Full working set for calendar API

Friday, July 25, 2025 9:41 AM

The solution is split into two main scripts:

1. Calendar Event Creation Script – Automates the creation of a Microsoft Teams calendar event using Microsoft Graph API.
2. Service Principal and App Registration Script – Automates the provisioning of an Azure AD application and service principal with secure credential storage in Azure Key Vault.

1. Calendar Event Creation via Microsoft Graph

Authentication & Setup

- Modules Used:
- Microsoft.Graph.Calendar
- Az.Accounts
- Key Steps:
- Connect to Azure using Connect-AzAccount.
- Set the subscription context with Set-AzContext.
- Connect to Microsoft Graph using Connect-MgGraph with delegated permissions:
- Application.ReadWrite.All
- AppRoleAssignment.ReadWrite.All

Azure Key Vault Integration

- Vault Name: wolffofficekvkv2
- Resource Group: Adminrg
- App Registration: WOLFFCalendarManagerAppspn
- Key Vault Commands:
- Update-AzKeyVaultNetworkRuleSet
- Update-AzKeyVault
- Get-AzKeyVaultSecret
- Dynamic IP whitelisting if access is denied.

Token Acquisition

- Uses Invoke-RestMethod to call:
  https://login.microsoftonline.com/<tenantId>/oauth2/v2.0/token

  with client_credentials grant type.

Calendar Event Definition

- Creates a Teams meeting titled “POC Strategy Sync” for jerrywolff@wpi-corp.com.
- Event includes:
- HTML body
- Online meeting via Teams
- Attendee list
- Uses Invoke-RestMethod to POST to:
  https://graph.microsoft.com/v1.0/users/<userEmail>/events

2. Service Principal & App Registration Script

App Registration

- Modules Used:
- Microsoft.Graph.Applications
- Az.Resources
- Az.KeyVault
- Az.AD
- Key Steps:
- Create app with New-AzADApplication.
- Assign ownership (commented out).
- Create service principal with New-AzADServicePrincipal.

Graph API Permissions Assigned

- Calendars.Read
- Calendars.ReadWrite
- MailboxSettings.Read
- User.Read.All

Metadata Recording

- Captures:
- Tenant ID
- Client ID
- Secret
- Key ID
- Validity dates

Key Vault Configuration

- Adds public and automation IPs to firewall rules.
- Sets access policies using Set-AzKeyVaultAccessPolicy.
- Stores client secret using Set-AzKeyVaultSecret.

calendar_create_response

Error Handling

- If access is denied:
- Extracts client IP from error.
- Adds IP to Key Vault firewall.
- Retries secret creation.

Admin Consent

- Outputs a URL for admin consent:
  https://login.microsoftonline.com/<tenantId>/adminconsent?client_id=<appId>

Microsoft-Specific References

PowerShell Modules

- Microsoft.Graph.Calendar
- Microsoft.Graph.Applications
- Az.Accounts
- Az.Resources
- Az.KeyVault
- Az.AD

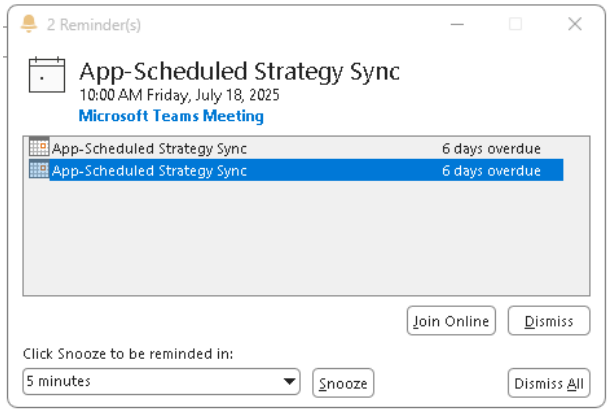
Microsoft Graph API Endpoints

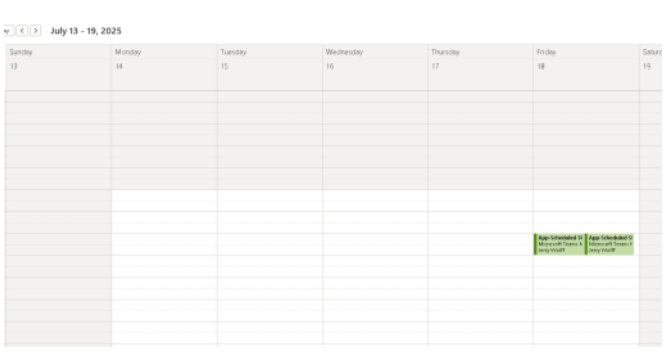
- https://graph.microsoft.com/v1.0/users/<userEmail>/events
- https://login.microsoftonline.com/<tenantId>/oauth2/v2.0/token
- https://login.microsoftonline.com/<tenantId>/adminconsent?client_id=<appId>

