Hamburg PowerShell UserGroup

PowerShell Usergroup Hamburg

PowerShell Module Development on GitHub

Speaker

Jan-Henrik Damaschke

CTO / Senior Cloud Architect @ Visorian

Microsoft MVP for Azure

@JanDamaschke

/in/jan-henrik-damaschke

itinsights.org

Content

1. PSMDE module overview
2. PowerShell modules
3. Source folder
4. Feature focus
5. Documentation
6. Repository structure
7. Automation CI/CD
8. Questions?

PSMDE module overview

PSMDE is a wrapper module for the Microsoft Defender for Endpoint (MDE) API.

ci passing | powershell gallery v0.29.0 | coverage 93.75%

- **Native PowerShell wrapper** - JSON parsing, working with PS objects, piping
- **Token refresh** - Automatically refresh token if expired or about to expire
- **429 handling** - Exponential backoff mechanism implemented
- **Permission check** - Verify permissions on a per API basis
- **Documentation** - Automatic generated documentation and PowerShell help files
- **CI/CD** - Automatic PR testing, versioning and releasing
- **Fully tested** - 100% test coverage of public functions, >95% overall
- **Security** - Saves temporary values encrypted, helps with token scopes

PowerShell modules

- PowerShell modules are collections of commands that can be used to perform specific tasks or automate processes
- Stored in .psm1 or .psd1 files
- They can be written in a variety of languages, including PowerShell, C#, and Python
- Normally shared and distributed through the PowerShell Gallery or by using NuGet package manager
- To create a module, use the New-ModuleManifest command and provide metadata information like Author, Company, Description and also the version
- To export functions/cmdlets from module, use the Export-ModuleMember command
- To interact with modules, use `Import-Module`, `Remove-Module`, `Get-Module` and `Update-Module`

Source folder

- Manifest file (`PSMDE.psd1`)
- Module file (`PSMDE.psm1`)
- `private` folder
- `public` folder

Source folder

Module file (`PSMDE.psd1`)

```
# Declare module variables

New-Variable -Name tenantId -Value $null -Scope Script -Force
New-Variable -Name appId -Value $null -Scope Script -Force
New-Variable -Name appSecret -Value $null -Scope Script -Force
New-Variable -Name tokenCache -Value $null -Scope Script -Force
New-Variable -Name initialize -Value $false -Scope Script -Force

# Import private and public scripts and expose the public ones

$privateScripts = @(Get-ChildItem -Path "$PSScriptRoot\private" -Recurse -Filter "*.ps1") | Sort-Object Name
$publicScripts = @(Get-ChildItem -Path "$PSScriptRoot\public" -Recurse -Filter "*.ps1") | Sort-Object Name

foreach ($script in ($privateScripts + $publicScripts)) {
  Write-Verbose $script
  try {
      . $script
      Write-Verbose -Message ("Imported function {0}" -f $script)
  } catch {
      Write-Error -Message ("Failed to import function {0}: {1}" -f $script, $_)
  }
}
```

Source folder

Module file (`PSMDE.psd1`)

```
# Add PSReadLine handler to avoid credentials being saved in the command history

$scriptBlock = {
  param(
    [string]
    $line
  )
  return $line.ToLower().StartsWith('set-mdeauthorizationinfo') ? $false : $true
}
Set-PSReadlineOption -AddToHistoryHandler $scriptBlock

Export-ModuleMember -Function $publicScripts.BaseName
```

