Prerequisites

Basic Mac OS X troubleshooting

GDB Commands

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

LLDB Commands

We use these boxes to introduce LLDB commands used in practice exercises
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 (30 minutes)
- Core dump collection methods (10 minutes)
- Basic Core Memory Dumps (1 hour 20 minutes)

Day 2
- Core Memory Dumps (2 hours)
Part 1: Fundamentals
Memory/Kernel/User Space

Kernel Space

User Space

NULL Pointers

0000000000000000
00007FFFFFFF
00000000FFFFFFFF
0000000100000000
App/Process/Library

Kernel Space

Cocoa
libobjc.A.dylib

User Space (PID 3743)

libobjc.A.dylib
Cocoa

TextEdit
Process Memory Dump

GDB Commands

- **info sharedlibrary**
  Lists dynamic libraries

- **maintenance info sections**
  Lists memory regions

LLDB Commands

- **image list**
  Lists dynamic libraries

- **image dump sections**
  Lists memory regions

Diagram:

- **Kernel Space**
- **User Space (PID 3743)**
  - Cocoa
  - `libobjc.A.dylib`

Regions:

- **core.3743**
  - 0000000000000000
  - 00007FFFFFFF

- **TextEdit**
  - 0000001000000000
**Process Threads**

- **Kernel Space**
  - TID 0
  - TID 1
  - `libsystem_kernel.dylib`

- **User Space (PID 362)**
  - `AppA`
  - `libsystem_c.dylib`

**GDB Commands**

- `info threads`
  - Lists threads

- `thread <n>`
  - Switches between threads

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

**LLDB Commands**

- `thread list`
  - Lists threads

- `thread select <n>`
  - Switches between threads

- `thread backtrace all`
  - Lists stack traces from all threads

© 2014 Software Diagnostics Services
Thread Stack Raw Data

GDB Commands

\texttt{x/<n>a <address>}

Prints \( n \) addresses with corresponding symbol mappings if any

LLDB Commands

\texttt{x/<n>a <address>}

Prints \( n \) addresses with corresponding symbol mappings if any

Kernel Space

\text{User Space (PID 362)}

libsystem_kernel.dylib

Stack for TID 0

Stack for TID 1

libsystem_c.dylib

AppA
Thread Stack Trace

**GDB Commands**

```
(gdb) bt
#0 0x00007fff885e982a in FunctionD ()
#1 0x00007fff83288a9c in FunctionC ()
#2 0x0000000104da3ea9 in FunctionB ()
#3 0x0000000104da3edb in FunctionA ()
```

**LLDB Commands**

```
(lldb) bt
frame #0: 0x00007fff885e982a Module`FunctionD + offset
frame #1: 0x00007fff83288a9c Module`FunctionC + 130
frame #2: 0x0000000104da3ea9 AppA`FunctionB + 220
frame #3: 0x0000000104da3edb AppA`FunctionA + 110
```
# GDB and LLDB vs. WinDbg

## GDB Commands

<table>
<thead>
<tr>
<th>(gdb) bt</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0 0x00007fffffff885e982a in FunctionD ()</td>
</tr>
<tr>
<td>#1 0x00007fffffff83288a9c in FunctionC ()</td>
</tr>
<tr>
<td>#2 0x0000000104da3ea9 in FunctionB ()</td>
</tr>
<tr>
<td>#3 0x0000000104da3edb in FunctionA ()</td>
</tr>
</tbody>
</table>

## LLDB Commands

<table>
<thead>
<tr>
<th>(lldb) bt</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame #0: 0x00007fffffff885e982a Module`FunctionD + offset</td>
</tr>
<tr>
<td>frame #1: 0x00007fffffff83288a9c Module`FunctionC + 130</td>
</tr>
<tr>
<td>frame #2: 0x0000000104da3ea9 AppA`FunctionB + 220</td>
</tr>
<tr>
<td>frame #3: 0x0000000104da3edb AppA`FunctionA + 110</td>
</tr>
</tbody>
</table>

## WinDbg Commands

<table>
<thead>
<tr>
<th>0:000&gt; kn</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00007fffffff83288a9c Module!FunctionD+offset</td>
</tr>
<tr>
<td>01 0000000104da3ea9 Module!FunctionC+130</td>
</tr>
<tr>
<td>02 0000000104da3edb AppA!FunctionB+220</td>
</tr>
<tr>
<td>03 0000000000000000 AppA!FunctionA+110</td>
</tr>
</tbody>
</table>
Thread Stack Trace (no dSYM)