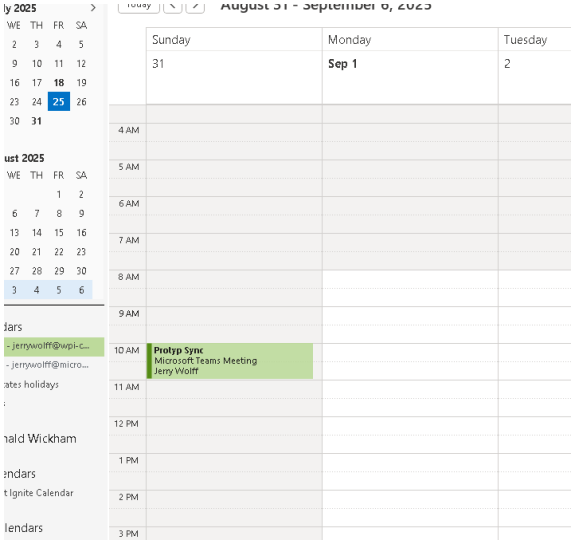
Documentation Links

- https://learn.microsoft.com/en-us/graph/api/resources/calendar?view=graph-rest-1.0
- https://learn.microsoft.com/en-us/powershell/module/az.keyvault/
- https://learn.microsoft.com/en-us/powershell/microsoftgraph/overview

From <https://m365.cloud.microsoft/chat/?fromcode=edgentp&redirectid=c301510486c841f18ee34c26d9cb78d1&auth=2&handinClickTs=1753462364804&internalredirect=CCM>







calendar_create_res...

```
@odata.context             :
https://graph.microsoft.com/v1.0/$metadata#users('jerrywolff%40wpi-corp.com')/events
                             /$entity
@odata.etag                : W/"Qc4jJH79qU+05CagRiRfmgAAAAEL/A=="
id                         :
AQMkADNjYmM5MTk4LWZlZjgtNDZlZi1iYgA1Ni1mZGNlNzhlYTBkZjMARgAAA6f3lv0BuLJDh1QtOUecWRYH

AEHOIyR_-alPtOQmoEYkX5oAAAIBDQAAAEHOIyR_-alPtOQmoEYkX5oAAAKWCgAAAA==
createdDateTime            : 2025-07-24T20:29:24.5194584Z
lastModifiedDateTime       : 2025-07-24T20:29:27.0414315Z
changeKey                  : Qc4jJH79qU+05CagRiRfmgAAAAEL/A==
categories                 : {}
transactionId              :
originalStartTimeZone      : Pacific Standard Time
originalEndTimeZone        : Pacific Standard Time
iCalUId                    :
040000008200E00074C5B7101A82E00800000000DF4C14A5D9FCDB0100000000000000000010000000C474
                             821F8D9B9F42B88E25D44A8E6C9A
uid                        :
040000008200E00074C5B7101A82E00800000000DF4C14A5D9FCDB0100000000000000000010000000C474
                             821F8D9B9F42B88E25D44A8E6C9A
reminderMinutesBeforeStart : 15
isReminderOn               : True
hasAttachments             : False
subject                    : Protyp Sync
bodyPreview                : This meeting was scheduled using application
permissions.

_____________________________________________________________________________
                             Microsoft Teams Need help?
                             Join the meeting now
                             Meeting ID: 282 309 443 863 6
                             Passcode: uv9ck6w2
                             ____________
importance                 : normal
sensitivity                : normal
isAllDay                   : False
isCancelled                : False
isOrganizer                : True
responseRequested          : True
seriesMasterId             :
showAs                     : busy
type                       : singleInstance
webLink                    :
https://outlook.office365.com/owa/?itemid=AQMkADNjYmM5MTk4LWZlZjgtNDZlZi1iYgA1Ni1mZG

NlNzhlYTBkZjMARgAAA6f3lv0BuLJDh1QtOUecWRYHAEHOIyR%2B%2FalPtOQmoEYkX5oAAAIBDQAAAEHOIy
```

validate_access_to_calendars

```
<#
.NOTES

    THIS CODE-SAMPLE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED

    OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR

    FITNESS FOR A PARTICULAR PURPOSE.

    This sample is not supported under any Microsoft standard support program or service.
```

