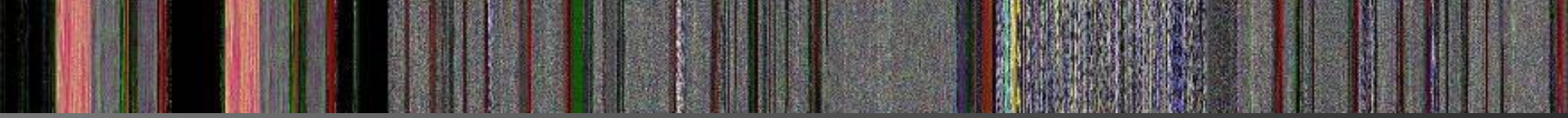


# Linux
# Core Dump Analysis
# Accelerated

**Second Edition**
**Revised and Extended**

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
- Beneficial to know basics of assembly language (depends on your platform):

Practical Foundations of Linux Debugging, Disassembling, Reversing

Practical Foundations of ARM64 Linux Debugging, Disassembling, Reversing

# Training Goals

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

# Training Principles

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

# Schedule Summary

## Day 1

- Analysis fundamentals (25 minutes)
- Process core dump collection (5 minutes)
- Process GDB core dump analysis (1 hour 30 minutes)

## Day 2

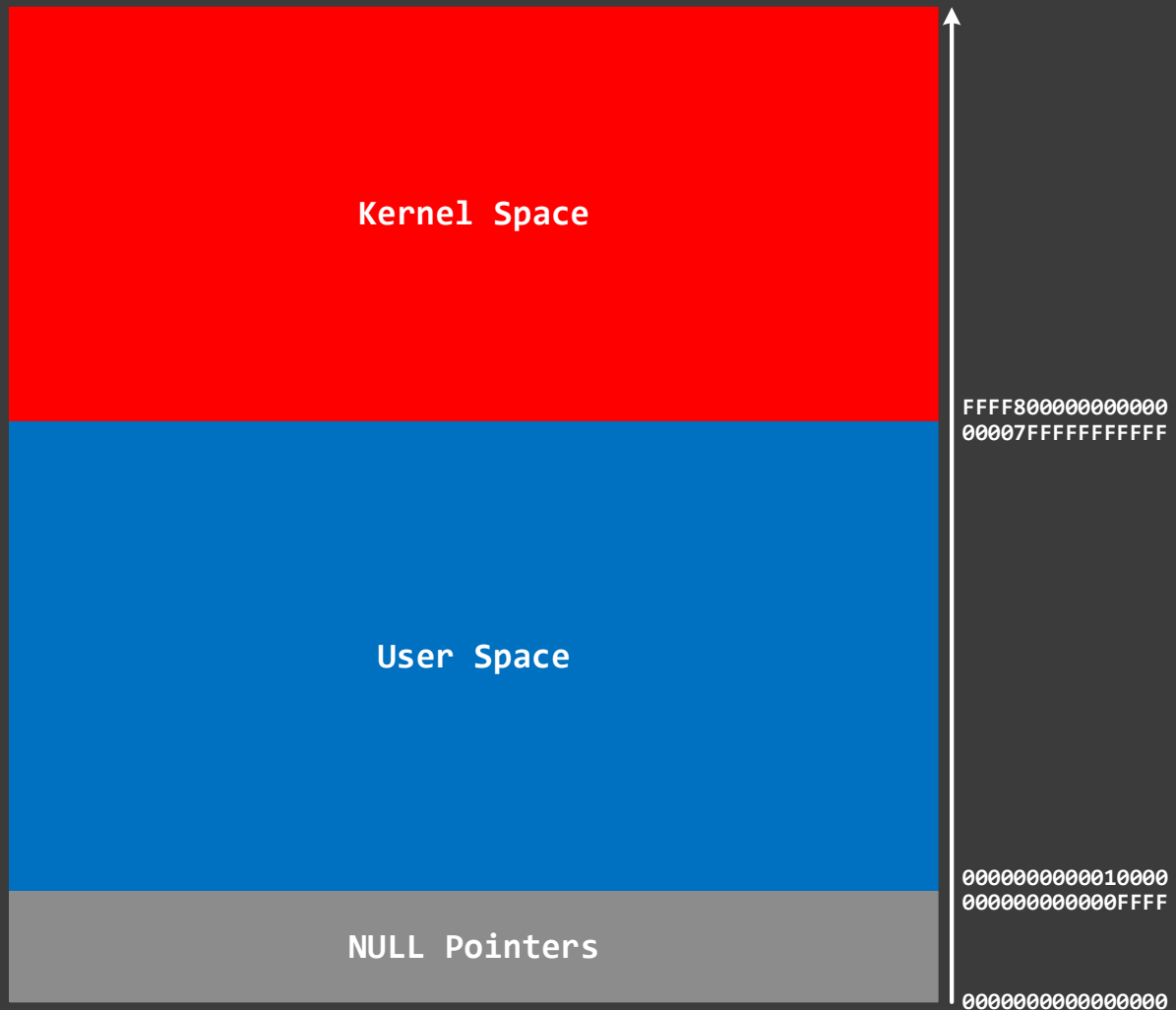
- Process GDB core dump analysis (1 hour 40 minutes)
- Process WinDbg core dump analysis (20 minutes)

