

Deadlock (A64, GDB)



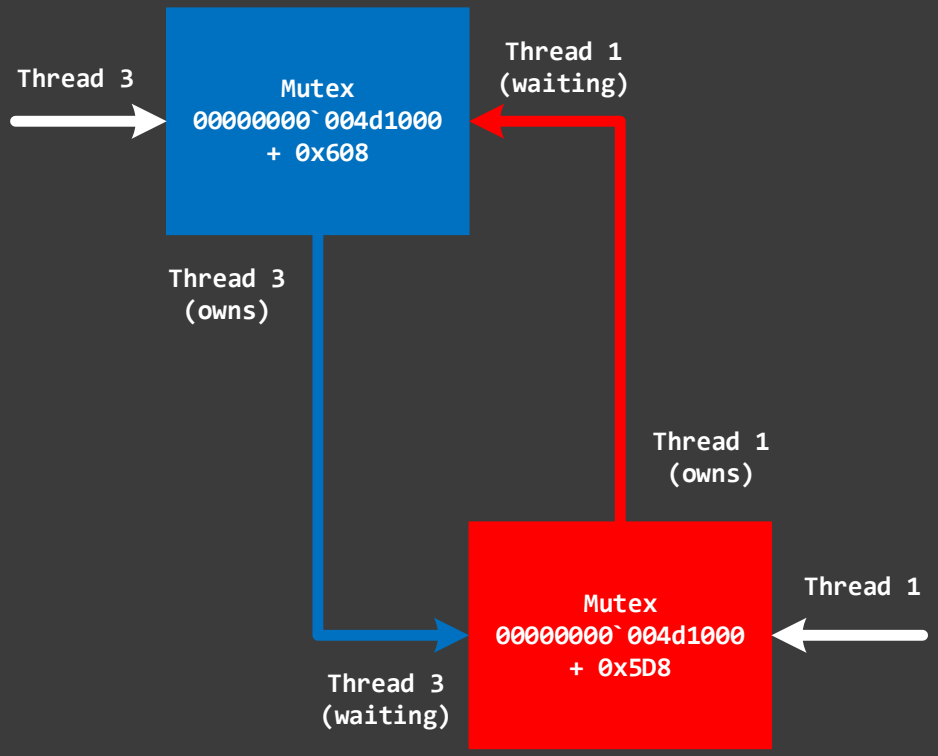
© 2025 Software Diagnostics Services

Deadlock (x64, WinDbg)



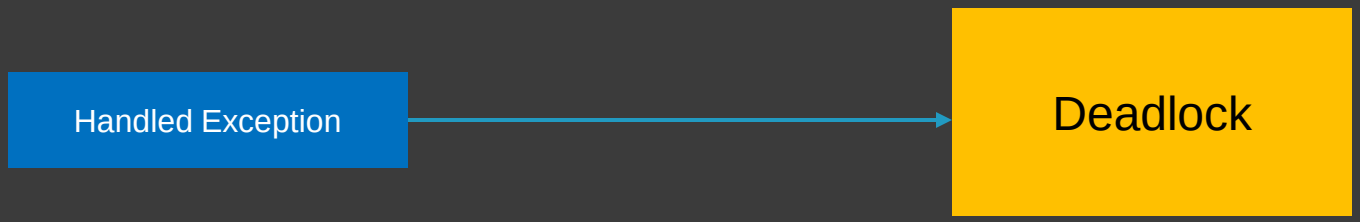
© 2025 Software Diagnostics Services

Deadlock (A64, WinDbg)



© 2025 Software Diagnostics Services

Mechanisms (Deadlock)



© 2025 Software Diagnostics Services

Exercise A12

- **Goal:** Learn how to dump memory for post-processing, get the list of functions and module variables, load symbols, disassemble with source code, inspect arguments, local variables, and types

- **Patterns:** Module Variable

- \ALCDA-Dumps\Exercise-A12-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A12-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-A12-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-A12-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Kernel Core Dumps

Exercises K1 – K5

© 2025 Software Diagnostics Services

Exercise K1

- **Goal:** Learn how to navigate a normal kernel dump
- **Patterns:** Manual Dump (Kernel); Stack Trace Collection
- \ALCDA-Dumps\Exercise-K1-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-K1-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-K1-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-K1-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise K2

- **Goal:** Learn how to navigate a problem kernel dump
- **Patterns:** Exception Stack Trace; Exception Module; NULL Pointer (Data); Execution Residue (Kernel Space); Value References
- \ALCDA-Dumps\Exercise-K2-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-K2-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-K2-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-K2-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise K3

- **Goal:** Learn how to recognize problems with kernel threads, identify their owner module, follow call chains, and reconstruct a stack trace if necessary

- **Patterns:** Origin Module; NULL Pointer (Code); Hidden Call

- \ALCDA-Dumps\Exercise-K3-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-K3-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-K3-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-K3-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise K4

