Windows API for Software Diagnostics
Accelerated
With Category Theory in View

Dmitry Vostokov
Software Diagnostics Services
Contents

About the Author .......................................................................................................................................................... 5

Introduction .................................................................................................................................................................... 7
  Exercise W0: Download, setup, and verify your WinDbg Preview or WinDbg installation, or Docker Debugging Tools for Windows image ......................................................................................................................... 21

General Windows API Aspects .................................................................................................................................. 37
  Exercise W1 ............................................................................................................................................................. 56
  Exercise W2 ............................................................................................................................................................. 71
  Exercise W3 ............................................................................................................................................................. 93
  Exercise W4 ............................................................................................................................................................. 136
  Exercise W5 ............................................................................................................................................................. 160
  Exercise W6 ............................................................................................................................................................. 175
  Exercise W7 ............................................................................................................................................................. 218
  Exercise W8 ............................................................................................................................................................. 227

Windows API Formalization ...................................................................................................................................... 233

Windows API and Languages ..................................................................................................................................... 247
  Exercise W9 ............................................................................................................................................................. 254

Windows API Classes .................................................................................................................................................. 261
  Exercise W10 ........................................................................................................................................................... 284

References and Resources .............................................................................................................................................. 297
Exercise W1

Goal: Compare calling conventions on x86 and x64 platforms.

ADDR Patterns: Call Prologue; Call Parameter; Function Prologue.


2. Open \AWAPI-Dumps\Process\wordpad.DMP

3. We get the dump file loaded:

Microsoft (R) Windows Debugger Version 10.0.25200.1003 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\AWAPI-Dumps\Process\wordpad.DMP]
User Mini Dump File with Full Memory: Only application data is available

*************** Path validation summary ***************
Response Time (ms) Location
Deferred srv*
Symbol search path is: srv*
Executable search path is:
Windows 10 Version 22000 MP (2 procs) Free x64
Product: WinNt, suite: SingleUserTS Personal
Edition build lab: 22000.1.amd64fre.co_release.210604-1628
Machine Name:
Debug session time: Sat Oct 15 20:19:05.000 2022 (UTC + 0:00)
System Uptime: 0 days 0:27:57.202
Process Uptime: 0 days 0:00:26.000

.....
Loading unloaded module list
For analysis of this file, run !analyze
win32u!NtUserGetMessage+0x14:
00007ffc`f3811414 c3 ret

4. Open a log file using .logopen:

0:000> .logopen C:\AWAPI-Dumps\W1.log
Opened log file 'C:\AWAPI-Dumps\W1.log'

5. There are several places in Wordpad code where windows are created. We look at two of them. First, disassemble the InitMainThreadWnd function from the combase module and find the CreateWindowExW API call:

0:000> uf combase!InitMainThreadWnd
combase!InitMainThreadWnd [omcore\com\combase\object\mainthrd.cxx @ 114]:
114 00007ffc`f5e14770 4c8bdc mov r11,rsp
114 00007ffc`f5e14773 53 push rbx
114 00007ffc`f5e14774 4883ec60 sub rsp,60h
148 00007ffc`f5e14778 498363f000 and qword ptr [r11-10h],0
148 00007ffc`f5e1477d 4c8d05b45e1d00 lea r8,[combase!`string' (00007ffc`f5fea638)]
148 00007ffc`f5e14783 488b659d1a2400 mov rax,qword ptr [combase!g_hinst (00007ffc`f6056228)]
148 00007ffc`f5e1478b 411000000000 mov r9d,000000000000h
The API function definition is:

```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
);
```

We see the first 4 parameters are passed via ECX, RDX, R8, and R9D according to their size, 32-bit or 64-bit:

```assembly
148 00007ffc`f5e1477d 4c8d05b45e1d00  lea     r8,[combase\string\ (00007ffc`f5fe638)]
148 00007ffc`f5e1478b 41b900000088    mov     r9d,88000000h
148 00007ffc`f5e14791 488b15c0232400  mov     rdx,qword ptr [combase!gOleWindowClass (00007ffc`f6056b58)]
148 00007ffc`f5e14798 33c9            xor     ecx,ecx
```

R8 contains the 64-bit pointer to a window name (UNICODE):

```
0:00> du 00007ffe`f5fe638
00007ffe`f5fe638  "OleMainThreadWndName"
```