## Day 3

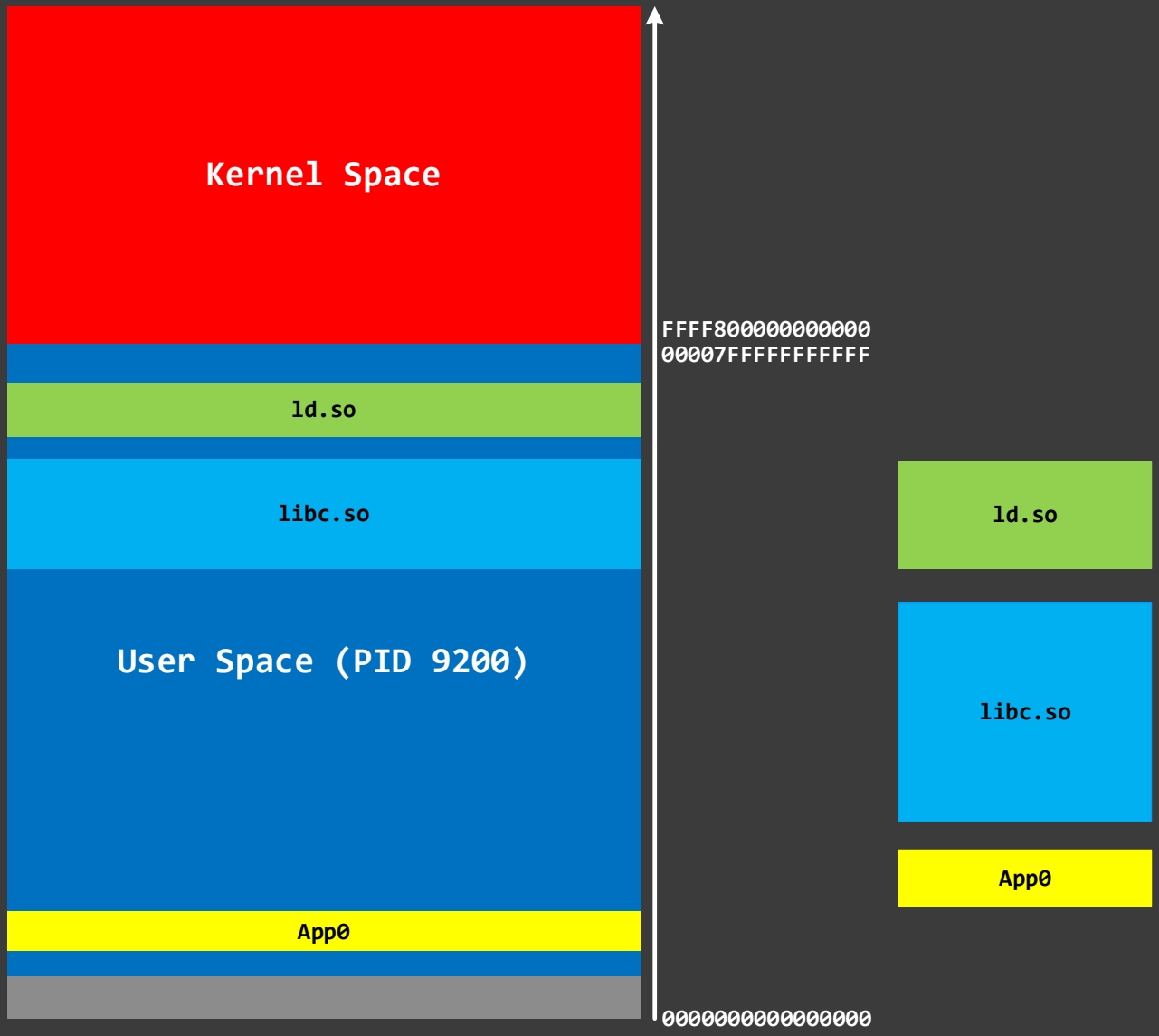
- Kernel core dump collection (5 minutes)
- Kernel core dump analysis (1 hour 25 minutes)
- Process WinDbg core dump analysis (30 minutes)

