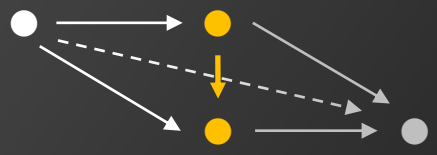
Part 1 Draft

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 sets and categories
- 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
- …

- API internals
- API name patterns
- API namespaces
- Marked API
- ADDR patterns
- DebugWare patterns
- Memory analysis patterns
- Trace and log analysis patterns
- WOW64

Windows API Formalization

- API compositionality
- A view of category theory
- API category
- API functor
- API natural transformation
- n-API
- …

Windows API Sets and Categories

- GUI
- Windowing
- GDI
- GDI+
- DirectX
- Process/Thread
- Services
- Security
- IPC
- Synchronization
- I/O
- Runtime
- COM
- Networking
- …

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
- Daily programming using Windows API 1990 – 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?

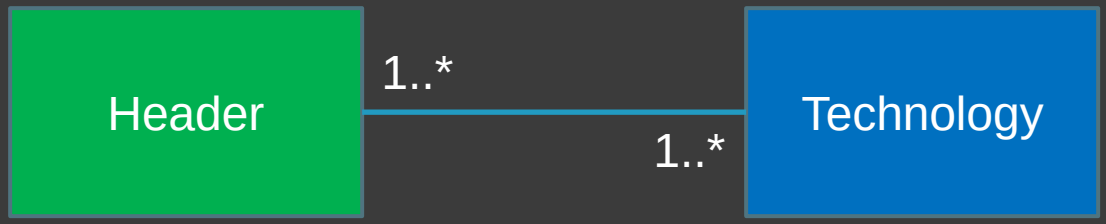
- Source code perspective (SDK and/or WDK)
- ABI (Application Binary Interface) perspective

Source Code Examples

- Windowing
- Multithreading
- Services
- COM
- Networking
- File I/O

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`

WinDbg Commands

```
0:000> dt WinTypes!DWORD
Uint4B

0:000> dt WinTypes!WPARAM
Uint8B

0:000> dt WinTypes!LONG
Int4B
```