The script is provided AS IS without warranty of any kind. Microsoft further disclaims all

implied warranties including, without limitation, any implied warranties of merchantability

or of fitness for a particular purpose. The entire risk arising out of the use or performance

of the sample and documentation remains with you. In no event shall Microsoft, its authors,

or anyone else involved in the creation, production, or delivery of the script be liable for

any damages whatsoever (including, without limitation, damages for loss of business profits,

business interruption, loss of business information, or other pecuniary loss) arising out of

the use of or inability to use the sample or documentation, even if Microsoft has been advised

of the possibility of such damages, rising out of the use of or inability to use the sample script,

even if Microsoft has been advised of the possibility of such damages.

Description:
Script Description: Calendar Event Creation via Microsoft Graph with Azure Key Vault Integration
This PowerShell script automates the creation of a Microsoft Teams calendar event using Microsoft Graph API and securely retrieves credentials from Azure Key Vault. It is designed for use within the context of the wolffofficesub Azure subscription.

Setup and Authentication
Modules & Context: Imports the Microsoft Graph Calendar module and sets the Azure subscription context.
Graph Connection: Connects to Microsoft Graph with delegated permissions for managing applications and role assignments.
Key Vault Access
Vault Configuration: Updates network rules and public access settings for the Key Vault named wolffofficekvkv2 in resource group Adminrg.
Secret Retrieval: Attempts to retrieve the client secret for the app registration WOLFFCalendarManagerAppspn.
If blocked by firewall rules, it dynamically adds the client IP to the Key Vault's access list and retries.
Token Acquisition
Uses the retrieved client secret to request an access token from Microsoft's OAuth2 endpoint for Graph API access.
Calendar Event Definition
Defines a Teams meeting titled "POC Strategy Sync" for user jerrywolff@wpi-corp.com, scheduled on July 31, 2025, from 10:00 AM to 11:00 AM PST.
Includes HTML-formatted body content and specifies the meeting as online via Teams.
Event Creation
Retrieves the target user's email via Graph.
Sends a POST request to Microsoft Graph to create the calendar event.
Logs the response to a local file for auditing.