# Part 1: Fundamentals

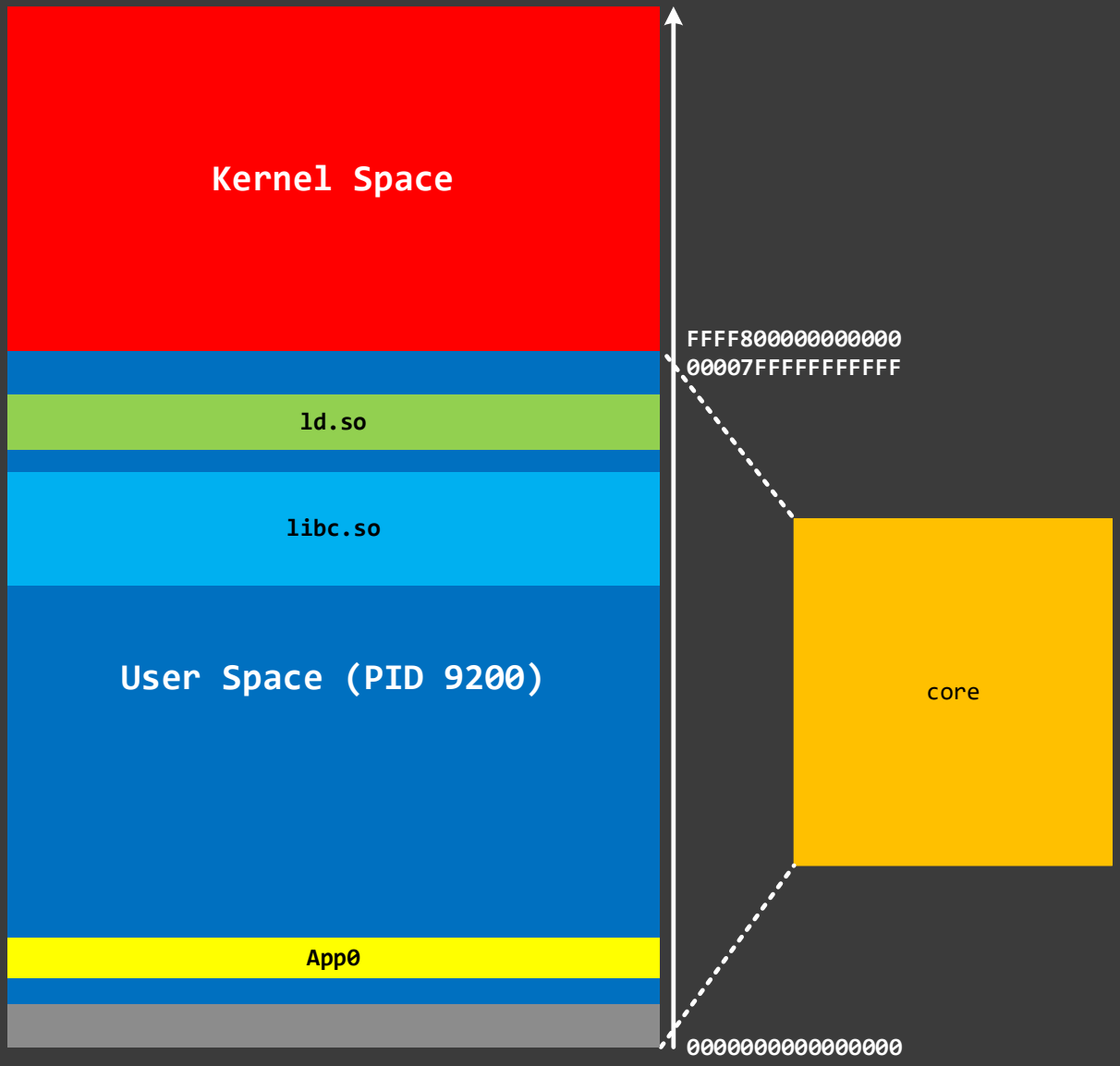
# Memory/Kernel/User Space



# App/Process/Library



# Process Memory Dump




## GDB Commands

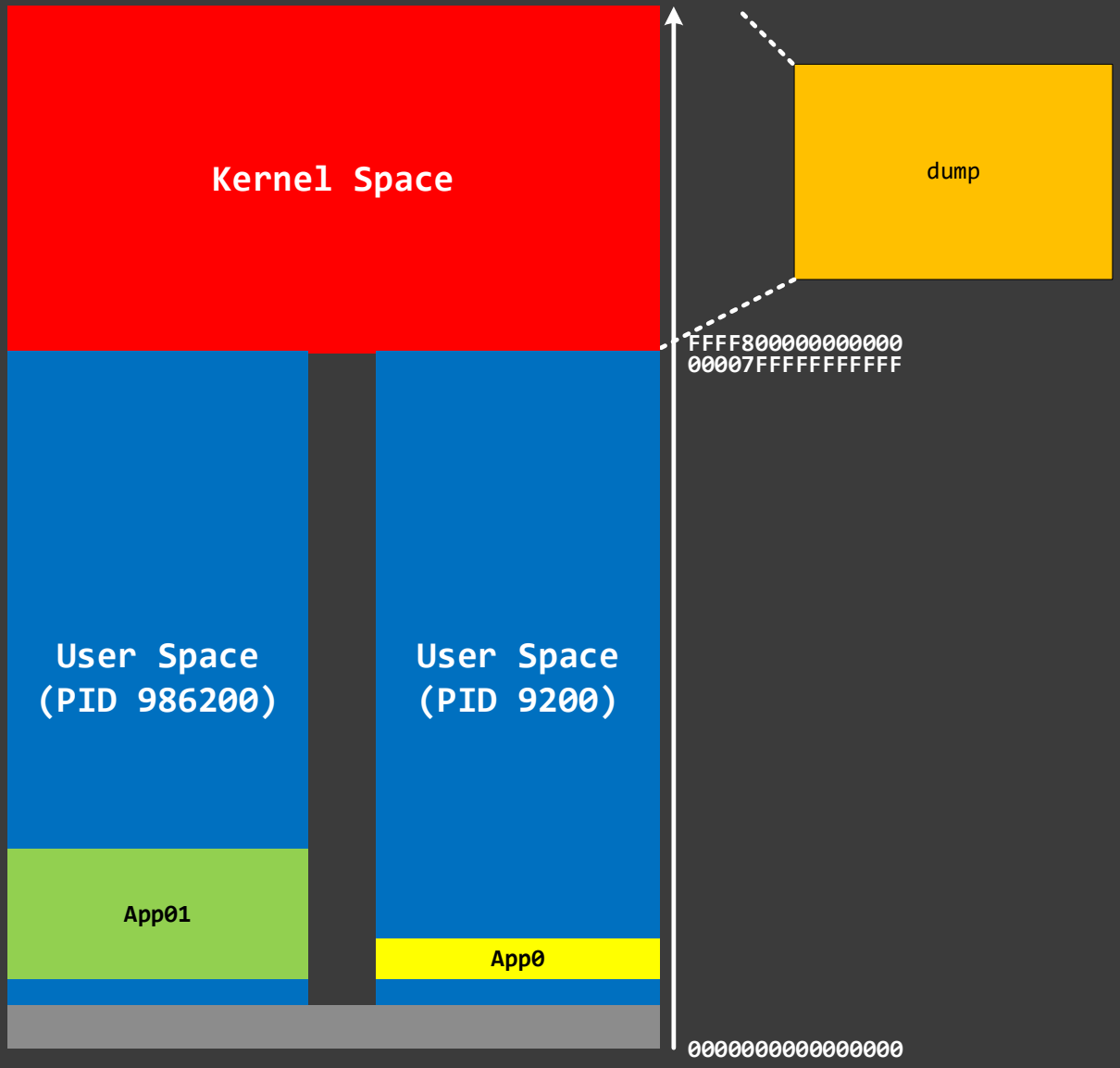
**info sharedlibrary**
Lists dynamic libraries