Feature focus

Encryption (`New-AesSessionSecret.ps1`)

```
$keyBytes = [Text.Encoding]::ASCII.GetBytes((Get-Process -PID $pid).ProcessorAffinity.ToString() + (Get-Process -P
$encryptionKey = [System.Byte[]]::new(32)
for ($i = 0; $i -lt $keyBytes.length; $i++) {
  $encryptionKey[$i] = $keyBytes[$i]
}
$rngCryptoServiceProvider = New-Object System.Security.Cryptography.RNGCryptoServiceProvider
$initializationVector = [System.Byte[]]::new(16)
$rngCryptoServiceProvider.GetBytes($initializationVector)
$aesCryptoServiceProvider = New-Object System.Security.Cryptography.AesCryptoServiceProvider
$aesCryptoServiceProvider.Key = $encryptionKey
$aesCryptoServiceProvider.IV = $initializationVector
$secretBytes = [System.Text.Encoding]::UTF8.GetBytes($secret)
$encryptor = $aesCryptoServiceProvider.CreateEncryptor()
$encryptedBytes = $encryptor.TransformFinalBlock($secretBytes, 0, $secretBytes.Length)
[byte[]] $data = $aesCryptoServiceProvider.IV + $encryptedBytes

$aesCryptoServiceProvider.Dispose()
$rngCryptoServiceProvider.Dispose()

return [System.Convert]::ToBase64String($data)
```

Feature focus

Encryption (`Get-AesSessionSecret.ps1`)

```
$keyBytes = [Text.Encoding]::ASCII.GetBytes((Get-Process -PID $pid).ProcessorAffinity.ToString() + (Get-Process -P
$encryptionKey = [System.Byte[]]::new(32)
for ($i = 0; $i -lt $keyBytes.length; $i++) {
  $encryptionKey[$i] = $keyBytes[$i]
}
$encryptedBytes = [System.Convert]::FromBase64String($cipherText)
$aesCryptoServiceProvider = New-Object System.Security.Cryptography.AesCryptoServiceProvider
$aesCryptoServiceProvider.Key = $encryptionKey
$aesCryptoServiceProvider.IV = $encryptedBytes[0..15]
$decryptor = $aesCryptoServiceProvider.CreateDecryptor();
$secretBytes = $decryptor.TransformFinalBlock($encryptedBytes, 16, $encryptedBytes.Length - 16)
$decryptedSecret = [System.Text.Encoding]::UTF8.GetString($secretBytes)

$aesCryptoServiceProvider.Dispose()
return $decryptedSecret
```

Feature focus

Token refresh (`Get-MdeAuthorizationHeader.ps1`)

```
$tc = $script:tokenCache ? (Get-AesSessionSecret -cipherText $script:tokenCache) : $null

$expired = $tc ? [DateTime]::Now -gt [TimeZoneInfo]::ConvertTimeFromUtc(
  ([DateTime]::UnixEpoch).AddSeconds(
    (Get-ParsedToken -token $tc).exp), [TimeZoneInfo]::FindSystemTimeZoneById('Central European Standard Time')
) : $true
```

Feature focus

Exponential backoff (`Invoke-RetryRequest.ps1`)

```
$headers = Get-MdeAuthorizationHeader
do {
  try {
    $retry = $false
    if (@('put', 'patch', 'post') -contains $method.ToLower()) {
      return Invoke-RestMethod -Method $method -Headers $headers -ContentType 'application/json' -Uri $uri -Body $
    }
    else {
      return Invoke-RestMethod -Method $method -Headers $headers -Uri $uri -ErrorAction Stop
    }
  }
  catch {
    if ($_.Exception.Response.StatusCode.value__ -ne 429) { $retry = $false; $_; break }
    $sleepDuration = $sleepDuration -eq 0 ? 4 : $sleepDuration * 2
    $retry = $true
    Write-Verbose "API returned 429, retrying in $sleepDuration seconds"
    Start-Sleep -Seconds $sleepDuration
  }
} until (
  -not $retry
)
```

Feature focus

Permission check (`Test-MdePermissions.ps1`)

```
$roles = (Get-MdeAuthorizationInfo).roles
$requiredRoles = (Get-Help $functionName -Full).role | Invoke-Expression
$containsRole = $false
foreach ($role in $requiredRoles) {
  $evaluation = $roles.contains($role.permission)
  Write-Verbose "Checking for '[$($role.permissionType)] $($role.permission)': $evaluation"
  $containsRole = $containsRole -or $evaluation
}
```

Tests

- Pester tests
- One test file per unit test
- Separated in public and private
- General tests located in `Functions.Tests.ps1`