The next 4 parameters are passed via direct stack storage (a default rectangle value):

```assembly
148 00007ffe`f5e147b0 89442438        mov     dword ptr [rsp+38h],eax
148 00007ffe`f5e147b4 89442430        mov     dword ptr [rsp+30h],eax
148 00007ffe`f5e147b8 89442428        mov     dword ptr [rsp+28h],eax
148 00007ffe`f5e147bc 89442420        mov     dword ptr [rsp+20h],eax
```

It is interesting to see that the last four parameters are passed via direct stack storage as well but using R11 as saved RSP value during function prologue (here, both prologues overlap in purpose):

```assembly
114 00007ffe`f5e14770 4c8bdc          mov     r11,rsp
114 00007ffe`f5e14773 53              push    rbx
114 00007ffe`f5e14774 4883ec60        sub     rsp,60h
148 00007ffe`f5e14778 498363f000      and     qword ptr [r11-10h],0
148 00007ffe`f5e1477a 498943e0        mov     qword ptr [r11-18h],rax
148 00007ffe`f5e1477c 498363e000      and     qword ptr [r11-20h],0
148 00007ffe`f5e1477e 49c743d8ffffff mov     qword ptr [r11-28h],0FFFFFFFFFFFFFFFDh
```

Before the `InitMainThreadWnd` function prologue is executed, RSP points to the save return address. So R11 points to that address as well. RSP is then decremented by 8 via the next push instruction. Then 60h bytes (0x60) are reserved, and RSP now points down 0x68 bytes down its original value:

```
R11:    return address
R11-8h (RSP+60h):  saved RBX
R11-10h (RSP+58h):
R11-18h (RSP+50h):
R11-20h (RSP+48h):
R11-28h (RSP+40h):
RSP+38h:
RSP+30h:
RSP+28h:
RSP+20h:
```
So, by this illustration above, we see the continuity of stack-based parameters despite different CPU registers and offsets used. We also see that all parameters are passed left-to-right.

6. Second, disassemble the CWnd::CreateEx function from the MFC42u module and find the CreateWindowExW API call:

```
0:000> uf MFC42u!CWnd::CreateEx
mfc42u!CWnd::CreateEx:
00007ffc`acc1ca50 488bc4    mov    rax,rsp
00007ffc`acc1ca53 48895808   mov    qword ptr [rax+8],rbx
00007ffc`acc1ca57 48897018   mov    qword ptr [rax+10h],rsi
00007ffc`acc1ca5b 48897818   mov    qword ptr [rax+18h],rdi
00007ffc`acc1ca5f 55         push    rbp
00007ffc`acc1ca60 488d68e1   lea    rbp,[rax-1Fh]
00007ffc`acc1ca64 4881ecb0000000 sub    rsp,0B0h
00007ffc`acc1ca6b 8b4547     mov    eax,dword ptr [rbp+47h]
00007ffc`acc1ca6e 488bd9     mov    rbx,rcx
00007ffc`acc1ca71 8945f7     mov    dword ptr [rbp-9],eax
00007ffc`acc1ca74 8b45f3     mov    eax,dword ptr [rbp+8Fh],edx
00007ffc`acc1ca77 8b45f7     mov    eax,dword ptr [rbp-11h],eax
00007ffc`acc1ca7a 8b455F     mov    edx,dword ptr [rbp-15h],eax
00007ffc`acc1ca7D 8b455F     mov    edx,dword ptr [rbp-15h],eax
00007ffc`acc1ca80 8b4567     mov    eax,dword ptr [rbp+6Fh]
00007ffc`acc1ca83 8b4567     mov    eax,dword ptr [rbp+6Fh]
00007ffc`acc1ca86 8b4567     mov    eax,dword ptr [rbp+6Fh]
00007ffc`acc1ca89 8b4567     mov    eax,dword ptr [rbp+6Fh]
00007ffc`acc1ca8c 8b4567     mov    eax,dword ptr [rbp+6Fh]
00007ffc`acc1ca90 8b4567     mov    eax,dword ptr [rbp+6Fh]
```

```
mfc42u!CWnd::CreateEx+0x1c1d2 (00007ffc`acc38c22)  Branch
mfc42u!CWnd::CreateEx+0x97:
00007ffc`acc1ca94 8b8b4577   mov    rax,qword ptr [rbp+77h]
00007ffc`acc1ca98 8b945d7    mov    qword ptr [rbp+29h],rax
00007ffc`acc1ca9c 8b955Ff    mov    rcx,dword ptr [rbp+8Fh],edx
00007ffc`acc1ca9F 4c8945f7   mov    qword ptr [rbp+15h],eax
00007ffc`acc1caaa 4c8945f7   mov    qword ptr [rbp+15h],eax
00007ffc`acc1cab0 488945c7   mov    qword ptr [rbp-39h],rax
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
00007ffc`acc1cab8 488b03     mov    rax,qword ptr [rbp-39h]
```
We see that 8 parameters are passed using RSP pointer and consecutive offsets.

