Public Preview Version

Reversing Disassembly Reconstruction Accelerated

Dmitry Vostokov
Software Diagnostics Services

Prerequisites

- Working C or 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
- www.Debugging.tv episode 0x31

Schedule Summary

Day 1

- Theory
- Exercise R1

Day 2

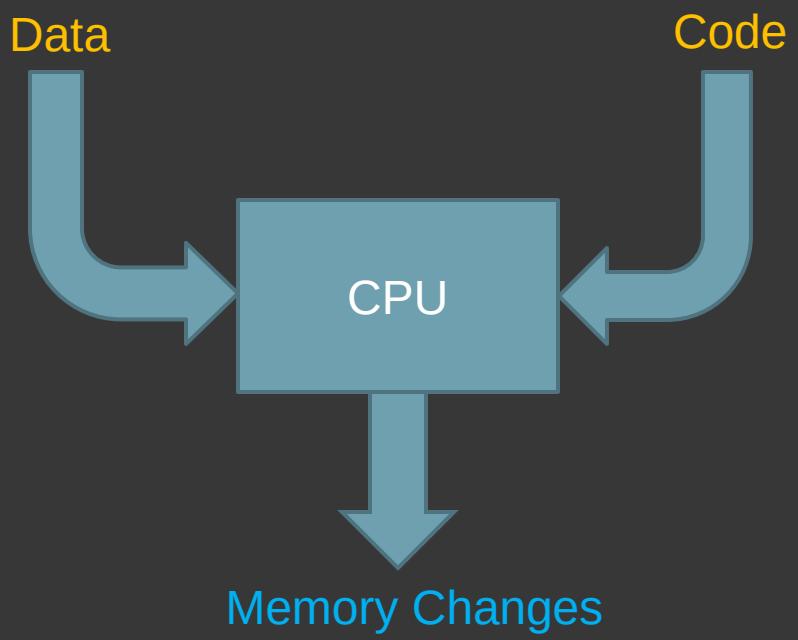
- Exercises R2 – R4

Day 3

- Exercises R5 – R6
- Q&A

Part 1: Theory

Computation



Disassembly

Data/Code numbers

Data/Code symbolic

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

Annotated Disassembly memory analysis pattern

The Problem of Reversing

- Compilation to **M**achine Language$_M$

Language$_1$  Language$_M$  Language$_2$

- Decompilation

Language$_M$  ?

The Solution to Reversing

- **M**emory Language$_M$ Semantics

Language$_1$  Language$_M$  Language$_2$

- Decompilation

Understanding of Language$_M$

The Reversing Tool

RSP				
8				
10				
18				
20				
28				
30				
38				
40				
48				
50				

Memory Cell Diagrams

RAX							

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

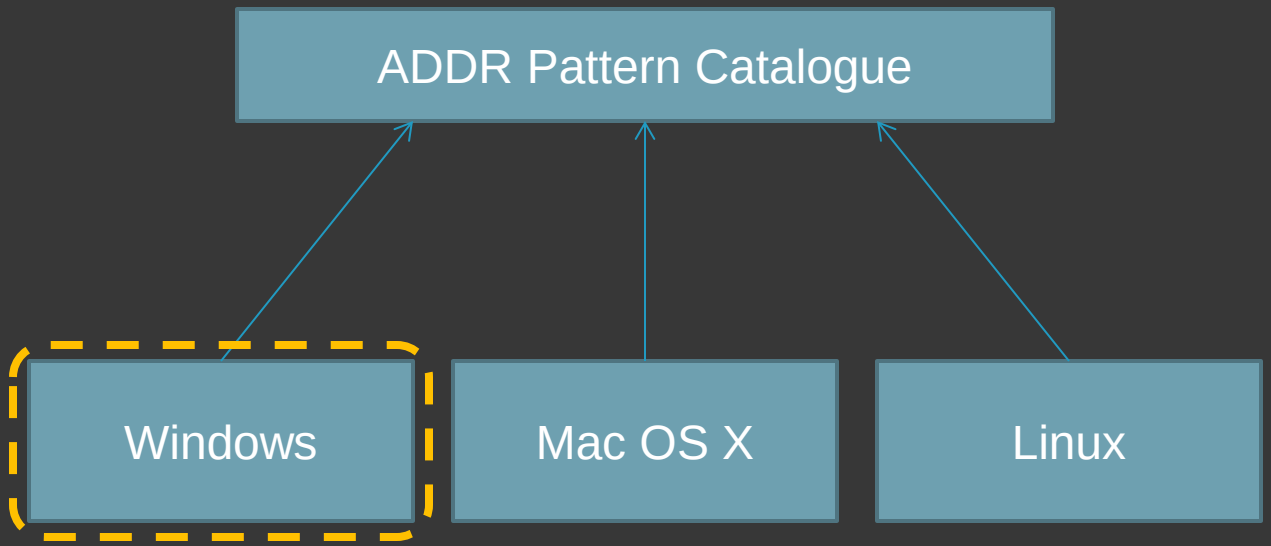
ADDR Patterns (II)