**maintenance info sections**
Lists memory regions

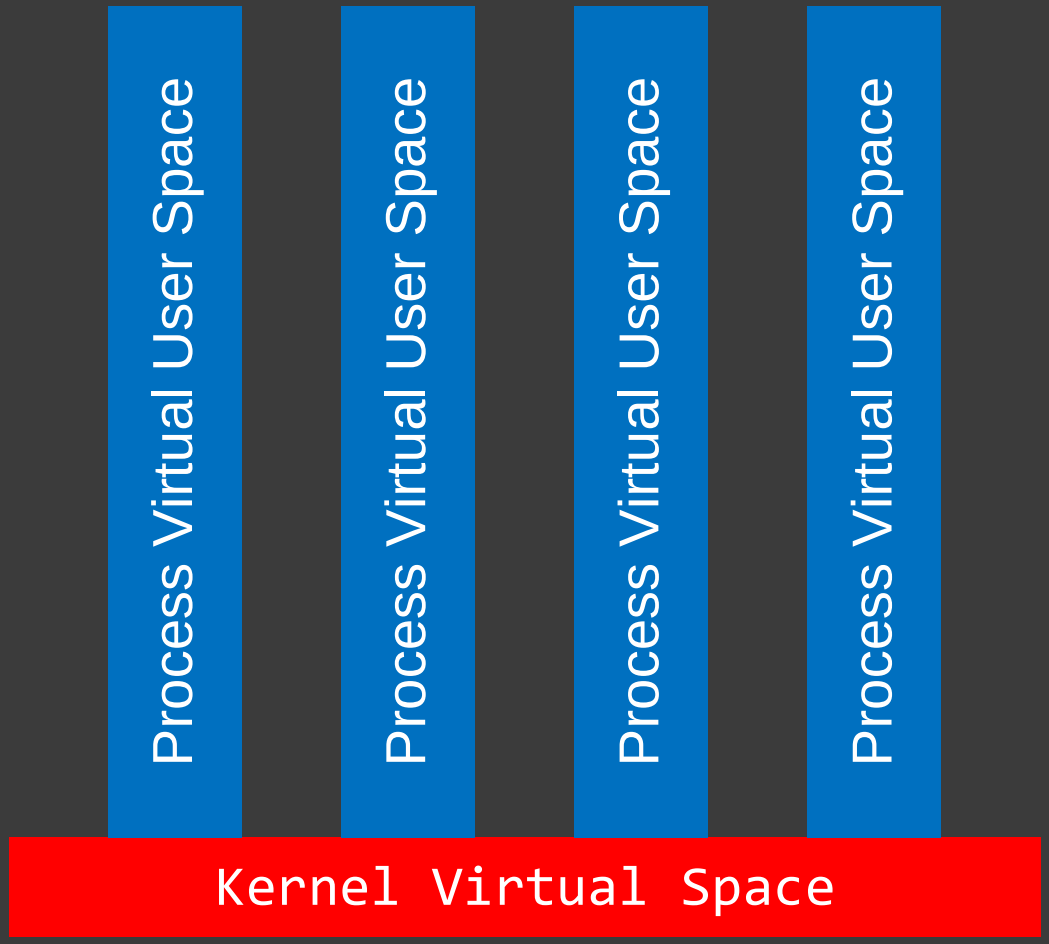
## WinDbg Commands

**!address**
Lists memory regions

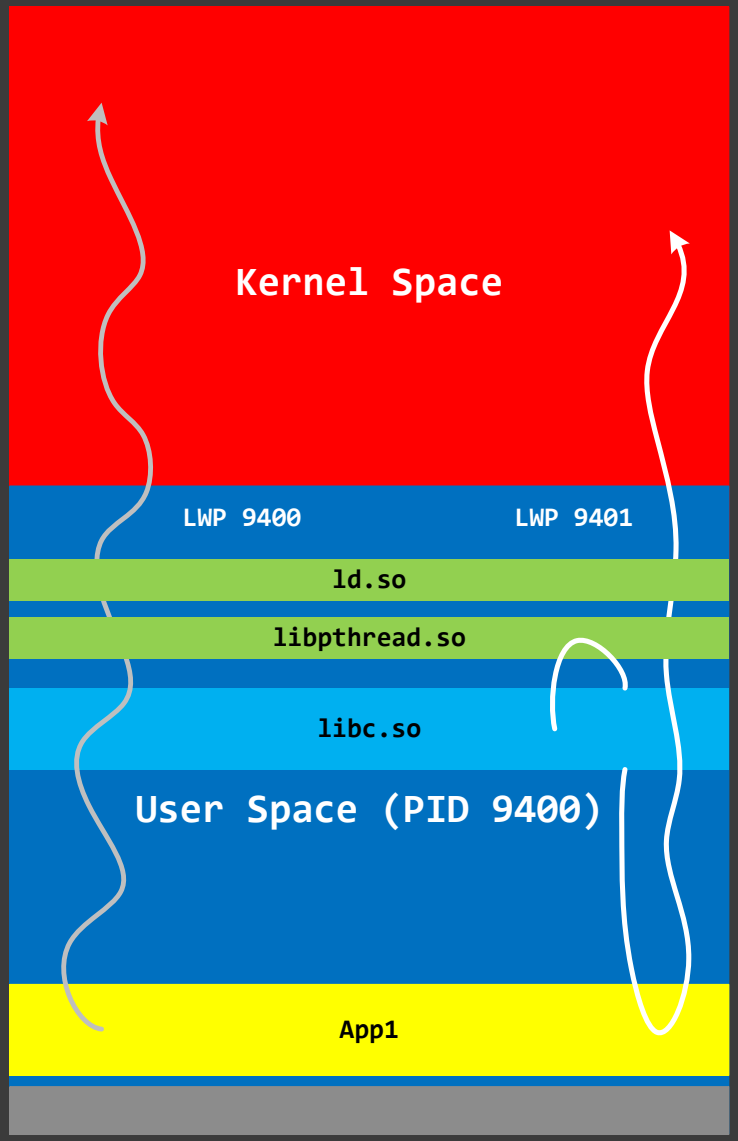
# Kernel Memory Dump



# Fiber Bindle Memory