Microsoft Teams

This book section is for MS Team documentations

- MS Teams Online Meetings Integration with 3rd-Party Apps (MS Graph API)
- Microsoft Teams Application Access Policy – User Assignment Automation
- Microsoft Teams Retention Policy Configuration in Microsoft Purview

MS Teams Online Meetings Integration with 3rd-Party Apps (MS Graph API)

Overview

Overview

This configuration enables a third-party application to create and manage Microsoft Teams online meetings using Microsoft Graph API integration.

The integration is typically used by:

- Student engagement platforms
- Recruitment systems
- CRM platforms
- Booking systems
- Service management applications
- Virtual appointment platforms

The implementation uses:

- Microsoft Entra ID App Registration
- Microsoft Graph API
- Teams OnlineMeetings API
- Teams Application Access Policies

This approach allows controlled programmatic creation of Teams meeting links while maintaining governance and restricting which user accounts the application may act on.

This configuration is important because it:

- Enables secure automation of Teams meeting creation
- Prevents unrestricted tenant-wide meeting creation by third-party apps
- Provides governance and operational control
- Reduces the blast radius in the event of application compromise

> Prerequisites

Licensing

Ensure the following licenses/services are available:

- Microsoft 365 tenant
- Microsoft Teams enabled
- Microsoft Entra ID
- Microsoft Graph API access

Required Roles

The implementing administrator should have:

- Global Administrator OR
- Application Administrator
- Teams Administrator

Required PowerShell Modules

Install the following PowerShell modules:

```
Install-Module Microsoft.Graph -Scope CurrentUser
Install-Module MicrosoftTeams -Scope CurrentUser
```

Required Permissions

The Entra ID application registration will require:

Microsoft Graph Application Permissions

- `OnlineMeetings.ReadWrite.All`
- `User.Read.All` OR `User.ReadBasic.All`

Admin consent must be granted.

Certificate Authentication (Recommended)

Certificate-based authentication is strongly recommended over client secrets for:

- Improved security

- Reduced credential exposure
- Better long-term automation support
- Reduced secret rotation overhead

Step 1: Create the Entra ID Application Registration

Navigate to:

Entra Admin Center > Applications > App registrations

Create the Application

Select:

- New registration

Configure:

- Name: `Teams Meetings Integration`
- Supported account type: Single tenant (recommended)

Select:

- Register

Record the following values

Save:

- Application (client) ID
- Directory (tenant) ID

These values will be required for:

- Graph authentication
- PowerShell automation
- Third-party application configuration

Step 2: Configure API Permissions

Navigate to:

App Registration > API permissions

Add Microsoft Graph Application Permissions

Add:

- `OnlineMeetings.ReadWrite.All`

- `User.Read.All`

OR for reduced exposure:

- `User.ReadBasic.All`

Grant Admin Consent

Select:

- Grant admin consent for tenant

Validate Permission Status

Ensure all permissions display:

- `Granted for <TenantName>`

Step 3: Configure Certificate Authentication

Generate Certificate

Run PowerShell:

```
$cert = New-SelfSignedCertificate `
    -Subject "CN=TeamsMeetingsIntegration" `
    -CertStoreLocation "Cert:\CurrentUser\My" `
    -KeySpec Signature `
    -KeyLength 2048 `
    -KeyExportPolicy Exportable `
    -HashAlgorithm SHA256 `
    -NotAfter (Get-Date).AddYears(2)
```

Export Public Certificate

```
Export-Certificate `
    -Cert $cert `
    -FilePath "C:\Temp\TeamsMeetingsIntegration.cer"
```

Upload Certificate to App Registration

Navigate to:
App Registration > Certificates & secrets > Certificates

Upload:

- `.cer` file

Record Certificate Thumbprint

Run:

```
$cert.Thumbprint
```

Save the thumbprint securely.