7. Let’s look at an indirect call through IAT:

0:000> dps 00007ffc`acd1c5d0 L1
00007ffc`acd1c5d0 0007ff`f5848030 user32!CreateWindowExW

0:000> idh mfc42u

File Type: DLL
FILE HEADER VALUES
8664 machine (X64)
7 number of sections
F91A937D time date stamp Fri Jun 9 04:54:37 2102

0 file pointer to symbol table
0 number of symbols
F0 size of optional header
2022 characteristics
  Executable
  App can handle >2gb addresses
  DLL

OPTIONAL HEADER VALUES
200 magic #
14.28 linker version
F200 size of code
81000 size of initialized data
0 size of uninitialized data
21730 address of entry point
1000 base of code
----- new -----
00007ffacc10000 image base
1000 section alignment
1000 file alignment
3 subsystem (Windows CUI)
10.00 operating system version
10.00 image version
10.00 subsystem version
174000 size of image
1000 size of headers
17BE15 checksum
0000000000000000 size of stack reserve
0000000000000000 size of stack commit
0000000000000000 size of heap reserve
000000000000010000 size of heap commit
4160 DLL characteristics
   High entropy VA supported
   Dynamic base
   NX compatible
   Guard
139980 [    6CC0] address [size] of Export Directory
140640 [    21C] address [size] of Import Directory
151000 [    100A4] address [size] of Exception Directory
 0 [    0] address [size] of Security Directory
16E000 [    5734] address [size] of Base Relocation Directory
116B60 [    70] address [size] of Debug Directory
 0 [    0] address [size] of Description Directory
 0 [    0] address [size] of Special Directory
FD4A0 [     28] address [size] of Thread Storage Directory
FBC00 [     138] address [size] of Load Configuration Directory
 0 [    0] address [size] of Bound Import Directory
10BD18 [    1548] address [size] of Import Address Table Directory
13838C [    180] address [size] of Delay Import Directory
 0 [    0] address [size] of COR20 Header Directory
 0 [    0] address [size] of Reserved Directory

SECTION HEADER #1
   .text name
   F124E virtual size
   1000 virtual address
   F2000 size of raw data
   1000 file pointer to raw data
   0 file pointer to relocation table
   0 file pointer to line numbers
   0 number of relocations
   0 number of line numbers
60000020 flags
   Code
   (no align specified)
   Execute Read

SECTION HEADER #2
   .rdata name
   51894 virtual size
   F3000 virtual address
   52000 size of raw data
   F3000 file pointer to raw data
0 file pointer to relocation table
0 file pointer to line numbers
0 number of relocations
0 number of line numbers
40000040 flags
   Initialized Data
   (no align specified)
   Read Only

Debug Directories(4)
   Type       Size     Address  Pointer
   cv           23      126810   126810  Format: RSDS, guid, 1, mfc42u.pdb
     (   13)     5e4      126834  126834
     (   16)      24      126e18   126e18
dllchar       4      126e3c   126e3c

00000001 extended DLL characteristics
   CET compatible

SECTION HEADER #3
   .data name
   B19C virtual size
   145000 virtual address
   5000 size of raw data
   145000 file pointer to raw data
      0 file pointer to relocation table
      0 file pointer to line numbers
      0 number of relocations
      0 number of line numbers
   C0000040 flags
      Initialized Data
      (no align specified)
      Read Write

SECTION HEADER #4
   .pdata name
   100A4 virtual size
   151000 virtual address
   11000 size of raw data
   14A000 file pointer to raw data
      0 file pointer to relocation table
      0 file pointer to line numbers
      0 number of relocations
      0 number of line numbers
   40000040 flags
      Initialized Data
      (no align specified)
      Read Only

SECTION HEADER #5
   .didat name
   548 virtual size
   162000 virtual address
   1000 size of raw data
   150000 file pointer to raw data
      0 file pointer to relocation table
      0 file pointer to line numbers
      0 number of relocations

