Windows API for Software Diagnostics
Accelerated
With Category Theory in View

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

- Development experience

and (optional)

- Basic memory dump analysis
Training Goals

- Review fundamentals of Windows API
- Learn diagnostic analysis techniques
- See how Windows API knowledge is used during diagnostics and debugging
Training Principles

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

- General Windows API aspects
- Windows API formalization
- Windows API and languages
- Windows API classes
- Practical exercises
Training Idea

- Cybersecurity
- Memory dump analysis
- Reading Windows-based Code training
General Windows API Aspects

- Header-technology view
- Naming convention
- Basic type system
- Hungarian notation
- Call types
- Export/import functions
- IAT
- Virtual process address space
- Calling convention
- API sequences
- API layers
- Documented/undocumented API
- Exports and imports
- API and system calls
- API source code

- Modules
- API usage
- API internals
- Delay-loaded API
- API sets
- API name patterns
- API namespaces
- API syntagms/paradigms
- Marked API
- ADDR patterns
- DebugWare patterns
- Memory analysis patterns
- Trace and log analysis patterns
- WOW64
- API and errors
Windows API Formalization

- API compositionality
- Category theory language
- A view of category theory
- Category theory square
- API category
- API functor
- API natural transformation
- API adjunction
- Informal n-API
Windows API and Languages

- C#
- API metadata
- Scala Native
- Golang
- Rust
- Python
Windows API Classes

- GUI
- Windowing
- GDI
- GDI+
- Module/Library
- Process/Thread
- Services
- Security
- Process Heap
- Virtual Memory
- IPC
- RPC
- Synchronization
- I/O
- Runtime
- COM
- Networking
- Console
Links

- **Memory Dumps**
  Included in Exercise W0

- **Exercise Transcripts**
  Included in this book
Exercise W0

- **Goal:** Install WinDbg Preview or Debugging Tools for Windows, or pull Docker image, and check that symbols are set up correctly.

- **Memory Analysis Patterns:** Stack Trace; Incorrect Stack Trace

- `\AWAPI-Dumps\Exercise-W0.pdf`
Why Windows API?

- Development
- Malware analysis
- Vulnerability analysis and exploitation
- Reversing
- Diagnostics
- Debugging
- Memory forensics
- Crash and hang analysis
- Secure coding
- Static code analysis
- Trace and log analysis
My History of Windows API

- I started using Windows API in 1990 ([Old CV](#))
- Windows SDK since 1990
- Win16 1990 – 1999
- Win32 since 1995
- Win64 since 2006 ([WindowHistory64](#), earlier than that)
- Windows NT since 1996
- Windows NT/2000/XP DDK and WDK since 2003
- DebugWare (DiagWare) tools and patterns 2004 – 2017
- Daily programming using Windows API 2017 – 2020
- Daily reading of Windows API for dump analysis since 2003
Perspectives of Windows API

- Memory analysis: dumps / live debugging
- Disassembly, reconstruction, reversing
- Trace and log analysis (Procmon)
- Category theory
What Windows API?

- Source code perspective (SDK and/or WDK)
- ABI (Application Binary Interface) perspective
General Windows API Aspects
Header-Technology View

- Programming reference for the Win32 API

- CreateThread is included in:
  - `processthreadsapi.h header` is used by:
    - Remote Desktop Services
    - Security and Identity
    - System Services
Naming Convention

- **Naming conventions**

- Functions, parameters, fields: **PascalCase**, **UpperCamelCase**
  - `GetCurrentThread`
  - `CreateWindowExA` / `CreateWindowExW`