Step 4: Create the Teams Application Access Policy

Connect to Microsoft Teams PowerShell

```
Connect-MicrosoftTeams
```

Create the Policy

```
New-CsApplicationAccessPolicy `
    -Identity "Tag:TeamsMeetingsIntegration" `
    -AppIds "<ApplicationClientID>" `
    -Description "Restricts Teams meeting creation to approved operator accounts"
```

Validate Policy Creation

```
Get-CsApplicationAccessPolicy
```

Record the exact policy identity name.

Step 5: Assign the Application Access Policy

Purpose

The Application Access Policy controls which user accounts the application may act on when creating Teams meetings using application permissions.

Without this policy:

- The application may attempt broader access
- Governance controls are weakened

Assign Policy to Approved Users

Example:

```
Grant-CsApplicationAccessPolicy `
    -Identity user@domain.com `
    -PolicyName "Tag:TeamsMeetingsIntegration"
```

Validate Assignment

```
Get-CsOnlineUser -Identity user@domain.com |
Select UserPrincipalName, ApplicationAccessPolicy
```

> **Important Notes**

- Policies are assigned per-user
- Native group assignment is not supported
- Automation is recommended for larger user populations

Step 6: Testing / Validation

Recommended Safe Rollout Approach

Start with:

- Test tenant
- Small pilot group
- Non-production accounts

Validate Graph Authentication

Example:

```
Connect-MgGraph `
    -TenantId "<TenantID>" `
    -ClientId "<ClientID>" `
    -CertificateThumbprint "<Thumbprint>"
```

Validate Teams PowerShell Authentication

```
Connect-MicrosoftTeams `
    -TenantId "<TenantID>" `
    -ApplicationId "<ClientID>" `
    -CertificateThumbprint "<Thumbprint>"
```

Test Meeting Creation Using Postman

Token Endpoint

Use:

- OAuth2 Client Credentials flow

Grant type:

```
client_credentials
```

Important Distinction

Ensure testing uses:

- Application permissions

NOT:

- Delegated user authentication

This is critical because:

- Application Access Policies only apply to app-only authentication flows

Test Online Meeting Creation

POST request:

```
POST https://graph.microsoft.com/v1.0/users/{user-id}/onlineMeetings
```

Example payload:

```
{
  "startDateTime": "2026-06-11T10:00:00Z",
  "endDateTime": "2026-06-11T10:30:00Z",
  "subject": "Teams Integration Test",
  "participants": {},
  "lobbyBypassSettings": {
      "scope": "everyone",
      "isDialInBypassEnabled": true
  },
  "allowedPresenters": "everyone"
}
```

Expected Behaviour

Users WITH policy assignment

- Meeting creation succeeds

Users WITHOUT policy assignment

- Request should fail with authorization-related error

Step 7: Monitoring & Validation

Entra Sign-In Logs

Navigate to:
Entra Admin Center > Monitoring > Sign-in logs

Review:

- Service principal sign-ins
- Authentication failures
- Unusual locations
- Unusual application activity

Audit Logs

Review:

- App permission changes
- Consent grants
- Policy modifications

Teams PowerShell Validation

Validate assigned users:

```
Get-CsOnlineUser |
Where-Object {
    $_.ApplicationAccessPolicy -eq "Tag:TeamsMeetingsIntegration"
}
```

Microsoft Graph Monitoring

Monitor:

- Failed API requests
- Excessive meeting creation
- Abnormal usage patterns

Step 8: Enforcement / Go-Live

Before Production Rollout

Validate:

- Policy assignment scope
- Authentication method
- Third-party application configuration
- Logging and monitoring
- Pilot user testing

Go-Live Activities

- Enable production application configuration
- Assign approved operator accounts
- Confirm successful meeting creation
- Monitor sign-in activity closely for first 7 days

Post-Go-Live Monitoring

Pay close attention to:

- Unexpected meeting creation volume
- Authentication anomalies
- Unauthorized access attempts
- Service principal activity

> ## Important Considerations

Delegated vs Application Authentication

This is one of the most important concepts in this integration.

Delegated Authentication

- User signs in interactively
- User acts as themselves
- Application Access Policy does NOT apply

Application Authentication

- App acts independently
- No user interaction required
- Application Access Policy IS enforced

Improper testing using delegated authentication can lead to incorrect assumptions about policy enforcement.

