Sign in

p0shkatz / Get-ADS Public

Notifications Fork 1 Star 2

Code Issues Pull requests Actions Projects Security Insights

Get-ADS / Get-ADS.ps1

178 lines (143 loc) · 6.02 KB

Code Blame Raw

```
<#
.SYNOPSIS

This script searches recursively through a specified file system for alternate data streams (ADS).

.DESCRIPTION

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

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

Author: Michael Garrison (@p0shkatz)
License: MIT

.PARAMETER path

This is a required parameter that sets the base or root path to search from, for example C:\ or \\s

.PARAMETER output

This is an optionaal parameter that sets an output location for the results, for example C:\ads-dat
```

```

.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,

    [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 NumberOfLogicalProcessors).Numbe

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

# Ignore read errors
$ErrorActionPreference = 'silentlycontinue'
Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Retrieving collection of file system objects..."
$items = Get-ChildItem $Path -recurse
$counter = [pscustomobject] @{ Value = 0 }
$groupSize = $items.Count / $threadCount
Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Collected $($items.count) file system objects. Spli
$groups = $items | Group-Object -Property { [math]::Floor($counter.Value++ / $groupSize) }
Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Searching for alternate data streams..."
foreach ($group in $groups)
{
    # Create the powershell instance and supply the scriptblock with the other parameters
    $powershell = [powershell]::Create().AddScript($scriptBlock).AddArgument($group).AddArgument($h

    # Add the runspace into the powershell instance
    $powershell.RunspacePool = $runspacepool

    # Create a temporary collection for each runspace
    $temp = "" | Select-Object PowerShell,Runspace,Group
    $Temp.Group = $group
    $temp.PowerShell = $powershell

    # Save the handle output when calling BeginInvoke() that will be used later to end the runspace
    $temp.Runspace = $powershell.BeginInvoke()
    $runspaces.Add($temp) | Out-Null
}

Get-RunspaceData -Wait

Write-Host "$(Get-Date -F MM-dd-yyyy-HH:mm:ss)::Completed"

$hash.GetEnumerator() | Format-List

if($output){
    Write-Host "Writing output to $output"
    $fileStream = New-Object System.IO.StreamWriter $output
    $fileStream.WriteLine("Alternate Data Streams")
    $hash.GetEnumerator() | foreach{
        $fileStream.WriteLine("$($_.Name)`r`n$($_.Value)")
    }
    $fileStream.Close()
}
# Clean up
$powershell.Dispose()
$runspacepool.Close()

[System.GC]::Collect()
[System.GC]::WaitForPendingFinalizers()
[System.GC]::Collect()
```