- 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



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 available in preview version

- Exercise Transcripts:

Not available in preview version

Exercise 0

- **Goal:** Install Debugging Tools for Windows and learn how to set up symbols correctly

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

Stack Reconstruction

1. Top frame from the current RIP_1, RSP_1 (**r**)
2. Disassemble around the current RIP_n (**u**[**f**] RIP_n)
3. Find out the beginning of the function prologue
4. Check RSP_n usage (**sub**, **push**) and count offsets
5. Get RIP_{n+1} for the next frame (**dps** @rsp_n + offset
6. Get RSP_{n+1} for the next frame (RSP_n+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

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

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

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

Exercise R6

- **Goal:** Learn how to to map code to execution residue and reconstruct past behaviour; recognise previously introduced ADDR patterns in the context of compiled C++ code
- **ADDR Patterns:** Separator Frames, Virtual Call
- **Memory Cell Diagrams:** Virtual Call

Live Debugging Techniques

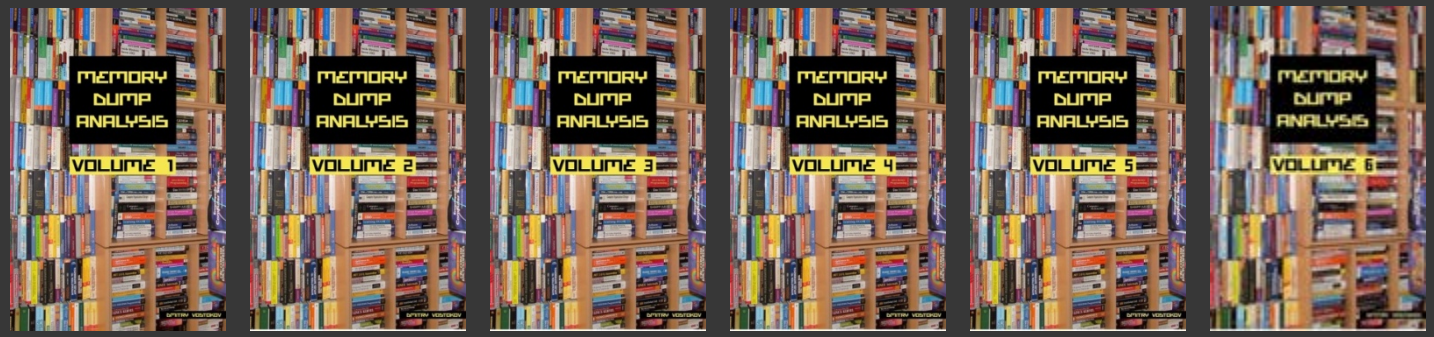
- **ADDR Patterns:** Component Dependencies, API Trace, Fibre Bundle (trace analysis pattern)
- Some dependencies can be learnt from crash dump stack traces (example)
- Debugging.TV / YouTube
- Live debugging training: Accelerated Windows Debugging[3]

Custom Tracing

- WinDbg logging extension
- www.Debugging.tv episode 0x8 YouTube

Resources

- WinDbg Help / WinDbg.org (quick links)
- DumpAnalysis.org
- Debugging.TV / DebuggingTV YouTube Channel
- Windows Debugging: Practical Foundations
- x64 Windows Debugging: Practical Foundations
- Software Diagnostics Library
- Memory Dump Analysis Anthology



x86 disassembly/reversing:
Hackers Disassembly Uncovered by Kris Kaspersky

Q&A

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

Thank you for attendance!