62
We see that the `mfc42u!_imp_CreateWindowExW` address is inside the IAT memory region of length 0x1548 bytes:

```
0:000> ? 00007ffcacc10000 + 10bd18
Evaluate expression: 140723207912728 = 00007ffc`acd1bd18
```

```
0:000> * mfc42u!_imp_CreateWindowExW (00007ffc`acd1c5d0)
```

```
0:000> ? 00007ffcacc10000 + 10bd18 + 1548
Evaluate expression: 140723207918176 = 00007ffc`acd1d260
```

8. We close logging before exiting WinDbg Preview:

```
0:000> .logclose
Closing open log file C:\AWAPI-Dumps\W1.log
```

**Note:** To avoid possible confusion and glitches, we recommend exiting WinDbg Preview or WinDbg after each exercise.

9. Let’s now look at a 32-bit process. Launch the new instance of WinDbg Preview.

10. Open \AWAPI-Dumps\Process\x86\wordpad.DMP
11. We get the dump file loaded:

Microsoft (R) Windows Debugger Version 10.0.25200.1003 X86
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\AWAPI-Dumps\Process\x86\wordpad.DMP]
User Mini Dump File with Full Memory: Only application data is available

************* Path validation summary **************
Response Time (ms) Location
Deferred srv*
Symbol search path is: srv*
Executable search path is:
Windows 10 Version 22000 MP (2 procs) Free x86 compatible
Product: WinNt, suite: SingleUserTS Personal
Edition build lab: 22000.1.amd64fre.co_release.210604-1628
Machine Name:
Debug session time: Fri Nov 4 21:05:05.000 2022 (UTC + 0:00)
System Uptime: 0 days 0:39:06.687
Process Uptime: 20 days 0:46:35.000

.........................................................

Loading unloaded module list
.
For analysis of this file, run !analyze-v

eax=00000000 ebx=010593c8 ecx=00000000 edx=00000000 esi=010593fc edi=010593fc
eip=75c032e3 esp=0013f870 ebp=0013f8a8 iopl=0         nv up ei pl nz ac po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000212

win32u!NtUserGetMessage+0xc:
75c032e3 c21000          ret     10h

12. Open the same log file using .logappend:

0:000> .logappend C:\AWAPI-Dumps\W1.log
Opened log file 'C:\AWAPI-Dumps\W1.log'

13. Let's look at the InitMainThreadWnd function from the combase module and find the CreateWindowExW API call:

0:000> uf combase!InitMainThreadWnd
0:000> uf combase!InitMainThreadWnd
combase!InitMainThreadWnd [onecore]\com\combase\objc\mainthrd.cxx @ 114]:

114 75c0332e3 8bff            mov     edi,edi
114 75c0332e5 51              push    ecx
114 75c0332e6 56              push    esi
115 75c0332e7 33f6            xor     esi,esi
115 75c0332e9 57              push    edi
115 75c0332ea 46              inc     esi
125 75c0332eb e8235ef8ff      call    combase!IsWOWThread (75b89113)
125 75c033f0 85c0            test    eax,eax
125 75c033f2 7546            jne     combase!InitMainThreadWnd+0x57 (75c0333a) Branch

combase!InitMainThreadWnd+0x11 [onecore]\com\combase\objc\mainthrd.cxx @ 148]:

148 75c0332f4 33ff            xor     edi,edi
148 75c0332f6 b000000000      mov    eax,80000000
148 75c0332f7 push    edi
148 75c0332f8 ff35e8ccd275    push    dword ptr [combase!g_hinst (75d2ccee8)]
148 75c033302 57              push    edi
148 75c033303 6afd            push    0FFFFFFFFH
push eax
push eax
push eax
push eax
push 88000000h
push offset combase!'string' (75d342e8)
push DWORD PTR [combase!ghwndOleMainThread (75d2d340)], eax
je combase!InitMainThreadWnd+0x8a3e9 (75c8d6cc) Branch
mov esi,edi
mov eax,esi,esi
mov esi,edi
xor ecx,ecx
call combase!ComTraceMessage (75b60469)
add esp,1Ch
mov esi,edi
mov esi,edi
mov esi,edi
mov esi,edi
add esp,1Ch
mov esi,edi
To remind you, the API function definition is:

```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
);
```

We see that all parameters are pushed right-to-left with the third parameter from the left containing the 64-bit pointer to a window name (UNICODE):

```bash
0:000> du 75b0f070
75b0f070 "OleMainThreadWndName"
```

14. We close logging before exiting WinDbg Preview:

```bash
0:000> .logclose
Closing open log file C:\AWAPI-Dumps\W1.log
```