- Types: **SCREAMING SNAKE CASE**
  - `SECURITY_ATTRIBUTES`
Basic Type System

- With a few exceptions (int), basic types are typedef-ed

- `minwindef.h`
  - BOOL, DWORD, LPDWORD, UINT, WPARAM, LPARAM

- `winnt.h`
  - CHAR, PSTR, PCSTR, HANDLE

- `basetsd.h`
  - LONG_PTR, UINT64

- `windows.h`
Hungarian Notation

- Wikipedia reference
- Microsoft reference
- CreateWindowExW / WNDCLASSW

```c
HWND CreateWindowExW(
    [in]           DWORD dwExStyle,
    [in, optional] LPCWSTR lpClassName,
    [in, optional] LPCWSTR lpWindowName,
    [in]           DWORD dwStyle,
    [in]           int       X,
    [in]           int       Y,
    [in]           int       nWidth,
    [in]           int       nHeight,
    [in, optional] HWND hWndParent,
    [in, optional] HMENU hMenu,
    [in, optional] HINSTANCE hInstance,
    [in, optional] LPVOID lpParam
);
```

```c
typedef struct tagWNDCLASSW {
    UINT      style;
    WNDPROC   lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HINSTANCE hInstance;
    HICON     hIcon;
    HCURSOR   hCursor;
    HBRUSH    hbrBackground;
    LPCWSTR   lpszMenuName;
    LPCWSTR   lpszClassName;
} WNDCLASSW, *PWNDCLASSW,
*NPWNDCLASSW, *LPWNDCLASSW;
```
Call Types

- **Direct** (the same module, non-exported functions)

  `USER32!CreateWindowExW:
  ...
  0007ffaf5e0cf20d e812000000 call USER32!CreateWindowInternal (00007ffaf5e0cf224)

- **Indirect**

  - Pointer (memory or register)
    
      - **IAT (Import Address Table)** inter-module call

  `App!wWinMain:
  ...
  00007ff677741101 ff15c1e10000 call qword ptr [App!_imp_CreateWindowExW (00007ff67774f2c8)]

© 2022 Software Diagnostics Services
API as Interface

- Provided by (exported from) some DLL module (may have different file extensions)
- Used by (imported by) EXE or DLL
- Can be functional or object-oriented
Export Directory

PE (App.exe)
- IAT (Import Address Table)
- Import Directory
  - "USER32.dll"
  - "CreateWindowExW"

PE (USER32.dll)
- IAT (Import Address Table)
- Export Directory
  - "CreateWindowExW": location
  - ...

CreateWindowExW:
  ...

© 2022 Software Diagnostics Services
IAT (Import Address Table)

PE (App.exe)

IAT (Import Address Table)

_imp_CreateWindowExW: CreateWindowExW

Export Directory

call qword ptr [_imp_CreateWindowExW]

PE (USER32.dll)

IAT (Import Address Table)

Export Directory

CreateWindowExW:

Virtual Process Address Space

PE (App.exe)
IAT (Import Address Table)
_imp_CreateWindowExW: CreateWindowExW
...
call qword ptr [_imp_CreateWindowExW]
...

PE (USER32.dll)
CreateWindowExW:
...

PE (App.exe)
IAT (Import Address Table)
_imp_CreateWindowExW: CreateWindowExW
...
call qword ptr [_imp_CreateWindowExW]
...

PE (USER32.dll)
CreateWindowExW:
...
Calling Convention

- **GetMessageW** (from documentation)

- Actual declaration (WinUser.h)

```c
WINUSERAPI
BOOL
WINAPI
GetMessageW(
    _Out_ LPMSG lpMsg,
    _In_opt_ HWND hWnd,
    _In_ UINT wMsgFilterMin,
    _In_ UINT wMsgFilterMax);
```

- **WINAPI** is defined as **__stdcall** (minwindef.h) vs. default **__cdecl** (C/C++)

- Argument passing order
  - x86: pushed to stack right-to-left, the **callee** cleans the stack (in **__cdecl** – the caller)
  - x64 (**__stdcall** and **__cdecl**): **left-to-right** via RCX, RDX, R8, R9, [RSP+20], [RSP+28], ...
Parameter Passing (x86)

Test8params(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8);

push p8  push p7  push p6  push p5  push p4  push p3  push p2  push p1

lower address


higher address

Parameter Passing (x64)