- **Goal:** Learn how to identify spiking kernel threads
- **Patterns:** Stack Trace Collection (CPUs); Interrupt Stack; Active Thread; Spiking Thread
- \ALCDA-Dumps\Exercise-K4-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-K4-A64-GDB.pdf
- \ALCDA-Dumps\Exercise-K4-A64-WinDbg.pdf

© 2025 Software Diagnostics Services

Exercise K5

- **Goal:** Learn how to identify kernel stack overflow and kernel stack boundaries

- **Patterns:** Stack Overflow (Kernel Mode); Debugger Bug

- \ALCDA-Dumps\Exercise-K5-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-K5-A64-GDB.pdf

- \ALCDA-Dumps\Exercise-K5-A64-WinDbg.pdf
- \ALCDA-Dumps\Exercise-K5-x64-WinDbg.pdf

© 2025 Software Diagnostics Services

Follow-up Courses

Advanced Linux Core Dump Analysis with Data Structures

Accelerated Linux Disassembly, Reconstruction, and Reversing

Accelerated Linux Debugging[4]

Accelerated Linux API for Software Diagnostics

Accelerated C & C++ for Linux Diagnostics

© 2025 Software Diagnostics Services

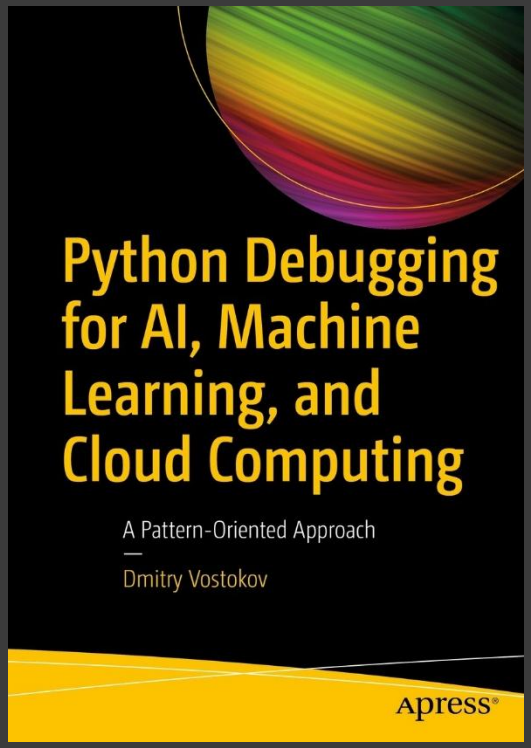
Rust Core Dump Analysis

Memory Thinking for Rust

Accelerated Rust Linux Core Dump Analysis

© 2025 Software Diagnostics Services

Python Core Dump Analysis



© 2025 Software Diagnostics Services

Educative Courses

Debugging, Disassembly & Reversing in Linux for x64 Architecture

Foundations of Linux ARM64: Debug, Disassemble, and Reverse

Accelerated Linux Core Dump Analysis

© 2025 Software Diagnostics Services

Pattern Links (Linux and GDB)

Active Thread
C++ Exception
Critical Region
Divide by Zero
Execution Residue
High Contention
Memory Leak
Local Buffer Overflow
Module Hint
Not My Version
NULL Pointer (Data)
Self-Diagnosis
Stack Overflow (User Mode)
Stack Trace Collection
Regular Data
Near Exception
Invalid Pointer
Past Stack Trace
Exception Stack Trace
Value References
Hidden Call
Interrupt Stack
Truncated Stack Trace
Disassembly Ambiguity
Wild Pointer
Debugger Bug

Annotated Disassembly
Coincidental Symbolic Information
Deadlock (Mutex Objects, User Space)
Environment Hint
Handled Exception
Dynamic Memory Corruption
Lateral Damage
Manual Dump (Process) / (Kernel)
Module Variable
NULL Pointer (Code)
Paratext
Spiking Thread
Stack Trace
Wait Chain (General)
Multiple Exceptions
Wait Chain (Mutex Objects)
Disassembly Hole
Deadlock (Mutex Objects, Kernel Space)
Origin Module
Stack Trace Collection (CPUs)
Stack Overflow (Kernel Mode)
Unrecognizable Symbolic Information
Crash Signature
Well-Tested Function
Historical Information
Exception Module

© 2025 Software Diagnostics Services

Resources

- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- Rosetta Stone for Debuggers
- Accelerated macOS Core Dump Analysis (also covers LLDB)
- GDB Pocket Reference
- GDB
- Intel® 64 and IA-32 Architectures Software Developer Manuals
- A64 Instruction Set Architecture
- A64 Base Instructions
- Encyclopedia of Crash Dump Analysis Patterns, Third Edition
- Memory Dump Analysis Anthology (Diagnomicon) (some articles in volumes 1, 7, 9A cover GDB)

© 2025 Software Diagnostics Services

Q&A

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

© 2025 Software Diagnostics Services

Thank you for attendance!

© 2025 Software Diagnostics Services