Cross-Tenant Meeting Access

External tenant users may:

- Require lobby admission
- Experience authentication prompts depending on federation configuration
- Behave differently depending on how the meeting was created

Summary

This implementation enables secure integration between Microsoft Teams and third-party applications using Microsoft Graph OnlineMeetings APIs.

The solution uses:

- Entra ID App Registrations
- Microsoft Graph Application Permissions
- Teams Application Access Policies
- Certificate-based authentication

The configuration provides:

- Controlled Teams meeting automation
- Restriction of application scope to approved users
- Improved governance and operational security
- Reduced tenant-wide exposure in the event of compromise

Proper implementation and testing of Application Access Policies is critical to ensuring the integration operates securely and as intended.

Microsoft Teams Application Access Policy – User Assignment Automation

Field	Details
Document Type	Microsoft Teams Application Access Policy – User Assignment Automation
Applies To	Microsoft Entra ID, Microsoft Teams & MS Graph
Audience	MS Teams Administrator / IT Engineer
Author	AK. Udofeh
Last Updated	June 2026

Overview

This document provides a high-level overview of the automated user assignment process used for Microsoft Teams Application Access Policies.

The automation is designed to simplify operational management of Teams Application Access Policies by using Entra ID group membership as the source of truth for policy assignment.

This approach is commonly used when integrating:

- Third-party applications
- Microsoft Graph OnlineMeetings APIs
- Teams meeting automation platforms
- Virtual appointment systems

The automation helps ensure that only approved users are assigned to the relevant Teams Application Access Policy.

Business Justification

Microsoft Teams Application Access Policies currently do not support:

- Native Active Directory group assignment
- Entra ID group targeting

Policies must instead be assigned individually per-user using PowerShell.

Without automation:

- User onboarding becomes manual
- User removals may be missed
- Administrative overhead increases
- Access governance becomes difficult to maintain
- Policy drift risks increase over time

The automation solves this limitation by:

- Using Entra ID groups as the source of truth
- Automatically assigning approved users
- Automatically removing users no longer approved
- Maintaining operational consistency

High-Level Process Flow

Source of Truth

Approved users are managed through designated:

- Entra ID security groups
  OR
- synced Active Directory groups

Automation Workflow

The automation process performs the following actions:

1. Connects to Microsoft Graph
2. Enumerates approved group members
3. Connects to Microsoft Teams PowerShell
4. Retrieves existing policy assignments
5. Performs delta comparison
6. Assigns missing users to the policy
7. Removes users no longer in scope
8. Logs all changes and failures

Operational Behaviour

User Additions

When a user is added to the approved group:

- The automation assigns the Teams Application Access Policy automatically during the next scheduled run

User Removals

When a user is removed from the approved group:

- The automation removes the Teams Application Access Policy assignment automatically during the next scheduled run

Delta-Based Processing

The automation uses delta comparison logic to:

- Avoid duplicate assignments
- Minimise unnecessary PowerShell operations
- Improve execution efficiency

Authentication Model

The automation uses:

- Microsoft Graph PowerShell
- Microsoft Teams PowerShell
- Entra ID App Registration
- Certificate-based authentication

Certificate authentication is recommended because it:

- Reduces credential exposure
- Supports unattended execution
- Improves operational security
- Avoids client secret lifecycle management

Validation & Testing

Validation should include:

- Successful group enumeration
- Successful Teams PowerShell connectivity
- Successful policy assignment
- Successful policy removal
- Delta comparison validation
- Logging verification

Example validation checks:

```
Get-CsApplicationAccessPolicy
```

```
Get-CsOnlineUser -Identity user@domain.com
```

Validation should confirm:

- Approved users receive the policy
- Removed users lose the policy
- Existing assignments are not duplicated

Monitoring & Logging

The automation should log:

- Added users
- Removed users
- Assignment failures
- Execution timestamps
- Authentication failures

The following should be monitored periodically:

- Script execution success
- Teams PowerShell connectivity
- Certificate expiry dates
- Group membership accuracy
- Failed assignment attempts

Important Considerations

No Native Group Support

This automation exists specifically because:

- Teams Application Access Policies currently do not support direct group assignment

Policy Scope

Policies should remain scoped only to:

- Approved operator accounts
- Approved service users

Global assignment is not recommended.