Test8params(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8);

caller

<table>
<thead>
<tr>
<th>Caller</th>
<th>Callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECX (p1)</td>
<td>ECX (p1)</td>
</tr>
<tr>
<td>EDX (p2)</td>
<td>EDX (p2)</td>
</tr>
<tr>
<td>R8D (p3)</td>
<td>R8D (p3)</td>
</tr>
<tr>
<td>R9D (p4)</td>
<td>R9D (p4)</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>RSP</th>
<th></th>
<th>RSP: return address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0`0</td>
<td></td>
<td>RSP+8: 0`0</td>
</tr>
<tr>
<td>0`0</td>
<td></td>
<td>RSP+10: 0`0</td>
</tr>
<tr>
<td>0`0</td>
<td></td>
<td>RSP+18: 0`0</td>
</tr>
<tr>
<td>0`0</td>
<td></td>
<td>RSP+20: 0`0</td>
</tr>
<tr>
<td>0`p5</td>
<td></td>
<td>RSP+28: 0`p5</td>
</tr>
<tr>
<td>0`p6</td>
<td></td>
<td>RSP+30: 0`p6</td>
</tr>
<tr>
<td>0`p7</td>
<td></td>
<td>RSP+38: 0`p7</td>
</tr>
<tr>
<td>0`p8</td>
<td></td>
<td>RSP+40: 0`p8</td>
</tr>
</tbody>
</table>
```
Exercise W1

- **Goal:** Compare calling conventions on x86 and x64 platforms

- **ADDR Patterns:** Call Prologue; Call Parameter; Function Prologue

- \AWAPI-Dumps\Exercise-W1.pdf
Modules

EXE

DLL A

DLL B

DLL C

WinDbg Commands

0:000> lm
0:000> x /v mpattern!_imp_fpattern
0:000> x /v *!fpattern
0:000> dps module!_imp_name L1
Modules and Analysis Patterns

- **Module memory analysis patterns**
  - Module Collection
  - Coupled Modules
  - Duplicated Module

- **Namespace malware analysis pattern**
Exercise W2

- **Goal:** Explore modules and their dependencies

- **Memory Analysis Patterns:** Module Collection; Coupled Modules

- **Malware Analysis Patterns:** Namespace

- \AWAPI-Dumps\Exercise-W2.pdf
API Usage

- Module usage (static analysis)
  - Hidden Module

- Function usage (dynamic analysis)

WinDbg Commands

```
0:000> .imgscan
0:000> bm mpattern!fpattern
```
Exercise W3

- **Goal:** Find usage of specific Windows API functions

- **Debugging Implementation Patterns:** Code Breakpoint; Breakpoint Action

- \`\`\`AWAPI-Dumps\Exercise-W3.pdf`\`\`
API Sequences (Prescriptive)

- CreateThread, ..., CloseHandle
- RegisterClass, CreateWindowEx
- GetMessage, TranslateMessage, DispatchMessage
- BeginPaint, ..., EndPaint
- GetDC, ..., ReleaseDC
API Sequences (Descriptive)

- **Horizontal**
  - Code disassembly
  - Traces and logs ([Thread of Activity](#) analysis pattern)

- **Vertical**
  - Stack trace
  - Traces and logs ([Fiber Bundle](#) analysis pattern)
API Internals

- Memory analysis patterns:
  - Hooked Functions (User Space)
  - Module patterns
    - Hooked Modules

- Malware analysis patterns:
  - Patched Code
Exercise W4

- **Goal:** Explore API layers and internals of specific API functions

- **ADDR Patterns:** Function Skeleton; Call Path

- `\AWAPI-Dumps\Exercise-W4.pdf`
Delay-loaded API

- **Documentation**

- **Example:**

