BloodHound / Collectors / AzureHound.ps1

1500 lines (1215 loc) · 58.2 KB

```
# AzureHound Beta
# Authors: Andy Robbins (@_wald0), Rohan Vazarkar (@cptjesus), Ryan Hausknecht (@haus3c)
# Copyright: SpecterOps, Inc. 2020

function Get-PrincipalMap {

    $PrincipalMap = @{}
    Get-AzureADUser -All $True | % {
        $PrincipalMap.add($_.objectid, $_.OnPremisesSecurityIdentifier)
    }
    Get-AzureADGroup -All $True | % {
        $PrincipalMap.add($_.objectid, $_.OnPremisesSecurityIdentifier)
    }
    $PrincipalMap
}
function Connect-AADUser {
    $ConnectionTest = try{ [Microsoft.Open.Azure.AD.CommonLibrary.AzureSession]::AccessTokens['Acce
    catch{"Error"}
    If($ConnectionTest -eq 'Error'){
    $context = [Microsoft.Azure.Commands.Common.Authentication.Abstractions.AzureRmProfileProvider]
    $aadToken = [Microsoft.Azure.Commands.Common.Authentication.AzureSession]::Instance.Authenticat
    Connect-AzureAD -AadAccessToken $aadToken -AccountId $context.Account.Id -TenantId $context.ter
}

function Get-AzureGraphToken
{
    $APSUser = Get-AzContext *>&1
    $resource = "https://graph.microsoft.com"
    $Token = [Microsoft.Azure.Commands.Common.Authentication.AzureSession]::Instance.Authenticatior
    $Headers = @{}
    $Headers.Add("Authorization","Bearer"+ " " + "$($token)")
    $Headers
}

$Verbose = $True
function Write-Info ($Message) {
    If ($Verbose) {
        Write-Host $Message
    }
}

function New-Output($Coll, $Type, $Directory) {

    $Count = $Coll.Count

    Write-Host "Writing output for $($Type)"
        if ($null -eq $Coll) {
        $Coll = New-Object System.Collections.ArrayList
    }

    # ConvertTo-Json consumes too much memory on larger objects, which can have millions
    # of entries in a large tenant. Write out the JSON structure a bit at a time to work
    # around this. This is a bit inefficient, but makes this work when the tenant becomes
    # too large.
    $FileName = $Directory + [IO.Path]::DirectorySeparatorChar + $date + "-" + "az" + $($Type) + ".
    try {
        $Stream = [System.IO.StreamWriter]::new($FileName)

        # Write file header JSON
        $Stream.WriteLine('{')
        $Stream.WriteLine("`t""meta"": {")
        $Stream.WriteLine("`t`t""count"": $Count,")
        $Stream.WriteLine("`t`t""type"": ""az$($Type)"",")
        $Stream.WriteLine("`t`t""version"": 4")
        $Stream.WriteLine("`t},")

        # Write data JSON
        $Stream.WriteLine("`t""data"": [")
        $Stream.Flush()

        $chunksize = 250
        $chunkarray = @()
        $parts = [math]::Ceiling($coll.Count / $chunksize)

        Write-Info "Chunking output in $chunksize item sections"
        for($n=0; $n -lt $parts; $n++){
            $start = $n * $chunksize
            $end = (($n+1)*$chunksize)-1
            $chunkarray += ,@($coll[$start..$end])
        }
        $Count = $chunkarray.Count

        $chunkcounter = 1
        $jsonout = ""
        ForEach ($chunk in $chunkarray) {
            Write-Info "Writing JSON chunk $chunkcounter/$Count"
            $jsonout = ConvertTo-Json($chunk)
            $jsonout = $jsonout.trimstart("[`r`n").trimend("`r`n]")
            $Stream.Write($jsonout)
            If ($chunkcounter -lt $Count) {
                $Stream.WriteLine(",")
            } Else {
                $Stream.WriteLine("")
            }
            $Stream.Flush()
            $chunkcounter += 1
        }
        $Stream.WriteLine("`t]")
        $Stream.WriteLine("}")
    } finally {
        $Stream.close()
    }
}

function Invoke-AzureHound {
    [CmdletBinding()]
    Param(
    [Parameter(Mandatory=$False)][String]$TenantID = $null,