Hungarian Notation

- Wikipedia reference
- Microsoft reference
- CreateWindowExW / WNDCLASSW

```
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
);
```

```
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)

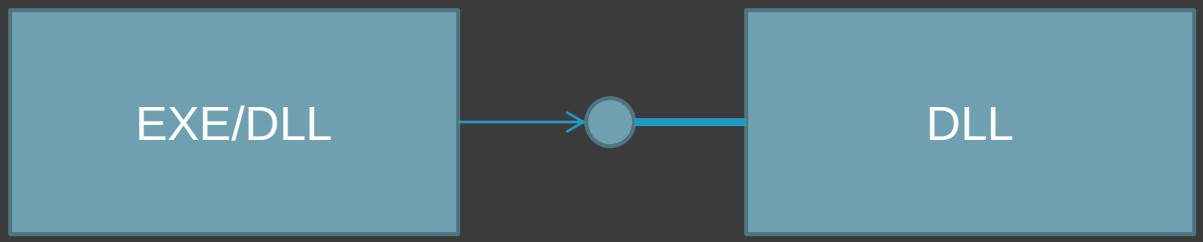
```
USER32!CreateWindowExW:
…
0007ffa`5e0cf20d e812000000 call USER32!CreateWindowInternal (00007ffa`5e0cf224)
```

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