  ```
  pub func 00007ffe`e85b6d30 0 winmm!__imp_load_waveInOpen (__imp_load_waveInOpen)
  pub global 00007ffe`e85db3c0 0 winmm!__imp_waveInOpen = <no type information>
  ```
API Sets

- **Documentation**

  \[\text{contract} \_\text{name} \rightarrow \text{module.dll}\]

- **Example of API contract:**

  \[\text{api}\_\text{ms}\_\text{win}\_\text{mm}\_\text{mme}\_\text{l1}\_1\_0 \rightarrow \text{winmmbase.dll}\]
Exercise W5

- **Goal**: Explore the delay-loaded API and API sets
- **Debugging Implementation Patterns**: Code Breakpoint
- **ADDR Patterns**: Call Path
- `\AWAPI-Dumps\Exercise-W5.pdf`
Exports and Imports

- WinDbg (manual/scripts)
- 3rd-party WinDbg extensions (SwishDbgEx)
- DUMPBIN
API and System Calls

- API that do not require kernel services
  - GetCurrentThreadId

- API that require kernel services
  - user32!CreateWindowExW → win32u!NtUserCreateWindowEx
  - kernel32!ReadFile → ntdll!NtReadFile
Exercise W6

- **Goal**: Explore exports and imports using dumpbin. Check whether the selected API functions use a system call

- **ADDR Patterns**: Call Path

- \AWAPI-Dumps\Exercise-W6.pdf
Documented API

- Online documentation
- Present in headers

Example:

Documentation

kernel32!SuspendThread → KERNELBASE!SuspendThread → ntdll!NtSuspendThread
Undocumented API

ntdll!NtSuspendProcess

0:000> x /v ntdll!*
...
pub func 00007ffc`f6346ff0 0 ntdll!NtSuspendProcess (NtSuspendProcess)
...
API Source Code

- **Wine** (GitLab) / **Wine-Mirror** (GitHub)

- Example:

  user32!CreateWindowExW

  https://gitlab.winehq.org/wine/wine/-/blob/master/dlls/user32/win.c
  https://github.com/wine-mirror/wine/blob/master/dlls/user32/win.c
API Name Patterns

- Create/Open/Delete/Close
- Process/Thread
- Memory
- Read/Write

WinDbg Commands

0:000> x /v module!fpattern
API Namespaces

- API sets / contracts
  - Example: `CreateDialogParamW`

- Functions required to accomplish a particular task
  - Example: `screen capture` vs. `saving image`
    - `gdi32!CreateCompatibleDC` vs. `GdiPlus!GdiplusStartup`
    - `gdi32!StretchBlt` vs. `GdiPlus!GdiSaveImageToStream`
    - `gdi32!CreateDIBSection` vs. `GdiPlus!GdiGetImageEncodersSize`
    - `gdi32!SelectObject` vs. `GdiPlus!GdiDisposeImage`
    - `user32!ReleaseDC` vs. `GdiPlus!GdiCreateBitmapFromHBITMAP`
    - `user32!NtUserGetWindowDC` vs. `GdiPlus!GdiGetImageEncoders`
    - `user32!GetWindowRect` vs. `ole32!CreateStreamOnHGlobal`
API Syntagms/Paradigms

- **Syntagms** / syntagmatic analysis
- **Paradigms** / paradigmatic analysis

Diagram:

- **Paradigmatic axis**: CreateEventW, CreateThread, CreateProcessW
- **Syntagmatic axis**: WaitForSingleObject, CloseHandle
Marked API

- **Marked Message** trace and log analysis pattern
- Points to presence or absence of activity

**Example:**

- `CreateThread [-]
- `socket [+]
- `GetMessageW [-]
- `ReadConsoleW [+]

WinDbg Commands