Dump



# Lightweight Processes (Threads)




## GDB Commands

**info threads**
Lists threads

**thread <n>**
Switches between threads

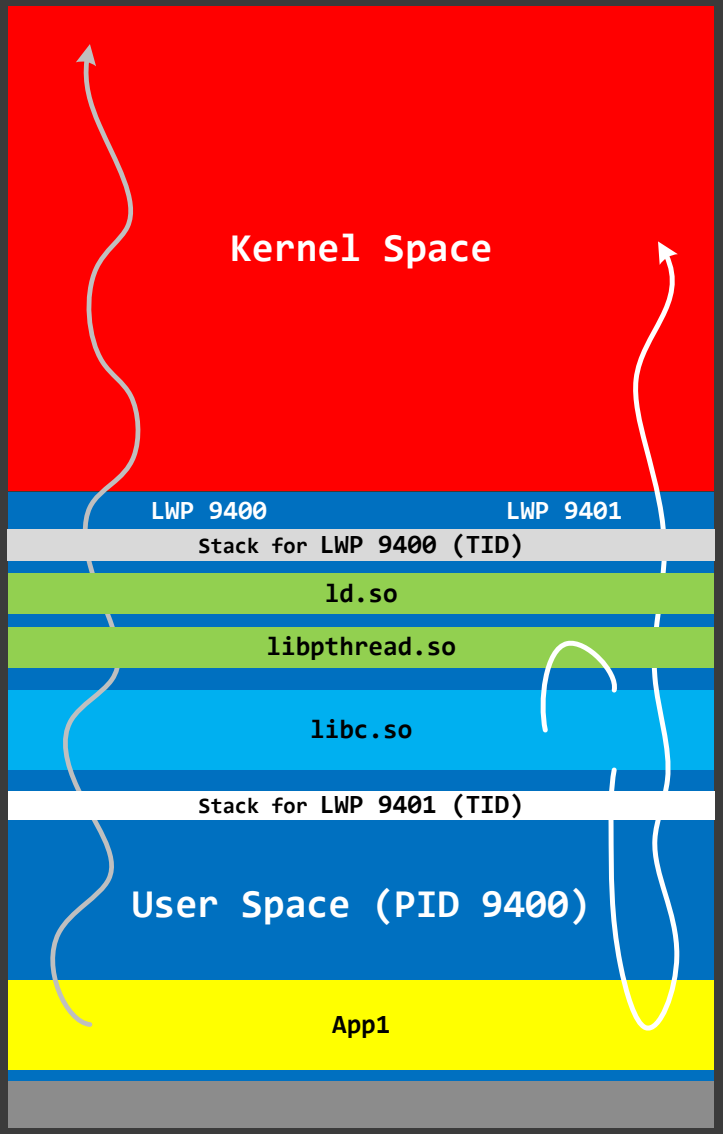
**thread apply all bt**
Lists stack traces from all threads