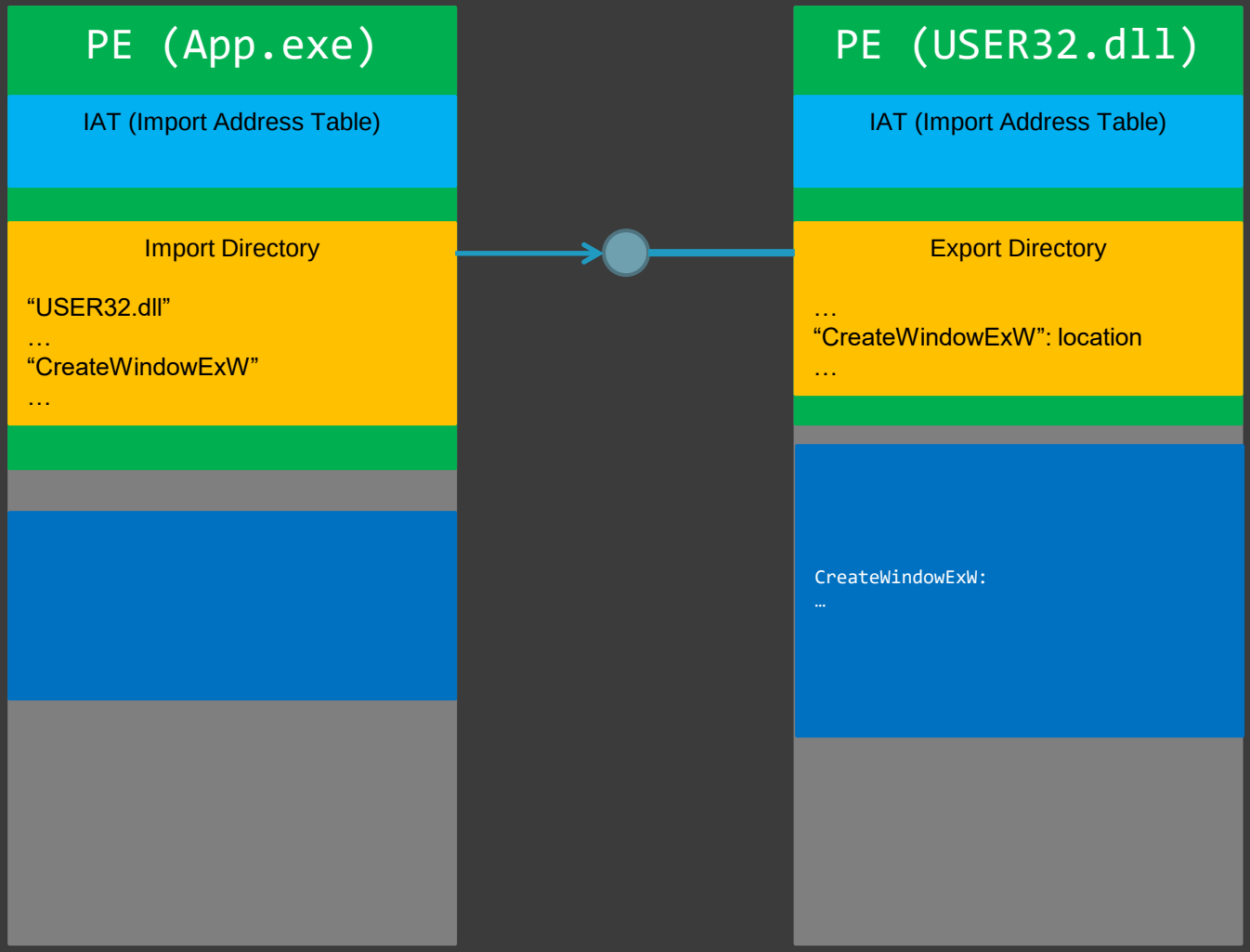
```
App!wWinMain:
…
00007ff6`77741101 ff15c1e10000 call qword ptr [App!_imp_CreateWindowExW (00007ff6`7774f2c8)]
```

API as Interface

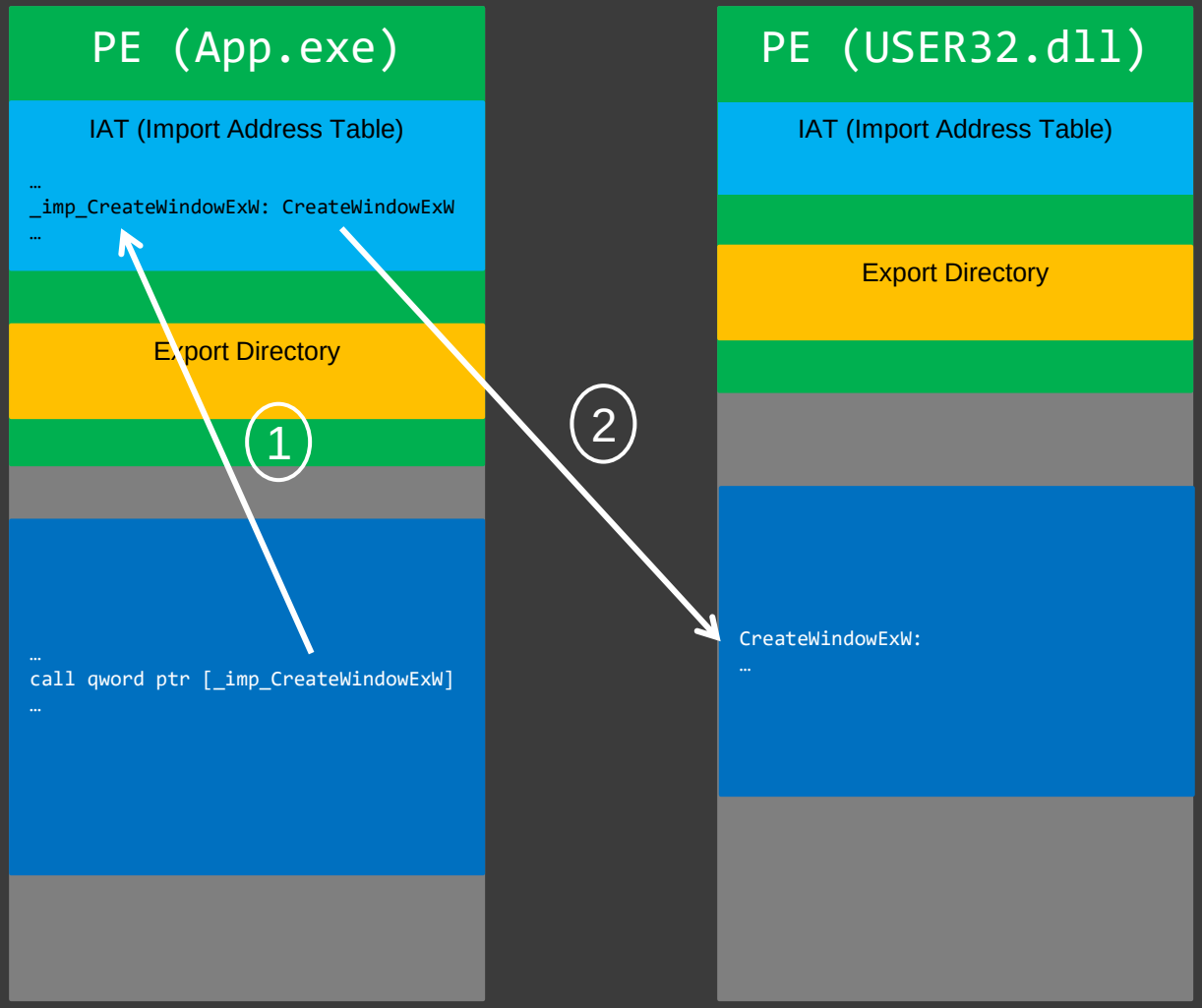
- Provided by (exported from) some DLL module (may have different file extensions)
- Used by (imported by) EXE or DLL
- Can be functional or object-oriented



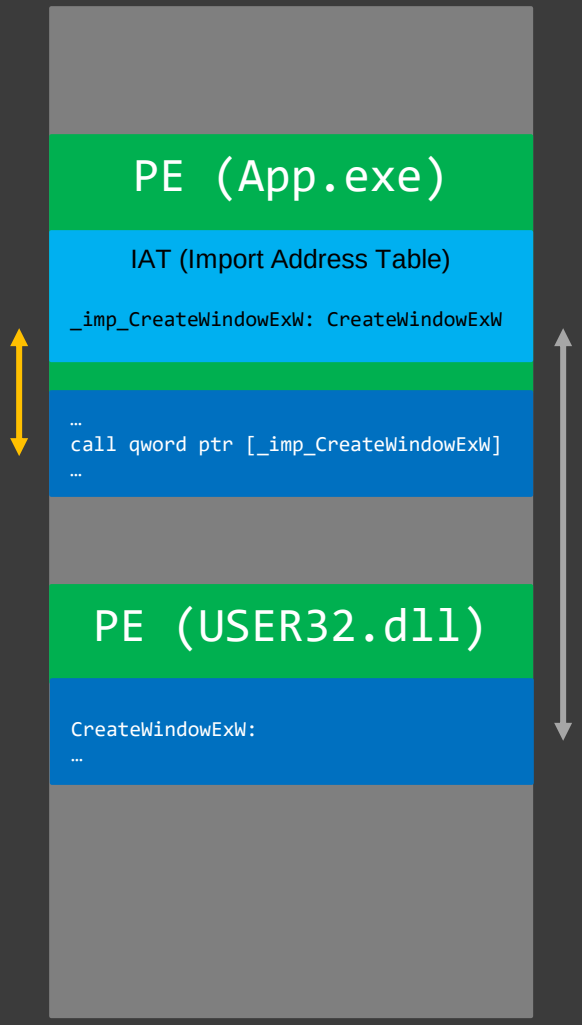
Export Directory