#>

```
Import-Module Microsoft.Graph.Calendar

# Set subscription context
Connect-AzAccount #-Identity
$context = Set-AzContext -Subscription "wolffofficesub"

# Variables
$vaultname = 'wolffofficekvkv2'
$resourceGroup = 'Adminrg'

$displayName = "WOLFFCalendarManagerAppspn"
$logFile = "$env:USERPROFILE\kv_access_log.txt"
$removeIpAfter = $true

# Connect to Microsoft Graph with delegated permissions
Connect-MgGraph -tenant $($subscription.TenantId) -Scopes "Application.ReadWrite.All", "AppRoleAssignment.ReadWrite.All"

# Get app registration info
$app = Get-AzADApplication -DisplayName $displayName
$clientId = $app.AppId
$tenantId = (Get-AzContext).Tenant.Id
$scopes = "https://graph.microsoft.com/.default"
$secretName = $($app.displayname)

        Update-AzKeyVaultNetworkRuleSet -DefaultAction Allow -VaultName $vaultname

        Update-AzKeyVault -ResourceGroupName $resourceGroup `
                -VaultName $vaultname `
                -PublicNetworkAccess Enabled
```

```
# Try to retrieve the client secret from Key Vault
try {
    $clientSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name $secretName -AsPlainText
    if ([string]::IsNullOrWhiteSpace($clientSecret)) {
        throw "Client secret is empty. Check the Key Vault secret value."
    }
}
catch {
    $rawError = $_.Exception.Message
    if ($rawError -match "Client address: (\d{1,3}(?:\.\d{1,3}){3})") {
        $clientIp = $matches[1]
        Write-Host "Detected client IP: $clientIp"

        # Add IP to firewall
        Update-AzKeyVaultNetworkRuleSet -VaultName $vaultName -IpAddressRange $clientIp
        Add-Content -Path $logFile -Value "$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss') - Added IP $clientIp to $vaultName"
        Start-Sleep -Seconds 15

        # Retry secret retrieval
        $clientSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name $secretName -AsPlainText
        if ([string]::IsNullOrWhiteSpace($clientSecret)) {
            # Fallback: open public access
            Update-AzKeyVaultNetworkRuleSet -DefaultAction Allow -VaultName $vaultName
            Update-AzKeyVault -ResourceGroupName $resourceGroup -VaultName $vaultName -PublicNetworkAccess Enabled
            $clientSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name $secretName -AsPlainText
        }
    } else {
        Write-Error "Could not extract IP address. Raw error: $rawError"
        return
    }
}

# Get access token
$tokenBody = @{
    grant_type    = "client_credentials"
    client_id     = $clientId
    client_secret = $clientSecret
    scope         = $scopes
}

try {
    $tokenResponse = Invoke-RestMethod -Uri "https://login.microsoftonline.com/$tenantId/oauth2/v2.0/token" `
        -Method POST -Body $tokenBody
    $accessToken = $tokenResponse.access_token
}
catch {
    Write-Error "Failed to acquire token: $($_.Exception.Message)"
    return
}

# Define calendar event
$UPN = "jerrywolff@wpi-corp.com"
$event = @{
    subject = "Protyp Sync"
    body = @{
        contentType = "HTML"
        content = "This meeting was scheduled using application permissions."
    }
    start = @{
        dateTime = "2025-08-31T10:00:00"
        timeZone = "Pacific Standard Time"
    }
    end = @{
        dateTime = "2025-08-31T11:00:00"
        timeZone = "Pacific Standard Time"
    }
    location = @{
        displayName = "Microsoft Teams Meeting"
    }
    attendees = @(
        @{
            emailAddress = @{
                address = $userEmail
                name = "Target User"
            }
            type = "required"
        }
    )
    isOnlineMeeting = $true
    onlineMeetingProvider = "teamsForBusiness"
} | ConvertTo-Json -Depth 10
```

```
# Create the event
try {
$useremailinfo = Get-MgUser | where Userprincipalname -eq  "$upn"
$useremail = $($useremailinfo.mail)

   $response = Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/users/$userEmail/events" `
      -Headers @{ Authorization = "Bearer $accessToken" } `
      -Method POST `
      -Body $event `
      -ContentType "application/json"
   $response
   $response | Out-File c:\temp\calendar_create_response.txt -Encoding unicode



}
catch {
   Write-Error "Failed to create calendar event: $($_.Exception.Message)"
}
```

---

v

create_calendar_ser...

```
<#
.NOTES

   THIS CODE-SAMPLE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED

   OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR

   FITNESS FOR A PARTICULAR PURPOSE.

   This sample is not supported under any Microsoft standard support program or service.

   The script is provided AS IS without warranty of any kind. Microsoft further disclaims all

   implied warranties including, without limitation, any implied warranties of merchantability

   or of fitness for a particular purpose. The entire risk arising out of the use or performance

   of the sample and documentation remains with you. In no event shall Microsoft, its authors,

   or anyone else involved in the creation, production, or delivery of the script be liable for

   any damages whatsoever (including, without limitation, damages for loss of business profits,

   business interruption, loss of business information, or other pecuniary loss) arising out of

   the use of or inability to use the sample or documentation, even if Microsoft has been advised

   of the possibility of such damages, rising out of the use of or inability to use the sample script,

   even if Microsoft has been advised of the possibility of such damages.

Description: Integration for Calendar Management
This script automates the creation and configuration of an Azure AD application and service principal
 for managing calendar events via Microsoft Graph. It also securely stores the app credentials in Azure Key Vault and configures network access policies.

☑ Actions Performed
Module Import & Authentication

Imports Microsoft.Graph.Applications module with verbose output.
Authenticates to Azure using managed identity and sets the subscription context to "wolffofficesub".
App Registration

Creates a new Azure AD application named WOLFFCalendarManagerAppspn with specified homepage and identifier URI.
Retrieves the app owner (admin@wpi-corp.com) and assigns ownership (commented out in current script).
Service Principal Creation

Generates a service principal for the registered app.
Retrieves the Microsoft Graph service principal and assigns required Graph API permissions:
Calendars.Read, Calendars.ReadWrite, MailboxSettings.Read, User.Read.All.
```

Metadata Recording

Captures and stores metadata about the service principal and app registration (e.g., tenant ID, client ID, secret, key ID, validity dates).
Key Vault Configuration