## WinDbg Commands

**~*k**
Lists stack traces from all threads

**~<n>s**
Switches between threads

# Thread Stack Raw Data



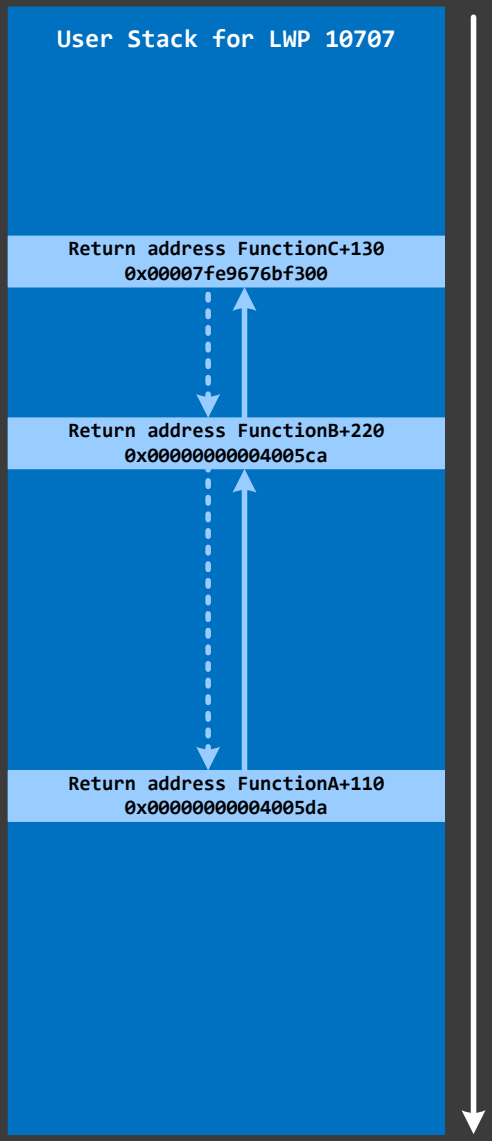

## GDB Commands

**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

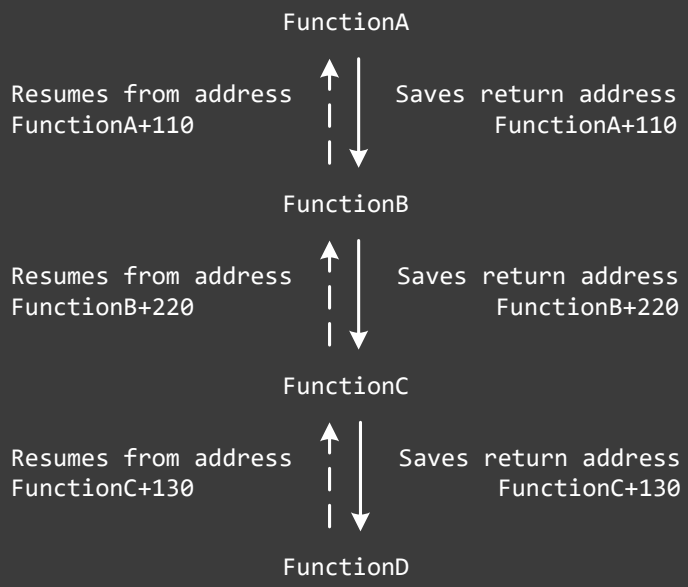
# Thread Stack Trace




```
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 ()
```

# GDB vs. WinDbg

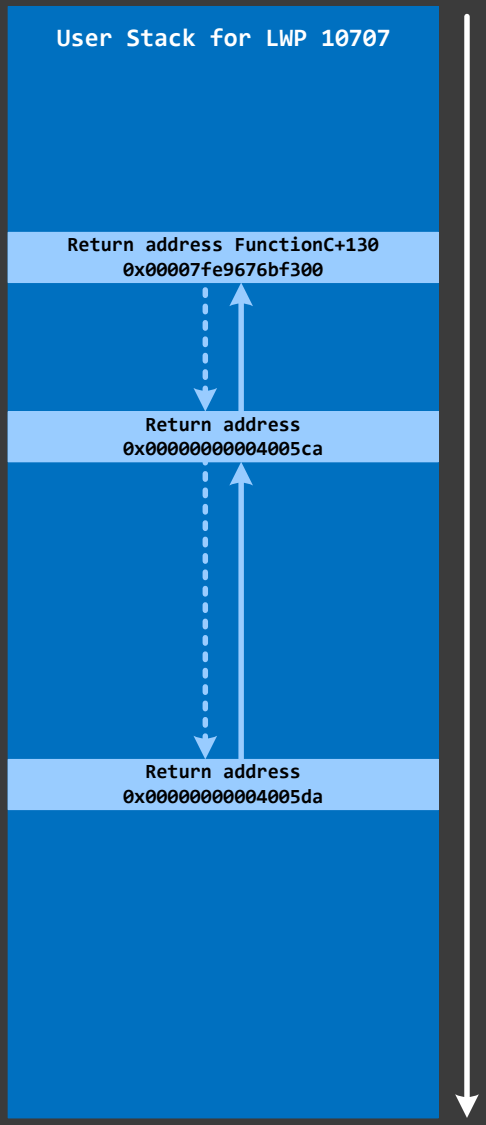
## GDB Commands

```
(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
```