```
0:000> x app!_imp_pattern
```
ADDR Patterns

- From *Accelerated Disassembly Deconstruction Reversing*
  - List of pattern names
  - Pattern descriptions
DebugWare Patterns

- Patterns for troubleshooting and debugging tools
  - API Query
    - Periodic or asynchronous query of the same set of API and logging of their input and output data.
  - Example: WindowHistory
Patterns vs. Analysis Patterns

**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, ...
Memory Dump Types

- Kernel Mode/Space
  - ntkrnlmp.exe
  - syscall
  - NtReadFile

- User Mode/Space
  - ntdll.dll
  - KERNELBASE.dll
  - Application.exe
  - ReadFile

Types:
- Kernel dump
- Process dump
- Complete (physical) dump

© 2022 Software Diagnostics Services
Memory Analysis Patterns

- User space
  - Process memory dumps
  - Complete memory dumps

- Function analysis patterns
  - Stack Trace Collection
  - Well-Tested Function
  - False Function Parameters
  - String Parameter
  - Small Value / Design Value
  - Virtualized Process
  - Stack Trace
  - Execution Residue
  - Hidden Parameter
  - Parameter Flow
  - Data Correlation
Thread and Adjoint Thread
Fiber Bundle
Trace and Log Analysis Patterns

- Process Monitor
- Function calls:
  - Thread of Activity
  - Fiber of Activity
  - Adjoint Thread of Activity
  - Strand of Activity
  - Discontinuity
  - Fiber Bundle
  - Weave of Activity
WOW64

- 64-bit process dumps
- 32-bit process dumps
- wow64 (kernel32)
- wow64win (user32, gdi32)
- wow64cpu
Exercise W7

- **Goal:** Explore Windows API calls in the WOW64 context

- **Memory Analysis Patterns:** Stack Trace Collection; Virtualized Process

- `\AWAPI-Dumps\Exercise-W7.pdf`
API and Errors

- Windows protocols
- Windows error codes reference
- Thread Information Block
- Win32 values
- NTSTATUS values
- HRESULT values
Exercise W8

- **Goal:** Explore different Windows API error types

- **Memory Analysis Patterns:** Last Error Collection

- **ADDR Patterns:** Function Skeleton; Call Path; Structure Field

- \AWAPI-Dumps\Exercise-W8.pdf
Windows API Formalization

Ideas from Conceptual Mathematics
API Compositionality

- **Principle of compositionality**
Category Theory Language

- **Category**
  - Objects
  - Arrows between objects (must be transitive, if $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$)

- **Functor**
  - Arrow between categories (can be the same category)
  - Maps objects to objects and arrows to arrows

- **Natural Transformation**
  - Arrows between functors in a category of functors

- **Adjunction**
  - Relationship between functors, change of perspective, back translation
A View of Category Theory
Category Theory Square

Category → Functor → Natural Transformation

Category ← Adjunction ← Natural Transformation
API Category

- API as objects, glue code as arrows
- API as arrows, glue code as objects fails at composition
- Initial and terminal API objects in subcategories
API Functor

- Translates between API layers (different API)
- Stack trace as functor
- Translates between different API sequences
- Endofunctor – between the same API
- Translates between different code implementations
### API Natural Transformation

- Maps between different vertical API sequences (stack traces)
- Maps between different code translations
- Diagnostics and debugging as natural transformation
API Adjunction

- Navigation between different API sequences
- Call and return stack trace sequences, callbacks (when stack traces correspond to vertical API sequences)
- Back translation between traces/logs (when traces correspond to API horizontal sequences)
Informal n-API

- Arrows between arrows

- 1-API – normal API usage

- 2-API – diagnostics, debugging

- 3-API – higher diagnostics, debugging (debugging the debugging)

- ∞-API – for homework 😊
API I/O

- Categories – one input, one output
- Operads – many inputs, one output
- Properads – many inputs, many outputs

```c
HANDLE CreateThread(
    [in, optional]  LPSECURITY_ATTRIBUTES lpThreadAttributes,
    [in]            SIZE_T dwStackSize,
    [in]            LPTHREAD_START_ROUTINE lpStartAddress,
    [in, optional]  __drv_aliasesMem LPVOID lpParameter,
    [in]            DWORD dwCreationFlags,
    [out, optional] LPDWORD lpThreadId
);
```
Windows API and Languages
API and C#

- P/Invoke example:

```csharp
using System;
using System.Runtime.InteropServices;

class WapiTest
{
    [DllImport("user32.dll")]
    public static extern uint MessageBox(ulong hWnd, string message, string title, uint flags);

    public static void Main()
    {
        MessageBox(0, "Hello Windows API!", "From C!", 0);
    }
}
```
API Metadata

- Overview

- C#/Win32 P/Invoke Source Generator
API and Scala Native

- **Documentation**
- **Scala Native**
- **Native code interoperability**
Exercise W9

- **Goal:** Install Scala Native environment and write a simple program that uses Windows API

- \AWAPI-Dumps\Exercise-W9.pdf
API and Golang

- **windows package**

- **Example:**