Retrieves current IP rules and adds the user's public IP and automation IP (20.236.10.163) if not already present.
Updates Key Vault network rules and enables public network access.
Sets access policies for the service principal and current user to manage secrets.
Secret Management

Generates a new client secret for the app and stores it in Azure Key Vault.
Tags the secret with metadata for automation tracking.
Implements error handling to dynamically add client IP to firewall rules if access is denied, then retries secret creation and storage.
Admin Consent URL Output

Displays the admin consent URL for granting permissions to the app.
📌 **Requirements**
Azure Modules: Microsoft.Graph.Applications, Az.Accounts, Az.Resources, Az.KeyVault, Az.AD
Permissions:
Admin privileges to register applications and assign Graph roles.
Access to modify Key Vault network rules and set access policies.
Key Vault: Must exist with name wolffofficekvkv2 in resource group Adminrg.
Subscription Selection: Interactive selection via Out-GridView.
Firewall Access: Script must be run from an IP address that can be added to Key Vault rules if needed.

```
#>
import-module -name az  -Force
import-module  Microsoft.Graph.Applications -verbose

 # Connect using managed identity
$context = Connect-AzAccount # -tenant wolffofficetenant -Subscription wolffofficesub # -Identity

# Set subscription context
Set-AzContext -Subscription "wolffofficesub"

# Define variables
$vaultname = 'wolffofficekvkv2'
$resourceGroup = 'Adminrg'
$spnname = 'wolffcalendarspn'
$displayName = "WOLFFCalendarManagerAppspn"
$appowner = get-azaduser | where mail -eq 'admin@wpi-corp.com'
$spnrec = @()

# Select subscription
$subscription = Get-AzSubscription | Out-GridView -Title "Select a subscription:" -PassThru | Select-Object Name, Id, TenantId -First 1


Write-Host "Tenant/sub : $($subscription.Name) - $($subscription.TenantId)" -ForegroundColor Green

# Connect to Microsoft Graph with admin privileges
Connect-MgGraph -tenant $($subscription.TenantId) -Scopes "Application.ReadWrite.All", "AppRoleAssignment.ReadWrite.All"


Set-AzContext -Subscription $($subscription.Name) -Tenant $($subscription.TenantId)


$app = New-AzADApplication -DisplayName $displayName `
   -IdentifierUris "https://wpi-corp.com/CalendarManagerApp" `
   -HomePage "https://wpi-corp.com/CalendarManagerApp"


 $params = @{
  "@odata.id" = "https://graph.microsoft.com/v1.0/directoryObjects/$($appowner.id)"
}

$appinfo = get-azadapplication | where displayname -eq $displayname

start-sleep -seconds 60
#$newowner = New-MgApplicationOwnerByRef -ApplicationId $($appinfo.AppId) -BodyParameter $params




# Create the service principal
$newsp = New-AzADServicePrincipal -ApplicationId $($appinfo.appid)
$sp = get-AzADServicePrincipal -ApplicationId $($appinfo.appid)
```

```
start-sleep -seconds 60



# Get Microsoft Graph SPN
$graphSp = Get-MgServicePrincipal -Filter "displayName eq 'Microsoft Graph'"



# Get your app's SPN
#$mySp = Get-MgServicePrincipal -ServicePrincipalId

# Assign Graph application permissions
$permissions = @("Calendars.Read", "Calendars.ReadWrite", "MailboxSettings.Read", "User.Read.All")
foreach ($perm in $permissions) {
  $appRole = $($graphSp.AppRoles) | Where-Object { $_.Value -eq $perm -and $_.AllowedMemberTypes -contains "Application" }
  if ($appRole) {
    New-MgServicePrincipalAppRoleAssignment -ServicePrincipalId $($sp.Id) `
      -PrincipalId $($Sp.Id) `
      -ResourceId $($graphSp.Id) `
      -AppRoleId $($appRole.Id)

    Write-Output "☑ Assigned $perm"
  } else {
    Write-Warning "⚠ Permission $perm not found in Microsoft Graph"
  }
}

# Record SPN metadata
$spnobj = New-Object PSObject
$spnobj | Add-Member -MemberType NoteProperty -Name Tenantid -Value $($subscription.TenantId)
$spnobj | Add-Member -MemberType NoteProperty -Name Applicationid -Value $($app.AppId)