GDB Commands

(gdb) bt
#0 0x00007fff885e982a in FunctionD ()
#1 0x00007fff83288a9c in FunctionC ()
#2 0x0000000104da3ea9 in ?? ()
#3 0x0000000104da3edb in ?? ()

LLDB Commands

(lldb) bt
frame #0: 0x00007fff885e982a Module\FunctionD + offset
frame #1: 0x00007fff83288a9c Module\FunctionC + 130
frame #2: 0x0000000104da3ea9 AppA\_lldb_unnamed_function1$$AppA + 220
frame #3: 0x0000000104da3edb AppA\_lldb_unnamed_function2$$AppA + 110
Exceptions (Access Violation)

GDB Commands

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

LLDB Commands

(lldb) x <address>
error: core file does not contain 0x<address>

© 2014 Software Diagnostics Services
Exceptions (Runtime)

libsystem_kernel.dylib

throws exception libA.dylib

TID 0

Stack for TID 0

User Space (PID 306)

Stack for TID 1

AppA
Pattern-Driven Analysis

**Pattern**: a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.
Part 2: Core Dump Collection
Enabling Collection

- Temporary for the current terminal session:
  
  $ ulimit -c unlimited

- Permanent for every user:
  
  $ sudo vi /etc/launchd.conf

  Add the line: `limit core unlimited`
Generation Methods

- **Command line:**
  
  ```
  $ kill -s SIGQUIT PID  
  $ kill -s SIGABRT PID  
  ```

- **GUI:**
  
  Utilities \ Activity Monitor  
  View \ Send Signal to Process
Part 3: Practice Exercises
Links

- Memory Dumps:
  NOT IN THE PUBLIC PREVIEW VERSION

- Exercise Transcripts:
  NOT IN THE PUBLIC PREVIEW VERSION
Exercise 0 (GDB)

- **Goal**: Install Xcode and check if GDB loads a core dump correctly

- **Patterns**: Incorrect Stack Trace

- AMCDA-Dumps\Exercise-A0-GDB.pdf

© 2014 Software Diagnostics Services
Exercise 0 (LLDB)

- **Goal:** Install Xcode and check if LLDB loads a core dump correctly

- **Patterns:** Incorrect Stack Trace

- \AMCDA-Dumps\Exercise-A0-LLDB.pdf
Process Core Dumps

Exercises A1-A12
Exercise A1 (GDB)

- **Goal:** Learn how to list stack traces, disassemble functions, check their correctness, dump data, compare core dumps with diagnostic reports, get environment

- **Patterns:** Manual Dump, Stack Trace, Stack Trace Collection, Annotated Disassembly, Paratext, Not My Version, Environment Hint

- \AMCDA-Dumps\Exercise-A1-GDB.pdf
Exercise A1 (LLDB)

- **Goal:** Learn how to list stack traces, disassemble functions, check their correctness, dump data, compare core dumps with diagnostic reports, get environment

- **Patterns:** Manual Dump, Stack Trace, Stack Trace Collection, Annotated Disassembly, Paratext, Not My Version, Environment Hint

- `\AMCDA-Dumps\Exercise-A1-LLDB.pdf`
Exercise A2 (GDB)

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

- **Patterns:** Multiple Exceptions, NULL Pointer (data), NULL Pointer (code)

- \\AMCDA-Dumps\Exercise-A2-GDB.pdf
Exercise A2 (LLDB)

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

- **Patterns:** Multiple Exceptions, NULL Pointer (data), NULL Pointer (code)

- \ AMCDA-Dumps\Exercise-A2-LLDB.pdf

© 2014 Software Diagnostics Services
Exercise A3 (GDB)

- **Goal:** Learn how to identify spiking threads

- **Patterns:** Spiking Thread

- \AMCDA-Dumps\Exercise-A3-GDB.pdf
Exercise A3 (LLDB)

- **Goal:** Learn how to identify spiking threads
- **Patterns:** Spiking Thread
- \AMCDA-Dumps\Exercise-A3-LLDB.pdf
Exercise A4 (GDB)

- **Goal:** Learn how to identify heap regions and heap corruption
- **Patterns:** Heap Corruption
- \AMCDA-Dumps\Exercise-A4-GDB.pdf
Exercise A4 (LLDB)

- **Goal:** Learn how to identify heap regions and heap corruption
- **Patterns:** Heap Corruption
- \AMCDA-Dumps\Exercise-A4-LLDB.pdf
Exercise A5 (GDB)