Teams RBAC Permissions

The Entra ID application used by the automation requires:

- Microsoft Graph permissions
- Teams administrative RBAC permissions

Certificate Lifecycle Management

Certificate expiry dates should be monitored carefully to avoid automation failure.

Automation Script

```
# Get list of MS Teams App Access policy
# Get-CsApplicationAccessPolicy

# ============================================================
# Teams Application Access Policy Automation Script
# App-only authentication | Delta-based | Lightweight Version
# ============================================================

# ---------------------------
# CONFIGURATION
# ---------------------------

$TenantId = "{your tenant id here}"   # UniEssex Entra ID tenant ID (GUID)
$AppId = "{your app id here}"   # Entra ID app registration application (client) ID (GUID)
$CertThumbprint = "{your cert thumbprint here}"   # Thumbprint of the certificate used for
authentication (no spaces)

# Teams Application Access Policy Name

$TeamsPolicyName = "{your Teams App Policy name here}"

# Entra ID Groups containing approved operators

$GroupIds = @("{your Entra ID Group name here}")

$LogFile = "{your log file location here}"

# ---------------------------
# LOGGING
# ---------------------------

$logDir = Split-Path $LogFile

if (!(Test-Path $logDir)) {
New-Item -ItemType Directory -Path $logDir | Out-Null
```

```
}

function Write-Log {
    param (
        [string]$Action,
        [string]$UPN
    )
    $timestamp = "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] -"
    if($Action.StartsWith("ERROR_")) {
        $timestamp = "*** $timestamp"
    }
    Add-Content -Path $LogFile -Value "$timestamp $Action,$UPN,$TeamsPolicyName"
    Write-Host "$timestamp $Action,$UPN,$TeamsPolicyName"
}
# ----------------------------
# CONNECT TO MICROSOFT GRAPH
# ----------------------------

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Connecting to MS Graph..."

Connect-MgGraph `
    -TenantId $TenantId `
    -ClientId $AppId `
    -CertificateThumbprint $CertThumbprint `
    -NoWelcome

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Querying Group Members..."

$GroupUsers = foreach($GroupId in $GroupIds) {

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Querying Group $GroupId Members..."

(Get-MgGroupMemberAsUser `
    -All `
    -GroupId $GroupId `
    -Property UserPrincipalName
).UserPrincipalName.ToLower()

}
```

```
# Deduplicate users

$GroupUsers = $GroupUsers | Sort-Object -Unique

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - $($GroupUsers.Length) Group Users to
evaluate..."

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Disconnecting from MS Graph... "

Disconnect-MgGraph

# ----------------------------
# LIGHTWEIGHT SAFETY CHECK
# ----------------------------

if ($GroupUsers.Count -lt 1) {
    Write-Host "*** [$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - ERROR No group users
returned"
    Write-Log "ABORT" "No group users returned"
exit 1
}

# ----------------------------
# CONNECT TO MICROSOFT TEAMS
# ----------------------------

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Connecting to Microsoft Teams..."

Connect-MicrosoftTeams `
    -TenantId $TenantId `
    -ApplicationId $AppId `
    -CertificateThumbprint $CertThumbprint

Get-CsOnlineUser |
ForEach-Object {

    Write-Host "User: $($_.UserPrincipalName)"
    Write-Host "Policy: '$($_.ApplicationAccessPolicy)'"
}
```

```
Write-Host "Configured Policy Name: '$TeamsPolicyName'"

# ----------------------------
# GET CURRENT POLICY ASSIGNMENTS
# ----------------------------
Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Getting Current Policy
Assignments..."

$PolicyUsers = foreach ($user in Get-CsOnlineUser) {

    if ($null -ne $user.ApplicationAccessPolicy) {

        $CurrentPolicy = $user.ApplicationAccessPolicy.ToString().ToLower()

        if ($CurrentPolicy -eq $TeamsPolicyName.ToLower()) {

            $user.UserPrincipalName.ToLower()
        }
    }
}

$PolicyUsers = $PolicyUsers | Sort-Object -Unique

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - $($PolicyUsers.Length) Existing
Policy Users found"

# ----------------------------
# DELTA CALCULATION
# ----------------------------
Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Calculating Delta..."
$UsersToAdd = $GroupUsers | Where-Object { $_ -notin $PolicyUsers }
$UsersToRemove = $PolicyUsers | Where-Object { $_ -notin $GroupUsers }

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - $($UsersToAdd.Length) users to add"
Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - $($UsersToRemove.Length) users to
remove"

# ----------------------------
# APPLY CHANGES
# ----------------------------
```

```
Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Applying Changes..."
foreach ($upn in $UsersToAdd) {
    try {

        Grant-CsApplicationAccessPolicy `
            -Identity $upn `
            -PolicyName $TeamsPolicyName `
            -ErrorAction Stop

        Write-Log "ADD" $upn
    }
    catch {
        Write-Log "ERROR_ADD" $upn
    }
}

foreach ($upn in $UsersToRemove) {
    try {

        Grant-CsApplicationAccessPolicy `
            -Identity $upn `
            -PolicyName $null `
            -ErrorAction Stop

        Write-Log "REMOVE" $upn
    }
    catch {
        Write-Log "ERROR_REMOVE" $upn
    }
}

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Disconnecting from Microsoft
Teams..."

Disconnect-MicrosoftTeams -Confirm:$false

Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] - Done"

exit 0
```

