Reversing Disassembly Reconstruction

Accelerated

Revised Version

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

- Working C or classic C++ knowledge
- Basic assembly language knowledge
Audience

- Novices
  Learn x64 assembly language

- Experts
  Learn the new pattern approach
Pattern-Oriented RDR

- Complex crashes and hangs (victimware analysis)
- Malware analysis
- Studying new products
Training Goals

- Review fundamentals
- Learn patterns and techniques
Training Principles

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

- Implicit memory leak resulted from wrong API call parameter

- Debugging.TV episode 0x31
Part 1: Theory
Computation

Data → CPU → Code

Memory Changes
Disassembly

Data/Code numbers

488d0d2cce0000  lea rcx,[CPUx64+0xe2f8 (00000001`3f85e2f8)]  ; "Hello World!"

Annotated Disassembly memory analysis pattern
The Problem of Reversing

- Compilation to $\text{Machine Language}_M$

$\text{Language}_1 \rightarrow \text{Language}_M \leftrightarrow \text{Language}_2$

- Decompilation

$\text{Language}_M \rightarrow ?$
The Solution to Reversing

- Memory Language_M Semantics

Language_1 → Language_M → Language_2

- Decompilation

Understanding of Language_M
The Reversing Tool

Memory Cell Diagrams

RSP
8
10
18
20
28
30
38
40
48
50

© 2020 Software Diagnostics Services
Re(De)construction

- Time dimension: sequence diagrams
- Space dimension: component diagrams

How does it work temporally and structurally?
ADDR Patterns

- Accelerated
- Disassembly patterns
- De(Re)construction patterns
- Reversing patterns
ADDR Patterns (II)

- Accelerated
- Disassembly patterns
- Decompilation patterns
- Reconstruction patterns
ADDR Schemas

- Function Prologue -> Function Epilogue
- Call Prologue -> Function Call -> Call Epilogue
- Potential Functionality -> Call Skeleton -> Call Path
- Call Parameter -> Function Parameter -> Local Variable
ADDR Implementations

ADDR Pattern Catalogue

Windows

Mac OS X

Linux

x86

x64

ARM

© 2020 Software Diagnostics Services
Pattern Catalogues

- Elementary Software Diagnostics Patterns
- Memory Analysis Patterns
- Trace and Log Analysis Patterns
- Unified Debugging Patterns
- ADDR Patterns
Pattern Orientation

- Pattern-Driven ADDR
- Pattern-Based ADDR
Part 2: Practice Exercises
Links

- Memory dumps:
  NOT IN THE PUBLIC PREVIEW VERSION

- Exercise Transcripts:
  NOT IN THE PUBLIC PREVIEW VERSION
Exercise 0

- **Goal:** Install Debugging Tools for Windows and WinDbg Preview and check that symbols are setup correctly

- \ADDRIExercise-0-Download-Setup-WinDbg.pdf
Main CPU Registers

Illustrated on memory cell diagrams in \ADDR\MCD-R1.xlsx

- 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)
Exercise R1

- **Goal:** Review x64 assembly fundamentals; learn how to reconstruct stack trace manually

- **ADDR Patterns:** Universal Pointer, Symbolic Pointer $S^2$, Interpreted Pointer $S^3$, Context Pyramid

- **Memory Cell Diagrams:** Register, Pointer, Stack Frame

- \ADD\Exercise-R1.pdf

- \ADD\MCD-R1.xlsx
Stack Reconstruction

1. Top frame from the current RIP₁, RSP₁ (r)
2. Disassemble around the current RIPₙ (u[f] RIPₙ)
3. Find out the beginning of the function prologue
4. Check RSPₙ usage (sub, push) and count offsets
5. Get RIPₙ₊₁ for the next frame (dps @rspₙ + offset)
6. Get RSPₙ₊₁ for the next frame (RSPₙ₊₁ₙ+8)
7. ++ₙ
8. goto #2
Exercise R2

- **Goal:** Learn how to map source code to disassembly

- **ADDR Patterns:** Potential Functionality, Function Skeleton, Function Call, Call Path, Local Variable, Static Variable, Pointer Dereference

- **Memory Cell Diagrams:** Pointer Dereference

- `\ADDR\Exercise-R2.pdf`

- `\ADDR\MCD-R2.xlsx`
Exercise R3

- **Goal**: Learn a function structure and associated memory operations

- **ADDR Patterns**: Function Prologue, Function Epilogue, Variable Initialization, Memory Copy

- **Memory Cell Diagrams**: Function Prologue, Function Epilogue

- \ADD\Exercise-R3.pdf

- \ADD\MCD-R3.xlsx
Exercise R4

- **Goal:** Learn how to recognize call and function parameters and track their data flow

- **ADDR Patterns:** Call Prologue, Call Parameter, Call Epilogue, Call Result, Control Path, Function Parameter, Structure Field

- `\ADDR\Exercise-R4.pdf`
Exercise R5

- **Goal:** Master memory cell diagrams as an aid to understanding complex disassembly logic

- **ADDR Patterns:** Last Call, Loop, Memory Copy

- **Memory Cell Diagrams:** Memory Copy

- \ADDRAbsolutePath\Exercise-R5.pdf

- \ADDRAbsolutePath\MCD-R5.xlsx
Exercise R6

- **Goal**: Learn how to map code to execution residue and reconstruct past behaviour; recognise previously introduced ADDR patterns in the context of compiled classic C++ code

- **ADDR Patterns**: Separator Frames, Virtual Call

- **Memory Cell Diagrams**: Virtual Call

- \ADDR\Exercise-R6.pdf

- \ADDR\MCD-R6.xlsx
Live Debugging Techniques

- **ADDR Patterns:** Component Dependencies, API Trace, Fibre Bundle (trace analysis pattern)

- Some dependencies can be learnt from crash dump stack traces

- [Debugging.TV](https://www.debugging.tv) / [YouTube](https://www.youtube.com)

- Live debugging training: [Accelerated Windows Debugging](https://www.acceleratedwindowsdebugging.com)
Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org / SoftwareDiagnostics.Institute
- PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- Practical Foundations of Windows Debugging, Disassembling, Reversing
- Memory Dump Analysis Anthology
Q&A

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