- **Goal:** Learn how to identify stack corruption

- **Patterns:** Local Buffer Overflow, Execution Residue

- \AMCDA-Dumps\Exercise-A5-GDB.pdf
Exercise A5 (LLDB)

- **Goal**: Learn how to identify stack corruption
- **Patterns**: Local Buffer Overflow, Execution Residue
- `\AMCDA-Dumps\Exercise-A5-LLDB.pdf`
Exercise A6 (GDB)

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

- **Patterns:** Stack Overflow, Execution Residue

- \AMCDA-Dumps\Exercise-A6-GDB.pdf
Exercise A6 (LLDB)

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

- **Patterns:** Stack Overflow, Execution Residue

- \\AMCDA-Dumps\\Exercise-A6-LLDB.pdf
Exercise A7 (GDB)

- **Goal:** Learn how to identify active threads
- **Patterns:** Divide by Zero, Active Thread
- \AMCDA-Dumps\Exercise-A7-GDB.pdf
Exercise A7 (LLDB)

- **Goal:** Learn how to identify active threads
- **Patterns:** Divide by Zero, Active Thread

\AMCDA-Dumps\Exercise-A7-LLDB.pdf

© 2014 Software Diagnostics Services
Exercise A8 (GDB)

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

- **Patterns:** C++ Exception, Execution Residue, Coincidental Symbolic Information, Handled Exception

- \AMCDA-Dumps\Exercise-A8-GDB.pdf
Exercise A8 (LLDB)

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

- **Patterns:** C++ Exception, Execution Residue, Coincidental Symbolic Information, Handled Exception

- \AMCDA-Dumps\Exercise-A8-LLDB.pdf
Exercise A9 (GDB)

- **Goal:** Learn how to identify heap leaks
- **Patterns:** Heap Leak, Execution Residue, Module Hint
- \AMCDA-Dumps\Exercise-A9-GDB.pdf
Exercise A9 (LLDB)

- **Goal:** Learn how to identify heap leaks
- **Patterns:** Heap Leak, Execution Residue, Module Hint
- \AMCDA-Dumps\Exercise-A9-LLDB.pdf
Exercise A10 (GDB)

- **Goal:** Learn how to identify heap contention wait chains, synchronization issues, advanced disassembly, dump arrays

- **Patterns:** Double Free, Heap Contention, Wait Chain, Critical Region, Self-Diagnosis

- \AMCDA-Dumps\Exercise-A10-GDB.pdf
Exercise A10 (LLDB)

- **Goal:** Learn how to identify heap contention wait chains, synchronization issues, advanced disassembly, dump arrays

- **Patterns:** Double Free, Heap Contention, Wait Chain, Critical Region, Self-Diagnosis

- AMCDA-Dumps\Exercise-A10-LLDB.pdf
Exercise A11 (GDB)

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

- **Patterns:** Wait Chains, Deadlock, Execution Residue, Handled Exception

- \AMCDA-Dumps\Exercise-A11-GDB.pdf
Exercise A11 (LLDB)

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

- **Patterns:** Wait Chains, Deadlock, Execution Residue, Handled Exception

- \`\`AMCDA-Dumps\`\`Exercise-A11-LLDB.pdf
Exercise A12 (GDB)

- **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

- \AMCDA-Dumps\Exercise-A12-GDB.pdf
Exercise A12 (LLDB)

- **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

- \\AMCDA-Dumps\Exercise-A12-LLDB.pdf
Pattern Links

Active Thread  Annotated Disassembly
C++ Exception  Coincidental Symbolic Information
Critical Region  Deadlock
Divide by Zero  Double Free
Environment Hint  Execution Residue
Incorrect Stack Trace  Handled Exception
Heap Contention  Heap Corruption
Heap Leak  Local Buffer Overflow
Manual Dump  Module Hint
Module Variable  Multiple Exceptions
Not My Version  NULL Pointer (data)
NULL Pointer (code)  Paratext
Self-Diagnosis  Spiking Thread
Stack Overflow  Stack Trace
Stack Trace Collection  Wait Chain
Resources

- Software Diagnostics Institute
- Pattern-Driven Software Diagnostics
- Pattern-Based Software Diagnostics
- Debugging TV
- Rosetta Stone for Debuggers
- GDB Pocket Reference
- GDB -> LLDB Map
- Memory Dump Analysis Anthology (volume 7 covers Mac OS X)
Q&A

Please send your feedback using the contact form on PatternDiagnostics.com
Thank you for attendance!