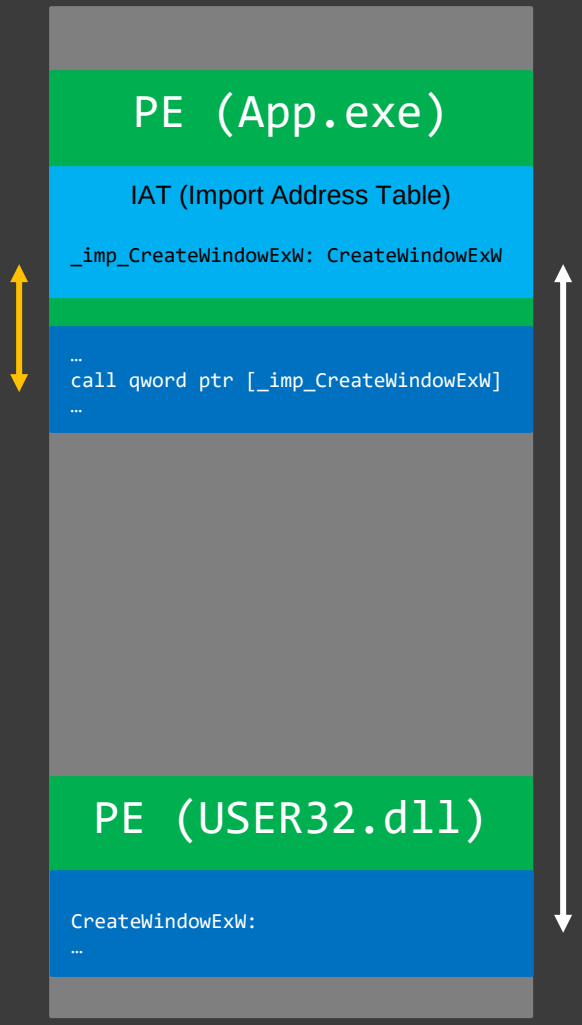


IAT (Import Address Table)



Virtual Process Address Space





Calling Convention

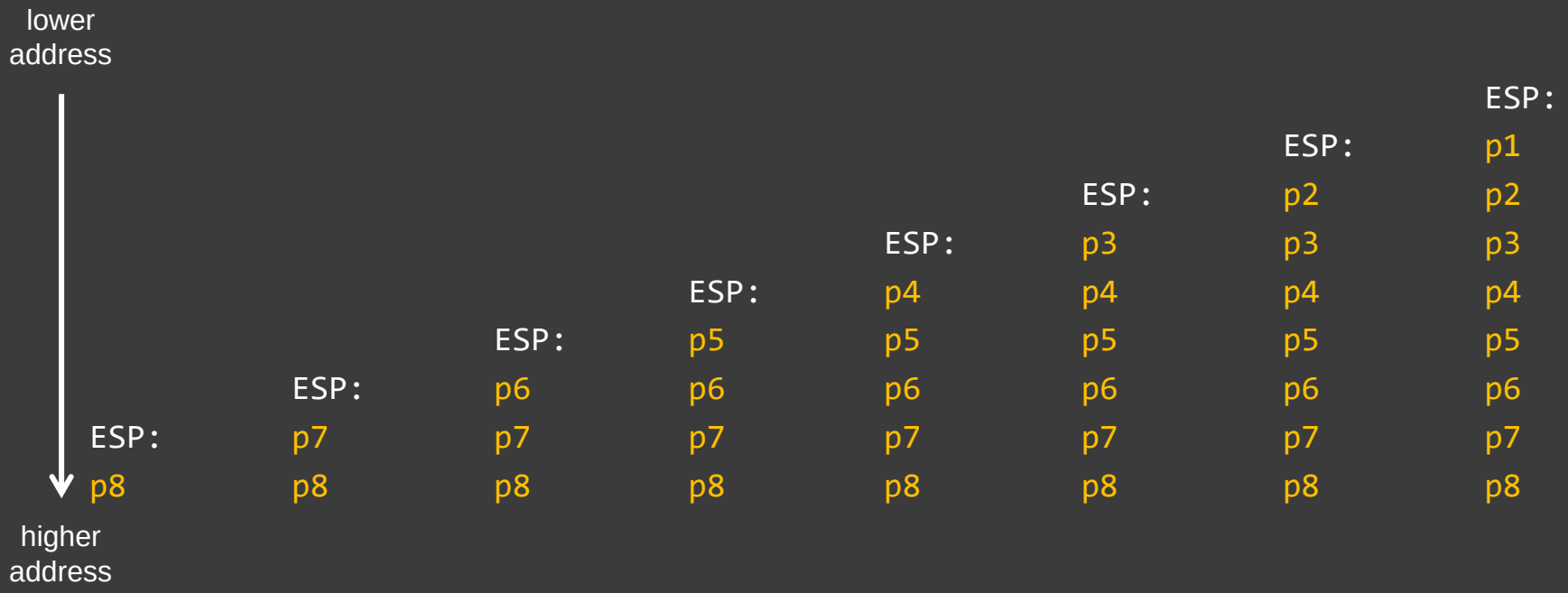
- `GetMessageW` (from documentation)
- Actual declaration (`WinUser.h`)

```
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);
```



Parameter Passing (x64)

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

Caller

```
ECX (p1)
EDX (p2)
R8D (p3)
R9D (p4)
```

```
RSP:     0`0
RSP+8:   0`0
RSP+10:  0`0
RSP+18:  0`0
RSP+20:  0`p5
RSP+28:  0`p6
RSP+30:  0`p7
RSP+38:  0`p8
```

call

Callee

```
ECX (p1)
EDX (p2)
R8D (p3)
R9D (p4)
```

```
RSP:     return address
RSP+8:   0`0
RSP+10:  0`0
RSP+18:  0`0
RSP+20:  0`0
RSP+28:  0`p5
RSP+30:  0`p6
RSP+38:  0`p7
RSP+40:  0`p8
```

Exercise W1

- **Goal:** Compare calling conventions on x86 and x64 platforms
- **ADDR Patterns:** Call Prologue
- \AWAPI-Dumps\Exercise-W1.pdf