Recommended script capabilities:

- Microsoft Graph group enumeration
- Teams policy assignment/removal
- Delta comparison logic
- Logging and error handling
- Certificate-based authentication

Summary

This automation provides scalable and governed user assignment management for Microsoft Teams Application Access Policies.

The solution compensates for the lack of native group-based policy assignment support within Teams by:

- Using Entra ID groups as the source of truth
- Automating user assignment and removal
- Maintaining operational consistency and governance

This approach significantly reduces manual administration while improving access control accuracy and auditability.

Microsoft Teams Retention Policy Configuration in Microsoft Purview

Overview

This configuration implements a Microsoft Teams retention policy using Microsoft Purview Data Lifecycle Management to automatically retain and delete Teams chat and channel messages after a defined retention period.

The policy helps organisations:

- Align with data governance and retention requirements
- Reduce unnecessary long-term data storage
- Support GDPR data minimisation principles

This implementation applies retention to:

- Teams 1:1 chats
- Teams group chats
- Teams meeting chats
- Teams channel messages

Once the retention period expires, messages are automatically deleted and are no longer recoverable through standard user access or Data Subject Access Requests (DSARs).

Prerequisites

Required licensing:

• Microsoft 365 A3/A5 or E3/E5 equivalent licensing
• Microsoft Purview Data Lifecycle Management access

Required roles and permissions:

• Compliance Administrator
or
• Purview Data Lifecycle Management Administrator

Dependencies:

• Microsoft Teams enabled in tenant
• Microsoft Exchange Online enabled
• Microsoft Purview portal access

Preparation tasks:

• Review existing retention and eDiscovery policies
• Confirm no conflicting legal hold requirements exist
• Prepare user communications prior to go-live

Step 1: Access Microsoft Purview

1. Open the Microsoft Purview portal:

https://purview.microsoft.com

2. Sign in using an administrative account with appropriate permissions.
3. From the left-hand navigation pane, select:

Solutions > Data lifecycle management > Retention policies

4. Select:

- New retention policy

Step 2: Configure Policy Scope

1. Enter a policy name.

Recommended naming convention:

Teams Chat Retention – 2 Years *(enter the number of years that Teams messages will be retained before deletion.)*

2. Optionally provide a description including:

• Business justification
• Approval reference
• RFC number
• Implementation date

Example:

Retains Teams chat and channel messages for 2 years in alignment with institutional data retention policy.