```go
package main

import "unsafe"
import "golang.org/x/sys/windows"

func main() {
    var user32 = windows.NewLazyDLL("user32.dll")
    var procMessageBox = user32.NewProc("MessageBoxW")

    message, _ := windows.UTF16PtrFromString("Hello Windows API!")
    title, _ := windows.UTF16PtrFromString("From Golang")
    procMessageBox.Call(0, uintptr(unsafe.Pointer(message)), uintptr(unsafe.Pointer(title)), 0)
}
```
API and Rust

- **Rust for Windows**

- **Example:**

```rust
use windows_sys::{
    core::*,
    Win32::UI::WindowsAndMessaging::*
};

fn main() {
    unsafe {
        MessageBoxW(0, w!("Hello Windows API"), w!("From Rust"), 0);
    }
}
```

- **Other bindings:** [winapi-rs](https://github.com/rustwg/winapi-rs)
API and Python

- ctypes library

- Example:

```python
from ctypes import windll

windll.user32.MessageBoxW(0, "Hello Windows API!", "From Python", 0)
```
Windows API Classes

With MAP (Memory Analysis Patterns)
General Resources

- **API Index**
- **Windows SDK**
Windowing API

- Documentation
- WindowHistory64
- Window2Dump
- WNDCLASS → Subclassing
- WNDPROC

Diagram:
- Process
- Session
- Desktop
- Thread
- Window
- Parent Window
- Child Window

© 2022 Software Diagnostics Services
Messaging API

- Documentation
- PostMessage to thread message queue
- Thread message loop
- SendMessage directly to WndProc (Wait Chain)
- Message IPC (WM_COPYDATA)
- MessageHistory(64) → Hooking (Message Hooks, modeling example)
- Structure:
  - Message ID (WM_xxx, EM_xxx, LB_xxx, ...) – WinUser.h
  - 2 parameters (wParam and lParam)
  - WM_LBUTTONDOWN

- Blocking: Message Box, Dialog Box, Input Thread
- UI Problem Patterns: Error Message Box, Unresponsive Window
GDI API

- Documentation
- Device contexts
- Handle Limit (User Space)
- Handle Limit (Kernel Space)
- Create → Delete
- Screen glitches
GDI+ API

- Documentation
- C++ library
- gdiplus module
- Imports from gdi32

© 2022 Software Diagnostics Services
Module/Library API

- `libloaderapi.h`
- `LoadLibraryW, GetProcAddress, FreeLibrary`
- Module analysis patterns
- Unloaded Module
- Dynamic linking (Missing Component)
- Static linking (Missing Component)
Process/Thread API

- Overview
- Process analysis patterns
- Thread analysis patterns
- CreateThread → CloseHandle
- Zombie Processes
- Insufficient Memory (Handle Leak)
Services API

- **Documentation**
- Sample service template (included in the book)
- **Input Thread**
Security API

- Documentation
- Deviant Token
IPC API

- **Overview**
- **Wait Chain (Named Pipes)**
- **Coupled Processes**
  - Strong
  - Weak
  - Semantics
RPC API

- Overview
- Wait Chain (RPC)
- Semantic Structure (PID.TID)
- LPC/ALPC/RPC patterns and case studies
Synchronization API

- syncapi.h
- Wait chain analysis patterns
- Deadlock and livelock analysis patterns
- Shared Structure
I/O API

- Device I/O `ioapiset.h`
- File I/O `fileapi.h`
- **Blocking File**
- Handle (**Invalid Handle**) 
- `CreateFile` → `CloseHandle` (**Handle Leak**)
Runtime API

- Reference (Windows Runtime)
- C/C++ runtime (ucrtbase, msvcr100, msvcp_win, msvcp110_win)
- Invalid Parameter
- C++ Exception
- Wait Chain (C++11, Condition Variable)
COM API

- Technology-Specific Subtrace (COM Client Call)
- Technology-Specific Subtrace (COM Interface Invocation)
- Errors: HRESULT
- COM Exception
- C++ Object
- COM Object

Diagram:
- DLL
- COM object
- COM object
- vptr
- QueryInterface
  AddRef
  Release
- vtbl
- foo
- bar
- ~destructor
- ... vtable
- ... foo:
  ...
  ...
  ...
- Heap
Exercise W10

- **Goal:** Find COM objects and their interfaces in raw stack regions

- **Memory Analysis Patterns:** Stack Trace Collection; Technology-Specific Subtrace (COM Client Call); Execution Residue (Unmanaged Space, User); COM Object; C++ Object

- \AWAPI-Dumps\Exercise-W10.pdf
Networking API

- **winsock2.h**
  - send and WSASend
- **Winsock SPI**
- **High Contention (Sockets)**
Console API

- Documentation
- Input Thread
- Main Thread
Process Heap API

- `heapapi.h`
- Used by C/C++ runtime like malloc
- HeapAlloc is redirected to ntdll!RtlAllocateHeap
- There can be several process heaps
- Large allocations use Virtual Memory API
- Leaks
  - Memory Leak (Process Heap)
  - Relative Memory Leak
  - Memory Fluctuation (Process Heap)
- Corruption / Double Free / Invalid Parameter

WinDbg Commands