    [Parameter(Mandatory=$False)][String]$OutputDirectory = $(Get-Location),[ValidateNotNullOrEmpty
    [Parameter(Mandatory=$False)][Switch]$Install = $null)

    if ($Install){
      Install-Module -Name Az -AllowClobber
      Install-module -Name AzureADPreview -AllowClobber
    }

    $Modules = Get-InstalledModule
    if ($Modules.Name -notcontains 'Az.Accounts' -and $Modules.Name -notcontains 'AzureAD'){
```

```
        if ($Modules.Name -notcontains 'Az.Accounts' -and $Modules.Name -notcontains 'AzureAD'){
```

```
                $null = $Coll.Add($AppRight)

            }
        }
        New-Output -Coll $Coll -Type "applicationadmins" -Directory $OutputDirectory
            Write-Info "Done processing Application Admins"

        # Cloud Application Admins - Can create new secrets for application service principals
        # Write to cloudappadmins.json
            Write-Info "Processing Cloud Application Admins"
        $Coll = New-Object System.Collections.ArrayList
        $CloudAppAdmins = $UserRoles | Where-Object {$_.RoleID -match '158c047a-c907-4556-b7ef-446551a6
        $SPsWithAzureAppAdminRole = $UserRoles | Where-Object {$_.RoleID -match '158c047a-c907-4556-b7e
        $AppsWithAppAdminRole = ForEach ($SP in $SPsWithAzureAppAdminRole) {
            $AppWithRole = $SPOS | ?{$_.ServicePrincipalID -Match $SP.UserID}
            $AppWithRole
        }
        $CloudAppAdminRights = ForEach ($Principal in $AppAdmins) {

            $TargetApps = $AppsWithAppAdminRole

            ForEach ($TargetApp in $TargetApps) {

                $AppRight = [PSCustomObject]@{
                    AppAdminID          = $Principal.UserID
                    AppAdminType        = $Principal.UserType
                    AppAdminOnPremID    = $Principal.UserOnPremID
                    TargetAppID         = $TargetApp.AppID
                }

                $null = $Coll.Add($AppRight)

            }

            ForEach ($TargetApp in $SPswithoutRoles) {

                $AppRight = [PSCustomObject]@{
                    AppAdminID          = $Principal.UserID
                    AppAdminType        = $Principal.UserType
                    AppAdminOnPremID    = $Principal.UserOnPremID
                    TargetAppID         = $TargetApp.AppID
                }

                $null = $Coll.Add($AppRight)
            }
        }
        New-Output -Coll $Coll -Type "cloudappadmins" -Directory $OutputDirectory
            Write-Info "Done processing Cloud Application Admins"

        Write-Host "Compressing files"
        $location = Get-Location
        $name = $date + "-azurecollection"
        If($OutputDirectory.path -eq $location.path){
            $jsonpath = $OutputDirectory.Path + [IO.Path]::DirectorySeparatorChar + "$date-*.json"
            $destinationpath = $OutputDirectory.Path + [IO.Path]::DirectorySeparatorChar + "$name.zip"
        }
        else{
            $jsonpath = $OutputDirectory + [IO.Path]::DirectorySeparatorChar + "$date-*.json"
            $destinationpath = $OutputDirectory + [IO.Path]::DirectorySeparatorChar + "$name.zip"
        }

        $error.Clear()
        try {
            Compress-Archive $jsonpath -DestinationPath $destinationpath
        }
        catch {
```

```
        catch {
            Write-Host "Zip file creation failed, JSON files may still be importable."
        }
        if (!$error) {
            Write-Host "Zip file created: $destinationpath"
            rm $jsonpath
            Write-Host "Done! Drag and drop the zip into the BloodHound GUI to import data."
        }
    }
```