$spnobj | Add-Member -MemberType NoteProperty -Name Displayname -Value $($app.DisplayName)
$spnobj | Add-Member -MemberType NoteProperty -Name ClientID -Value $($app.AppId)
$spnobj | Add-Member -MemberType NoteProperty -Name Appid -Value $($appinfo.AppId)
$spnobj | Add-Member -MemberType NoteProperty -Name Clientsecret -Value $appSecret
$spnobj | Add-Member -MemberType NoteProperty -Name Keyid -Value ([guid]::NewGuid())
$spnobj | Add-Member -MemberType NoteProperty -Name Enddate -Value (Get-Date).AddYears(1)
$spnobj | Add-Member -MemberType NoteProperty -Name Startdate -Value (Get-Date)
$spnobj | Add-Member -MemberType NoteProperty -Name Objectid -Value $($sp.Id)
$spnrec = $spnobj



############


 <#######  Add my current ip address to the network firewallrule if not there

# Get your current public IP address
$myIp = (Invoke-RestMethod -Uri "https://api.ipify.org?format=json").ip

# Get current IP rules
$kv = Get-AzKeyVault -VaultName $vaultName -ResourceGroupName $resourceGroup
$currentIps = $kv.NetworkAcls.IpRules.IpAddressOrRange

# Add your IP if it's not already in the list
if ($currentIps -notcontains $myIp) {
  $updatedIps = $currentIps + $myIp

  # Update the Key Vault network rules
  Update-AzKeyVaultNetworkRuleSet -VaultName $vaultName `
    -ResourceGroupName $resourceGroup `
    -IpAddressRange $updatedIps `
    -DefaultAction Deny

  Write-Host "☑ Added IP $myIp to Key Vault firewall rules."
} else {
  Write-Host "🛈 IP $myIp is already allowed."
}


#$automationIp = "20.236.10.163"  # for automation MI internal Microsoft IP only not public

# Get current IP rules
$kv = Get-AzKeyVault -VaultName $vaultName -ResourceGroupName $resourceGroup
$currentIps = $kv.NetworkAcls.IpRules.IpAddressOrRange

# Add the automation IP if not already present
if ($currentIps -notcontains $automationIp) {
  $updatedIps = $currentIps + $automationIp
```

```
    Update-AzKeyVaultNetworkRuleSet -VaultName $vaultName `
      -ResourceGroupName $resourceGroup `
      -IpAddressRange $updatedIps `
      -DefaultAction Deny

    Write-Host "☑ Added automation IP $automationIp to Key Vault firewall rules."
} else {
    Write-Host "🛈 IP $automationIp is already allowed."
}
#>

#$vault =   Get-AzKeyVault -VaultName "$vaultname" -ResourceGroupName "$resourceGroup"  -SubscriptionId $($subscription.Id)

Update-AzKeyVaultNetworkRuleSet -VaultName "$vaultname" -Bypass AzureServices


Set-AzKeyVaultAccessPolicy -VaultName "$vaultname" `
    -ObjectId "$($sp.id)" `
    -PermissionsToSecrets set,get,list


try {

        $clientSecret = New-AzADAppCredential -ApplicationId $($spnrec.ClientID)
        $secureSecretValue = ConvertTo-SecureString -String $clientSecret.SecretText -AsPlainText -Force

     #  Update-AzKeyVaultNetworkRuleSet -DefaultAction Allow -VaultName $vaultname

           Update-AzKeyVault -ResourceGroupName $resourceGroup `
                      -VaultName $vaultname `
                     -PublicNetworkAccess Enabled

        (Get-AzContext).Account.Id


        $userobjectid = (Get-AzADUser -UserPrincipalName (Get-AzContext).Account.Id).Id


        Set-AzKeyVaultAccessPolicy `
  -VaultName "wolffofficekvkv2" `
  -ObjectId "$userobjectid" `
  -PermissionsToSecrets get, list, delete, purge, set




    Set-AzKeyVaultSecret -VaultName $vaultname -Name $spnrec.Displayname `
    -SecretValue $secureSecretValue `
    -Tag @{Purpose = "Spnautomation"; Clientid = "$($spnrec.ClientID)"; Enddatetime = "$($spnrec.Enddate)"; keyid = "$($spnrec.keyid)"} `
    -ContentType "$($spnrec.Appid)"

}
catch {
    $rawError = $Error[0].ToString()

    if ($rawError -match "Client address: (\d{1,3}(?:\.\d{1,3}){3})") {
        $clientIp = $matches[1]
        Write-Host "Detected client IP: $clientIp"

        # Add IP to firewall
        Write-Host "Adding $clientIp to Key Vault firewall..."
        Update-AzKeyVaultNetworkRuleSet -VaultName $vaultName -IpAddressRange $clientIp