# Thread Stack Trace (no symbols)



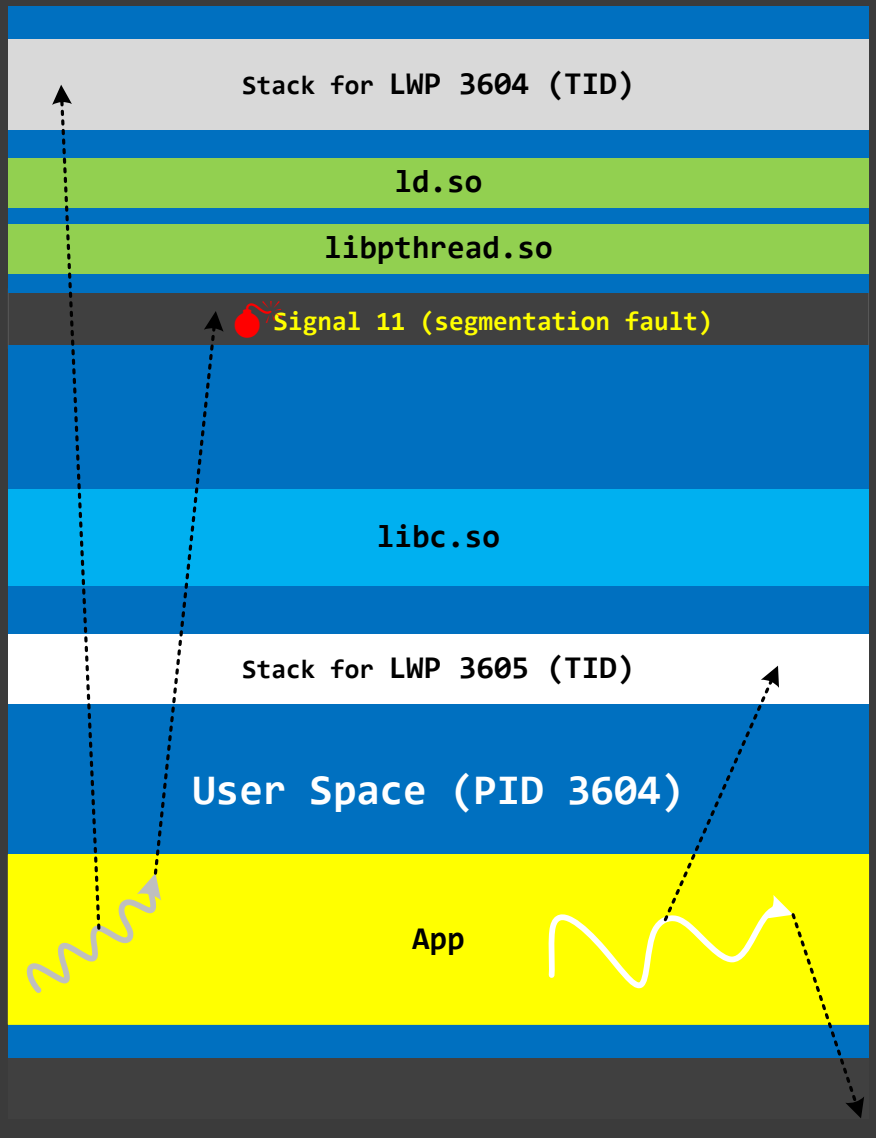

**Symbol file App.sym**

**FunctionA 22000 - 23000**
**FunctionB 32000 - 33000**

## GDB Commands

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

# Exceptions (Access Violation)



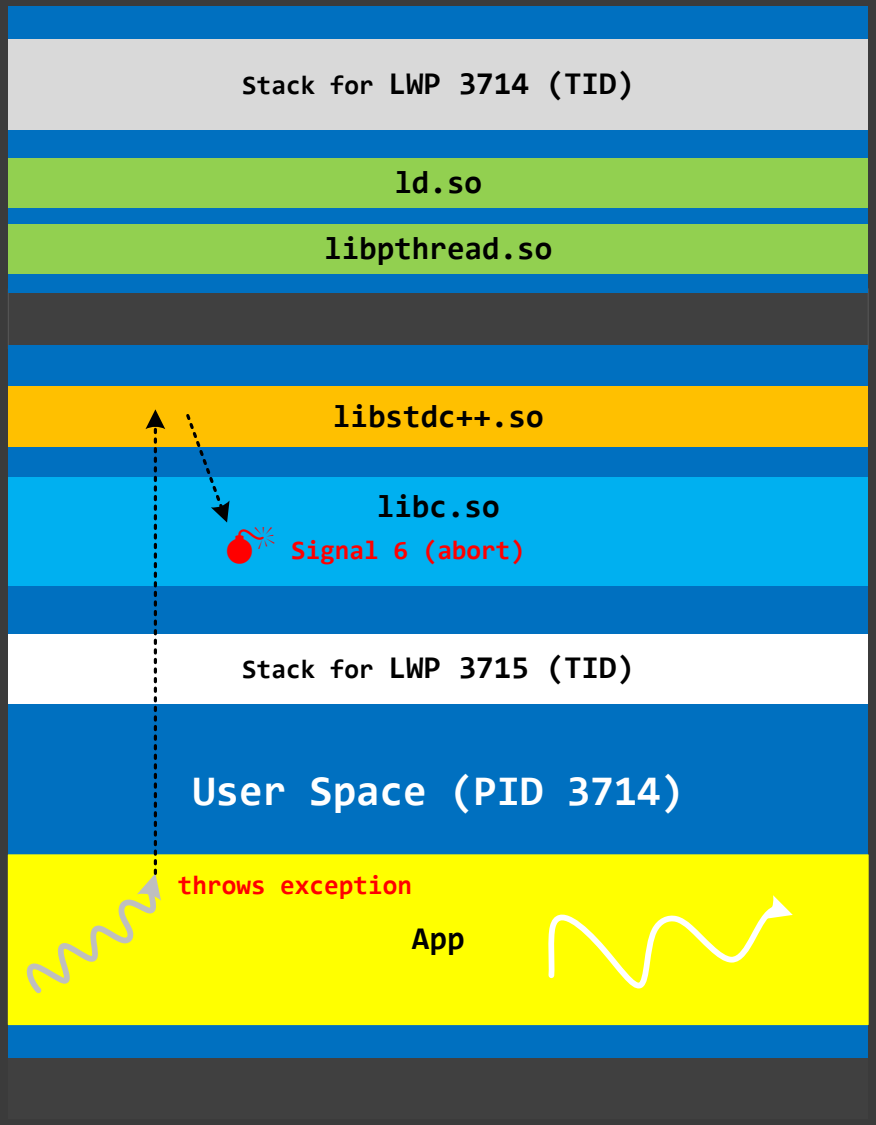

## GDB Commands

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

## WinDbg Commands

```
dp <address> L1
<address>  ????????`????????
```

# Exceptions (Runtime)



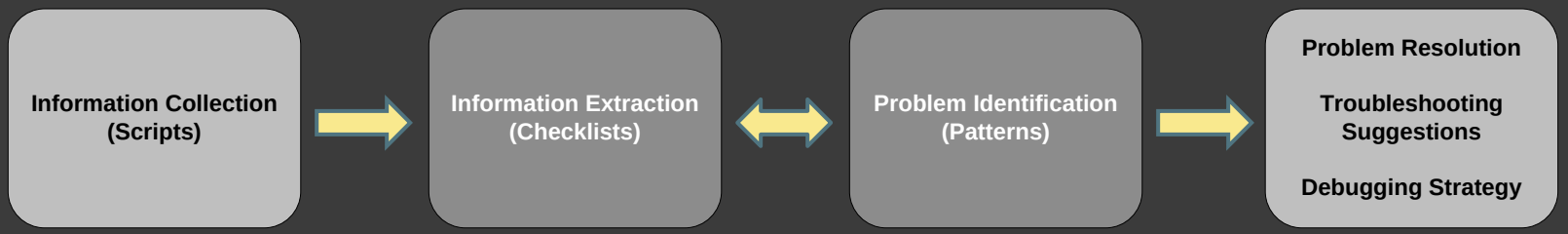
# Pattern-Oriented Diagnostic Analysis

**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, ...



# Part 2: Core Dump Collection

# 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
```

# 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

# Finding Core Dumps (Processes)

- Check the current core dump directory and naming pattern

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

- Further information

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

# Enabling Collection (Kernel)

- Uncompressed kernel image with symbols:

```
$ sudo apt install $(uname -r)-dbg
$ sudo apt install linux-headers-$(uname -r)
```

- Kdump (and kexec):

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

# Generation Methods (Kernel)

- Manual

```
$ sudo echo 1 > /proc/sys/kernel/sysrq
$ sudo echo c > /proc/sysrq-trigger
```

- Kernel modules

# Finding Core Dumps (Kernel)

- Core dumps

```
/var/crash
```

- vmlinux

```
/usr/lib/debug
```

# 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 bison
$ cd crash
$ make
$ sudo make install
```

# Part 3: Practice Exercises

# Links

- Memory Dumps:

Included in Exercise 0

- Exercise Transcripts:

Included in this book

# Exercise 0

- **Goal:** Install GDB and check if GDB loads a core dump correctly
- **Goal:** Install WinDbg Preview or Debugging Tools for Windows, or pull Docker image, and check that symbols are set up correctly
- **Patterns:** Incorrect Stack Trace
- \ALCDA-Dumps\Exercise-A0-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A0-A64-WinDbg.pdf

# Process Core Dumps

Exercises A1 – A12

# Exercise A1

- **Goal:** Learn how to list stack traces, disassemble functions, check their correctness, dump data, get environment
- **Patterns:** Manual Dump (Process); 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-WinDbg.pdf

# Exercise A2D

- **Goal:** Learn how to identify exceptions, find problem threads and CPU instructions
- **Patterns:** NULL Pointer (Data); Active Thread (GDB)
- \ALCDA-Dumps\Exercise-A2D-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A2D-A64-WinDbg.pdf

# Exercise A2C

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

# Exercise A2S

- **Goal:** Learn how to use external debugging information
- \ALCDA-Dumps\Exercise-A2S-x64-GDB.pdf

