

# An Alternative Method To Enumerate Processes

vx-underground collection // [smelly\\_vx](#)



# Introduction

I was exploring MSDN. I genuinely enjoy exploring the [Windows API reference](#) and looking for APIs which could potentially be used for malware. I found an interesting way to enumerate processes, without heavy reliance on the [Process Status API \(PSAPI\)](#) and I've decided to write a *micro paper* about it. This technique is not particularly complex and **does not** successfully enumerate all processes.

## **What this paper will discuss:**

This paper will illustrate an alternative way to enumerate *some* processes.

## **What this paper will not discuss:**

This paper will not show how authors typically use PSAPI to enumerate processes. There is a lot of documentation on it about it online using both [EnumProcesses](#) [1] and the alternative [Tool Help 32](#) option of [CreateToolhelp32Snapshot](#) in combination with [Process32First](#) and [Process32Next](#) [2].

# Using Windows Controls for Process Enumeration

MSDN supplies developers with a rather large set of APIs for GUI creation with the [Windows and Messages API subset](#). This API focuses almost exclusively on design and callback routines (as the name suggests). However, this API does offer some pretty interesting functions such as [EnumWindows](#). Per MSDN: *Enumerates all top-level windows on the screen by passing the handle to each window, in turn, to an application-defined callback function. EnumWindows continues until the last top-level window is enumerated or the callback function returns FALSE.*

```
BOOL EnumWindows(  
    WNDENUMPROC lpEnumFunc,  
    LPARAM      lParam  
);
```

The function is really simple. As you can see, this functions callback routine returns a HANDLE to each Window from an application. You can use this in conjunction with [GetWindowThreadProcessId](#) to get the process identifier from the callback routine from EnumWindows. GetWindowThreadProcessId is defined as such:

```
DWORD GetWindowThreadProcessId(  
    HWND    hWnd,  
    LPDWORD lpdwProcessId  
);
```

After successfully retrieving the process identifier from GetWindowsThreadProcessId simply invoke [OpenProcess](#) with the [PROCESS\\_QUERY\\_INFORMATION](#) flags because our subsequent call to [GetProcessImageFileName](#) can only be called when the [process handle has the PROCESS\\_QUERY\\_INFORMATION access right](#). GetProcessImageFileName is defined as such:

```
DWORD GetProcessImageFileNameA(  
    HANDLE hProcess,  
    LPSTR  lpImageFileName,  
    DWORD  nSize  
);
```

That's it. It's simple.

# Code

```
#include <Windows.h>
#include <stdio.h>
#include <Psapi.h>

BOOL CALLBACK EnumWindowsProc(HWND hWnd, LPARAM lParam);

int main(VOID)
{
    EnumWindows(EnumWindowsProc, 0);
}

BOOL CALLBACK EnumWindowsProc(HWND hWnd, LPARAM lParam)
{
    WCHAR wcPath[MAX_PATH] = { 0 };
    DWORD dwThreadId = 0;
    HANDLE hHandle;

    GetWindowThreadProcessId(hWnd, &dwThreadId);

    hHandle = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, dwThreadId);
    if (hHandle == NULL)
        goto FAILURE;

    if (GetProcessImageFileNameW(hHandle, wcPath, MAX_PATH) == 0)
        goto FAILURE;

    printf("%ws\r\n", wcPath);

    if (hHandle)
        CloseHandle(hHandle);

    return TRUE;

FAILURE:

    if (hHandle)
        CloseHandle(hHandle);

    return FALSE;
}
```

# Output

This code will enumerate all windows. The code will contain a lot of duplicates depending on how many windows the application in question has allocated. Here is a sanitized sample of the output.

```
\Device\HarddiskVolume2\Windows\explorer.exe  
\Device\HarddiskVolume2\Windows\System32\cmd.exe  
\Device\HarddiskVolume2\Windows\System32\RuntimeBroker.exe  
\Device\HarddiskVolume2\Windows\ImmersiveControlPanel\SystemSettings.exe  
\Device\HarddiskVolume2\Windows\System32\ApplicationFrameHost.exe  
\Device\HarddiskVolume2\Users\dwThr\AppData\Local\Microsoft\OneDrive\OneDrive.exe  
\Device\HarddiskVolume2\Windows\System32\RuntimeBroker.exe  
\Device\HarddiskVolume2\Windows\System32\SecurityHealthSystray.exe  
\Device\HarddiskVolume2\Windows\System32\SettingSyncHost.exe  
\Device\HarddiskVolume2\Windows\System32\MicrosoftEdgeCP.exe  
\Device\HarddiskVolume2\Windows\System32\ApplicationFrameHost.exe  
\Device\HarddiskVolume2\Windows\System32\svchost.exe  
\Device\HarddiskVolume2\Windows\System32\taskhostw.exe
```