        Write-Host "IP address added. Retrying secret retrieval..."

        # Log the IP and timestamp
        $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        Add-Content -Path $logFile -Value "$timestamp - Added IP $clientIp to $vaultName"

        Start-Sleep -Seconds 15

        try {

             $clientSecret = New-AzADAppCredential -ApplicationId $($spnrec.ClientID)
        $secureSecretValue = ConvertTo-SecureString -String $clientSecret.SecretText -AsPlainText -Force

           Set-AzKeyVaultSecret -VaultName $vaultname -Name $($spnrec.Displayname)`
    -SecretValue $secureSecretValue `
    -Tag @{Purpose = "Spnautomation"; Clientid = "$($spnrec.ClientID)"; Enddatetime = "$($spnrec.Enddate)"; keyid = "$($spnrec.keyid)"} `
    -ContentType "$($spnrec.Appid)"

        }
        catch {
```

```
    Set-AzKeyVaultAccessPolicy `
 -VaultName "wolffofficekvkv2" `
 -ObjectId "$userobjectid" `
 -PermissionsToSecrets get, list, delete, purge, set

 ######### To cleanup old secrets if this is re-run with the same app name


        Remove-AzKeyVaultSecret -VaultName $vaultname -Name "$($spnrec.Displayname)" -Force -InRemovedState

############################

          Update-AzKeyVaultNetworkRuleSet -DefaultAction Allow -VaultName $vaultname

          Update-AzKeyVault -ResourceGroupName $resourceGroup `
                -VaultName $vaultname `
                -PublicNetworkAccess Enabled


      $clientSecret = New-AzADAppCredential -ApplicationId $($spnrec.ClientID)
    $secureSecretValue = ConvertTo-SecureString -String $clientSecret.SecretText -AsPlainText -Force

       Set-AzKeyVaultSecret -VaultName $vaultname -Name $($spnrec.Displayname) `
  -SecretValue $secureSecretValue `
  -Tag @{Purpose = "Spnautomation"; Clientid = "$($spnrec.ClientID)"; Enddatetime = "$($spnrec.Enddate)"; keyid = "$($spnrec.keyid)"} `
  -ContentType "$($spnrec.Appid)"

       Write-Error "Retry failed. fixing network access."
       return
    }
  }
  else {
    Write-Host "Could not extract IP address from error message."
    Write-Host "Raw error: $rawError"
    return
  }
}



# Output admin consent URL
Write-Host "`n Admin consent URL:"
Write-Host "https://login.microsoftonline.com/$($subscription.TenantId)/adminconsent?client_id=$($app.ApplicationId)"
```

```
 wolffofficesub Azure subscription.

  Setup and Authentication
Modules & Context: Imports the Microsoft Graph Calendar module and sets the Azure
subscription context.
Graph Connection: Connects to Microsoft Graph with delegated permissions for
managing applications and role
assignments.
  Key Vault Access
Vault Configuration: Updates network rules and public access settings for the Key
Vault named wolffofficekvkv2
in resource group Adminrg.
Secret Retrieval: Attempts to retrieve the client secret for the app registration
WOLFFCalendarManagerAppspn.
If blocked by firewall rules, it dynamically adds the client IP to the Key Vault's
access list and retries.
  Token Acquisition
Uses the retrieved client secret to request an access token from Microsoft's OAuth2
endpoint for Graph API access.
  Calendar Event Definition
Defines a Teams meeting titled "POC Strategy Sync" for user jerrywolff@wpi-corp.com,
scheduled on July 31, 2025,
from 10:00 AM to 11:00 AM PST.
Includes HTML-formatted body content and specifies the meeting as online via Teams.
  Event Creation
Retrieves the target user's email via Graph.
Sends a POST request to Microsoft Graph to create the calendar event.
Logs the response to a local file for auditing.


 #>



    Import-Module Microsoft.Graph.Calendar

# Set subscription context
Connect-AzAccount #-Identity
$context = Set-AzContext -Subscription "wolffofficesub"

# Variables
$vaultname = 'wolffofficekvkv2'
$resourceGroup = 'Adminrg'

$displayName = "WOLFFCalendarManagerAppspn"
$logFile = "$env:USERPROFILE\kv_access_log.txt"
$removeIpAfter = $true

# Connect to Microsoft Graph with delegated permissions
```