# Exercise A3

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

# Exercise A4

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

# Exercise A5

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

# 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-WinDbg.pdf

# Exercise A7

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

# Exercise A8

- **Goal:** Learn how to identify runtime exceptions, past execution residue and stack traces, identify handled exceptions
- **Patterns:** C++ Exception; Execution Residue (User Space); Past Stack Trace; Coincidental Symbolic Information; Handled Exception (User Space)
- \ALCDA-Dumps\Exercise-A8-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A8-A64-WinDbg.pdf

# Exercise A9

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

# 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-WinDbg.pdf

# 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-WinDbg.pdf

# Exercise A12

- **Goal:** Learn how to dump memory for post-processing, get the list of functions and module variables, load symbols, inspect arguments and local variables
- **Patterns:** Module Variable
- \ALCDA-Dumps\Exercise-A12-x64-GDB.pdf
- \ALCDA-Dumps\Exercise-A12-A64-WinDbg.pdf

# Kernel Core Dumps

Exercises K1 – K5

# 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

# Exercise K2

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

# Exercise K3

- **Goal:** Learn how to recognize problems with kernel threads, identify their owner module, and follow call chains
- **Patterns:** Origin Module; NULL Pointer (Code); Hidden Call
- \ALCDA-Dumps\Exercise-K3-x64-GDB.pdf

# Exercise K4

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

# Exercise K5

- **Goal:** Learn how to identify kernel stack overflow and kernel stack boundaries
- **Patterns:** Stack Overflow (Kernel Mode)
- \ALCDA-Dumps\Exercise-K5-x64-GDB.pdf

# Follow-up Courses

Advanced
Linux Core Dump Analysis
with Data Structures

Accelerated Linux Debugging[4]

# 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

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)
Missing Frame
Disassembly Hole
Deadlock (Mutex Objects, Kernel Space)
Origin Module
Stack Trace Collection (CPUs)
Stack Overflow (Kernel Mode)

# Resources

- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- Rosetta Stone for Debuggers
- Accelerated Mac OS X Core Dump Analysis (also covers LLDB)
- GDB Pocket Reference
- Encyclopedia of Crash Dump Analysis Patterns, Third Edition
- Practical Foundations of Linux Debugging, Disassembling, Reversing
- Practical Foundations of ARM64 Linux Debugging, Disassembling, Reversing
- Memory Dump Analysis Anthology (some articles in volumes 1, 7, 9A cover GDB)

# Q&A

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

Thank you for attendance!