Get-ADS / Get-ADS.ps1

p0shkatz Create Get-ADS.ps1 bff2fa1 · 7 years ago History

Code Blame 178 lines (143 loc) · 6.02 KB Raw

```
<#
.SYNOPSIS

This script searches recursively through a specified file system for alternate data str

.DESCRIPTION

The script can search local and UNC paths speciffied by the $path paramenter. All reada
attrubute inspected ignoring the default DATA and FAVICON (image file on URL files) str
amazing Get-RunspaceData function and other code to multithread the search. The default
number of logical cores plus one. This can be adjusted by specifiying the $threads para
runspaces can easily chomp resources (CPU and RAM).

Once the number of file system objects (files and folders) is determined, they are spli
divided by the number of threads. Then each thread has a subset of the total objects to

Author: Michael Garrison (@p0shkatz)
License: MIT

.PARAMETER path

This is a required parameter that sets the base or root path to search from, for exampl

.PARAMETER output

This is an optionaal parameter that sets an output location for the results, for exampl

.PARAMETER threads

This is an optional parameter that sets the number of threads to run concurrently.

.EXAMPLE

Get-ADS.ps1 -Path C:\

.EXAMPLE

Get-ADS.ps1 -Path C:\ -Threads 16

.EXAMPLE

Get-ADS.ps1 -Path \\servername\sharename -Output \\servername\sharename\ads-report.log

#>

Param
(
    [parameter(Mandatory=$true,
    ValueFromPipeline=$true,
    HelpMessage="Supply the root path (e.g. C:\)")]
    [ValidateScript({(Test-Path $_)})]
    [String[]]$Path,

    [parameter(Mandatory=$false,
    HelpMessage="Supply the full path to an output file")]
    [ValidateScript({(Test-Path $_.SubString(0,$_.LastIndexOf("\")))})]
    [String[]]$Output
```

```
        [string[]]$Output,

        [parameter(Mandatory=$false,
        HelpMessage="Supply the number of threads to use")]
        [int]$Threads
    )

    Function Get-RunspaceData {
        [cmdletbinding()]
        param(
            [switch]$Wait
        )
        Do {
            $more = $false
            Foreach($runspace in $runspaces) {
                If ($runspace.Runspace.isCompleted) {
                    $runspace.powershell.EndInvoke($runspace.Runspace)
                    $runspace.powershell.dispose()
                    $runspace.Runspace = $null
                    $runspace.powershell = $null
                } ElseIf ($runspace.Runspace -ne $null) {
                    $more = $true
                }
            }
            If ($more -AND $PSBoundParameters['Wait']) {
                Start-Sleep -Milliseconds 100
            }
            # Clean out unused runspace jobs
            $temphash = $runspaces.clone()
            $temphash | Where {
                $_.runspace -eq $Null
            } | ForEach {
                $Runspaces.remove($_)
            }
            $Remaining = ((@($runspaces | Where {$_.Runspace -ne $Null}).Count))

        } while ($more -AND $PSBoundParameters['Wait'])
    }

    $ScriptBlock = {
        Param ($group, $hash)
        $i=1
        foreach($item in $group.Group)
        {
            Write-Progress `
                -Activity "Searching through group $($group.Name)" `
                -PercentComplete (($i / $group.Count) * 100) `
                -Status "$($group.count - $i) remaining of $($group.count)" `
                -Id $($group.Name)
            $streams = Get-Item $item.FullName -stream *
            foreach($stream in $streams.Stream)
            {
                # Ignore DATA and favicon streams
                if($stream -ne ':$DATA' -and $stream -ne 'favicon')
                {
                    $streamData = Get-Content -Path $item.FullName -stream $stream
                    $hash[$item.FullName] = "Stream name: $stream`nStream data: $streamData
                }
            }
            $i++
        }
    }

    if($threads){$threadCount = $threads}
    # Number of threads defined by number of cores + 1
    else{$threadCount = (Get-WmiObject -class win32_processor | select NumberOfLogicalProce

    $Script:runspaces = New-Object System.Collections.ArrayList
    $hash = [hashtable]::Synchronized(@{})
    $sessionstate = [system.management.automation.runspaces.initialsessionstate]::CreateDef
    $runspacepool = [runspacefactory]::CreateRunspacePool(1, $threadCount, $sessionstate, $
    $runspacepool.Open()

    # Ignore read errors
    $ErrorActionPreference = 'silentlycontinue'
```

```
132     Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Retrieving collection of file system ob
133     $items = Get-ChildItem $Path -recurse
134     $counter = [pscustomobject] @{ Value = 0 }
135     $groupSize = $items.Count / $threadCount
136     Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Collected $($items.count) file system o
137     $groups = $items | Group-Object -Property { [math]::Floor($counter.Value++ / $groupSize
138     Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Searching for alternate data streams...
139     foreach ($group in $groups)
140     {
141         # Create the powershell instance and supply the scriptblock with the other paramete
142         $powershell = [powershell]::Create().AddScript($scriptBlock).AddArgument($group).Ad
143
144         # Add the runspace into the powershell instance
145         $powershell.RunspacePool = $runspacepool
146
147         # Create a temporary collection for each runspace
148         $temp = "" | Select-Object PowerShell,Runspace,Group
149         $Temp.Group = $group
150         $temp.PowerShell = $powershell
151
152         # Save the handle output when calling BeginInvoke() that will be used later to end
153         $temp.Runspace = $powershell.BeginInvoke()
154         $runspaces.Add($temp) | Out-Null
155     }
156
157     Get-RunspaceData -Wait
158
159     Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Completed"
160
161     $hash.GetEnumerator() | Format-List
162
163     if($output){
164         Write-Host "Writing output to $output"
165         $fileStream = New-Object System.IO.StreamWriter $output
166         $fileStream.WriteLine("Alternate Data Streams")
167         $hash.GetEnumerator() | foreach{
168             $fileStream.WriteLine("$($_.Name)`r`n$($_.Value)")
169         }
170         $fileStream.Close()
171     }
172     # Clean up
173     $powershell.Dispose()
174     $runspacepool.Close()
175
176     [System.GC]::Collect()
177     [System.GC]::WaitForPendingFinalizers()
178     [System.GC]::Collect()
```