Tests

Example

```
BeforeAll {
  Remove-Module PSMDE -Force -ErrorAction SilentlyContinue
  Import-Module (Split-Path $PSCommandPath).replace('tests', 'src').Replace('public', 'PSMDE.psd1')
}

Describe "Enable-MdeMachineIsolation" {

  It 'Should have the PSMDE module loaded' {
    $module = Get-Module PSMDE
    $module | Should -Not -BeNullOrEmpty
  }

  It 'Should have access to internal functions' {
    InModuleScope PSMDE {
      $iar = Get-Command Invoke-AzureRequest
      $iar | Should -Not -BeNullOrEmpty
    }
  }
```

Tests

Example

```
  It 'Should correctly create the request uri' {
    InModuleScope PSMDE {
      Mock Invoke-RetryRequest { return $uri }
      Mock Test-MdePermissions { return $true }
      $id = '12345'
      $comment = 'Comment'
      Enable-MdeMachineIsolation -id $id -comment $comment | Should -Be "https://api.securitycenter.microsoft.com/
    }
  }
}
```

Documentation

- Public (external) help
- Markdown help
- Uses platyPS
- Pushed to GitHub integrated wiki

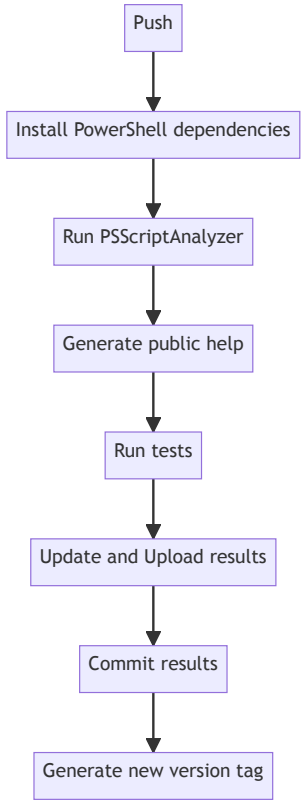
```
Import-Module .\src\PSMDE.psd1
Update-MarkdownHelpModule -Path 'wiki' -RefreshModulePage -UpdateInputOutput -Force
(Get-Content .\wiki\PSMDE.md) -replace '(.md\))', ')' | Out-File .\wiki\PSMDE.md -Force -Encoding ascii
New-ExternalHelp -Path 'wiki' -OutputPath 'en-us' -Force
```

Repository structure

- *.github/workflows/* - Workflows for CI, releasing and testing PRs
- *en-us/* - `PSMDE-help.xml` PowerShell external help file
- *icon* - Public icon folder for PS-Gallery
- *scripts/* - Helper functions for module developers/contributors
- *src/* - Source files
- *tests/* - Pester unit tests
- *wiki* - Git submodule linking to wiki repo for generated markdown help

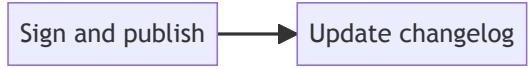
Automation CI/CD

CI Action



Automation CI/CD

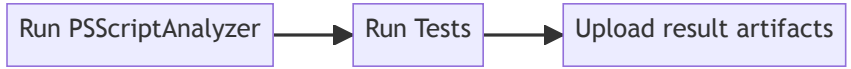
Release Action




```
$certPath = Join-Path -Path $PSScriptRoot -ChildPath "code_signing.pfx"
Set-Content -Value $([System.Convert]::FromBase64String($env:SIGNING_CERTIFICATE)) -Path $certPath -AsByteStream
$cert = Import-PfxCertificate -FilePath $certPath -Password ($env:SIGNING_PASSWORD | ConvertTo-SecureString -AsPla
Get-ChildItem src -Recurse -Force -Filter *.ps* | Set-AuthenticodeSignature -Certificate $cert -TimestampServer 'h
Copy-Item .\src\ -Recurse -Destination .\PSMDE\ -Force
Publish-Module -Path .\PSMDE\ -NuGetApiKey $env:NUGET_KEY
```

Automation CI/CD

Test Action



Questions?

Go ahead and ask!