```
Connect-MgGraph -tenant $($subscription.TenantId) -Scopes
"Application.ReadWrite.All", "AppRoleAssignment.ReadWrite.All"

# Get app registration info
$app = Get-AzADApplication -DisplayName $displayName
$clientId = $app.AppId
$tenantId = (Get-AzContext).Tenant.Id
$scopes = "https://graph.microsoft.com/.default"
$secretName = $($app.displayname)

                  Update-AzKeyVaultNetworkRuleSet -DefaultAction Allow -VaultName
$vaultname

                 Update-AzKeyVault -ResourceGroupName $resourceGroup `
                             -VaultName $vaultname `
                             -PublicNetworkAccess Enabled


# Try to retrieve the client secret from Key Vault
try {
    $clientSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name $secretName
-AsPlainText
    if ([string]::IsNullOrWhiteSpace($clientSecret)) {
        throw "Client secret is empty. Check the Key Vault secret value."
    }
}
catch {
    $rawError = $_.Exception.Message
    if ($rawError -match "Client address: (\d{1,3}(?:\.\d{1,3}){3})") {
        $clientIp = $matches[1]
        Write-Host "Detected client IP: $clientIp"

        # Add IP to firewall
        Update-AzKeyVaultNetworkRuleSet -VaultName $vaultName -IpAddressRange
$clientIp
        Add-Content -Path $logFile -Value "$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')
- Added IP $clientIp to $vaultName"
        Start-Sleep -Seconds 15

        # Retry secret retrieval
        $clientSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name $secretName
-AsPlainText
        if ([string]::IsNullOrWhiteSpace($clientSecret)) {
            # Fallback: open public access
            Update-AzKeyVaultNetworkRuleSet -DefaultAction Allow -VaultName
$vaultName
            Update-AzKeyVault -ResourceGroupName $resourceGroup -VaultName
$vaultName -PublicNetworkAccess Enabled
            $clientSecret = Get-AzKeyVaultSecret -VaultName $vaultName -Name
$secretName -AsPlainText
```

```
        }
    } else {
        Write-Error "Could not extract IP address. Raw error: $rawError"
        return
    }
}

# Get access token
$tokenBody = @{
    grant_type    = "client_credentials"
    client_id     = $clientId
    client_secret = $clientSecret
    scope         = $scopes
}

try {
    $tokenResponse = Invoke-RestMethod -Uri
"https://login.microsoftonline.com/$tenantId/oauth2/v2.0/token" `
        -Method POST -Body $tokenBody
    $accessToken = $tokenResponse.access_token
}
catch {
    Write-Error "Failed to acquire token: $($_.Exception.Message)"
    return
}

# Define calendar event
$UPN = "jerrywolff@wpi-corp.com"
$event = @{
    subject = "Protyp Sync"
    body = @{
        contentType = "HTML"
        content = "This meeting was scheduled using application permissions."
    }
    start = @{
        dateTime = "2025-08-31T10:00:00"
        timeZone = "Pacific Standard Time"
    }
    end = @{
        dateTime = "2025-08-31T11:00:00"
        timeZone = "Pacific Standard Time"
    }
    location = @{
        displayName = "Microsoft Teams Meeting"
    }
    attendees = @(
        @{
            emailAddress = @{
                address = $userEmail
                name = "Target User"
```

```
            }
            type = "required"
        }
    )
    isOnlineMeeting = $true
    onlineMeetingProvider = "teamsForBusiness"
} | ConvertTo-Json -Depth 10

# Create the event
try {
$useremailinfo = Get-MgUser | where Userprincipalname -eq  "$upn"
$useremail = $($useremailinfo.mail)

    $response = Invoke-RestMethod -Uri
"https://graph.microsoft.com/v1.0/users/$userEmail/events" `
        -Headers @{ Authorization = "Bearer $accessToken" } `
        -Method POST `
        -Body $event `
        -ContentType "application/json"
    $response
    $response | Out-File c:\temp\calendar_create_response.txt -Encoding unicode



}
catch {
    Write-Error "Failed to create calendar event: $($_.Exception.Message)"
}
```