```
0:000> !heap
```
Virtual Memory API

- `memoryapi.h`
- Used by large heap allocations and .NET
- Different memory protections
- Insufficient Memory (Reserved Virtual Memory)
- Insufficient Memory (Region)
- Memory Leak (Regions)
- Insufficient Memory (Module Fragmentation) – 32-bit

WinDbg Commands

0:000> !address
Structures, Types, and Variables

- Injected Symbols
- Windows Data Types
References and Resources
Reading Windows-based Code

- Legacy Windows code and C language
  - Part 1
  - Part 2
  - Part 3
  - Part 4
  - Part 5
  - Part 6
Resources (Construction)

- Learning DCOM
- Programming Windows, 5th Edition
- Subclassing and Hooking with Visual Basic: Harnessing the Full Power of VB/VB.NET
- Windows Graphics Programming: Win32 GDI and DirectDraw
- Introduction to 3D Game Programming with DirectX 12
- Windows via C/C++
- Windows System Programming, 4th Edition
- Windows 10 System Programming
- Concurrent Programming on Windows
- The Old New Thing (also for postconstruction)
- Introducing Windows 7 for Developers
- Fundamentals of Audio & Video Programming for Games
- Software Application Development: A Visual C++, MFC, and STL Tutorial
Resources (Postconstruction)

- 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, Second Edition
- Software Diagnostics Library
- Encyclopedia of Crash Dump Analysis Patterns, Third Edition
- Trace, Log, Text, Narrative
- Memory Dump Analysis Anthology (Diagnomicon)
Resources (Training)

- Accelerated Windows Memory Dump Analysis, Fifth Edition, Revision 3
- Accelerated Windows Malware Analysis with Memory Dumps, Third Edition
- Accelerated Disassembly, Reconstruction and Reversing, Second Revised Edition
- Accelerated Windows Trace and Log Analysis, Second Edition
Resources (Category Theory)

Applied category theory books that have chapters explaining category theory:

- Conceptual Mathematics: A First Introduction to Categories
- The Joy of Abstraction: An Exploration of Math, Category Theory, and Life
- *Category Theory for Programmers*
- Categories for Software Engineering
- An Invitation to Applied Category Theory: Seven Sketches in Compositionality
- Life Itself: A Comprehensive Inquiry Into the Nature, Origin, and Fabrication of Life
- Category Theory for the Sciences
- Conceptual Mathematics and Literature: Toward a Deep Reading of Texts and Minds
- Diagrammatic Immanence: Category Theory and Philosophy
- Mathematical Mechanics: From Particle To Muscle
- Memory Evolutive Systems; Hierarchy, Emergence, Cognition
- Mathematical Structures of Natural Intelligence
- Sheaf Theory Through Examples
- *Visual Category Theory*
Q&A

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