3. Select:

Static

4. Continue to location configuration.

Step 3: Target Resources / Components

Select the following Microsoft 365 locations:

 Teams chats
 Teams channel messages

Recommended scope:

• All users (enterprise-wide deployment)

Optional phased rollout:

• Select specific pilot users or groups during testing phase

> **Important:**

This configuration targets:

• Teams 1:1 chats
• Group chats
• Meeting chats
• Channel conversations

This policy does NOT apply to:

• Teams meeting recordings
• SharePoint files
• OneDrive files
• Email retention

These workloads require separate retention policies.

Step 4: Configure Retention Settings

1. Choose:

Retain items for a specific period

2. Configure retention duration:

2 Years

3. Start retention based on:

When items were created

Recommended rationale:

Using creation date ensures predictable and consistent lifecycle management and prevents indefinite retention caused by message edits.

4. After the retention period:

Delete items automatically

5. Review the policy summary carefully before proceeding.

Recommended configuration:

• Retain content for 2 years
• Automatically delete after expiry

Step 5: Access Control / Enforcement

Policy enforcement is handled automatically by Microsoft Purview once deployed.

Important operational notes:

• Users cannot bypass centrally managed retention settings
• Messages deleted after expiry are permanently removed from standard access
• eDiscovery or Litigation Hold policies override deletion where applicable

Why this matters:

This ensures:

• Consistent organisational retention enforcement
• Reduced data sprawl

• Improved compliance posture
• Controlled information lifecycle management

Step 6: Testing / Report Mode

Recommended safe rollout approach:

Phase 1: Pilot Deployment

1. Deploy policy to:

• IT administrators
• Small pilot group
• Test accounts

2. Validate:

• Messages remain accessible during retention period
• Older content is identified correctly
• No unexpected user impact occurs

3. Perform test exports using Microsoft Purview eDiscovery.

Validate:

• Teams chats can be searched
• Export formats are usable
• Retention scope behaves as expected

Suggested test scenarios:

• 1:1 chat retention
• Group chat retention
• Meeting chat retention
• eDiscovery export validation

Important:

Retention processing may take several days to fully apply due to Microsoft background processing.

Step 7: Monitoring & Validation

Monitoring locations:

Microsoft Purview Portal:

Solutions > Data lifecycle management > Retention policies

Validation checks:

• Policy status shows enabled
• Locations are correctly assigned
• No policy errors present

Additional monitoring:

• Review Purview audit logs
• Validate Teams message lifecycle behaviour
• Confirm retention processing completion

Expected behaviour:

• Messages older than retention threshold become unavailable
• Users lose access to expired messages
• Content becomes unavailable through standard retrieval methods

Troubleshooting indicators:

• Policy not applying after several days
• Users outside expected scope
• Retention conflicts with existing compliance policies

Step 8: Enforcement / Go-Live

1. Expand policy scope to all required users.
2. Confirm organisational communications have been issued.

Important Considerations

• Teams chat retention does not automatically apply to Teams meeting recordings
• Recordings stored in OneDrive or SharePoint require separate retention configuration
• Deleted messages cannot be recovered once permanently removed
• Data Subject Access Requests cannot retrieve deleted content after retention expiry
• Litigation Hold or eDiscovery Hold overrides retention deletion
• Retention processing is not immediate and may take several days

Potential operational impact:

• Users may lose access to historical conversations
• Teams should not be used as a long-term records repository

Operational recommendations:

• Store important records in SharePoint or approved document systems
• Avoid relying on Teams chat for permanent record keeping
• Validate eDiscovery export functionality periodically

Summary

This implementation deploys a Microsoft Teams retention policy through Microsoft Purview to retain Teams chat and channel messages for a defined period before automatic deletion.

The policy supports:

• Data minimisation
• Compliance requirements
• Reduced long-term data exposure
• Consistent lifecycle management across collaboration platforms

Once enforced, Teams chat content exceeding the retention period is automatically and permanently deleted in accordance with organisational policy.