Automatic mapping of Shared Mailbox in M365

Problem

ESG could not handle for permission reason, the mapping of Microsoft365 delegated user and the mapping as secondary email addresses for users.

ESG integrates with Microsoft 365 via the Graph API for user synchronization and Single Sign-On (SSO). This process creates Shared Mailboxes as ‘Functional Users’ within ESG’s user management system. A limitation of the Graph API is its lack of support for detailed permission information regarding Shared Mailbox access. Consequently, ESG cannot determine which users have permissions to utilize specific Shared Mailboxes and therefore cannot accurately map these as secondary addresses within the ESG platform.

Solution :

Prerequisites

In order to automate the mapping of shared to their corresponding members in ESG some prerequisites must be in place:

- ESG integration with Microsoft 365 must be configured and working
- At least 1 import in ESG must be done from with M365 integration
- Shared Mailboxes are present in ESG in the User Management

This will restrict the activity to only mapping the shared mailbox to existing ESG users according to the member list extracted via Powershell.

Automatic mapping via Powershell & ESG APIs

Info : We provide this script as an example and is intended to be a template for the actual Implementation. Please remind to test it properly before using it in production environment.This script should be scheduled to keep updated the mapping of the shared mailbox user secondary addresses.

M365SharedMailboxMappingTemplate.ps1

```
#Import Exchange Online module, it must be installed first here
Import-Module ExchangeOnlineManagement

Connect-ExchangeOnline -UserPrincipalName <M365 Tenant admin>
# Get all shared mailboxes
$sharedMailboxes = Get-Mailbox -RecipientTypeDetails SharedMailbox
-ResultSize Unlimited

$shared_mailbox_list = @()

Get-Mailbox -RecipientTypeDetails SharedMailbox -ResultSize
Unlimited | ForEach-Object {
    $mailbox = $_

    # Retrieve the permissions for the mailbox: filter for
FullAccess (non-inherited and non-system accounts)
    $permissions = Get-MailboxPermission -Identity
$mailbox.PrimarySmtpAddress | Where-Object {
        $_.AccessRights -contains "FullAccess" -and `
        -not $_.IsInherited -and `
        $_.User -notmatch "NT AUTHORITY|S-1-5-|SELF|SYSTEM"
    }
    # Process each permission entry
    foreach ($perm in $permissions) {
        $delegatedUser = $perm.User.ToString()

        $isGroup = $false
        # Try to determine if the delegated user is a group using
Get-Recipient
        try {
            $recipient = Get-Recipient -Identity $delegatedUser -
ErrorAction Stop
            if ($recipient.RecipientType -match "Group") {
                $isGroup = $true
            }
        }
        catch {
            # If not found or error occurs, assume it's not a
```

```
group.
            $isGroup = $false
        }
        if ($isGroup) {
            # Expand the group: get each member using Get-
DistributionGroupMember
            $groupMembers = Get-DistributionGroupMember -Identity
$delegatedUser -ErrorAction SilentlyContinue
            if ($groupMembers) {
                foreach ($member in $groupMembers) {
                    $shared_mailbox_list += [PSCustomObject]@{
                        SharedMailbox      = $mailbox.DisplayName
                        PrimarySmtpAddress =
$mailbox.PrimarySmtpAddress
                        ExternalObjectId   =
$mailbox.ExternalDirectoryObjectId
                        User               =
$member.PrimarySmtpAddress.ToString()
                        AccessRights       = ($perm.AccessRights -
join ", ")
                    }
                }
            }
        }
        else {
            # Not a group: add one object with the delegated user
identity
            $shared_mailbox_list   += [PSCustomObject]@{
                SharedMailbox      = $mailbox.DisplayName
                PrimarySmtpAddress = $mailbox.PrimarySmtpAddress
                ExternalObjectId   =
$mailbox.ExternalDirectoryObjectId
                User               = $delegatedUser
                AccessRights       = ($perm.AccessRights -join ",
")
            }
        }
    }
}
```

```
# Display the results in a formatted table could be uncommented to
see all the shared mailbox mapped.
#$shared_mailbox_list

$shared_unique = $shared_mailbox_list.PrimarySmtpAddress | Group-
Object


# Number of users returned per page in each GET, 150 is the maximum
chunk.
$page_size = 150
$page_num = 1

# Define the ESG API endpoint to get the list of users with maximum
page size (150)
$apiUrl =
"<youresgAddress>/api/v2/user?page="+$page_num+"&itemsPerPage=150"

# Define headers for the api call, use your apiToken generated
$headers = @{
    'Accept' = 'application/hal+json'
    'X-ESG-Auth-Token' = 'apiToken'
}

# Send the GET request
$get_users_response = Invoke-RestMethod -Uri $apiUrl -Method Get -
Headers $headers

$users_list_body = $get_users_response._embedded.item

# Number of total users extracted from the GET call
$total_users = $get_users_response.totalItems
Write-Host "Total users" $total_users


# Check if number of total users is < than page size, if yes no
multiple calls are needed to loop through pages
if ($total_users -lt 50){
    Write-Host "user page page > total"
    $users_list = $users_list_body
```

```
} else {

    # Number of total users is > than page size, looping through
pages to get all users
    Write-Host "users per page < total, looping through pages"

    while ($true){

        # Check if GET request for page_num is empty, meaning we
reached end of user list
        if ($get_users_response.PSObject.Properties.Name -
notcontains '_embedded') {

            break

        } else {

            # Go to next page
            $page_num += 1

            # GET request of current page is not empty
            $apiUrl = "<youresgAddress>/api/v2/user?page=" +
$page_num +"&itemsPerPage=150"



            # Send the GET request
            $get_users_response = Invoke-RestMethod -Uri $apiUrl -
Method Get -Headers $headers



            # Appending list of users of each page
            $users_list_body += $get_users_response._embedded.item
        }

    }
}

# Filter only users that are members of the Shared mailbox list
$users_list = $users_list_body| Where-Object
{$shared_mailbox_list.User -contains $_.username}
Write-Host "total number of users to map: " $users_list.count
```

```
#Filter only shared mailbox , this could be also ->
$shared_mailbox_list.PrimarySmtpAddress
$mailbox_list_values = $users_list_body| Where-Object
{$shared_mailbox_list.SharedMailbox -contains $_.username}
Write-Host "total number of Mailbox to remap: "
$mailbox_list_values.count

#delete all shared mailbox saved as functional_users

# Loop all users and check if the shared mailbox is already present
as an alias, if not add the Shared Mailbox as alias

foreach ($user in $users_list){
    # ID of the ESG user
    $id = $user.id
    Write-Host "Esg user id:" $id

    $apiUrl = "<youresgAddress>/api/v2/user/$id"

    # Loop each Shared mailbox
    foreach($shared in $shared_mailbox_list){
    $match = $user.emailAddresses | Where-Object {
        $_.PSObject.Properties['address'].Value -eq $shared.User
    }
        if ($match){
        $shared_mailbox = [PSCustomObject]@{
            address = $shared.PrimarySmtpAddress
            active = $true
            primary = $false
            }
        $headers = @{
                "Accept" = "application/hal+json"
                "Content-Type" = "application/merge-patch+json"
                "X-ESG-Auth-Token" = "apiToken"
            }
            $user.emailAddresses = $user.emailAddresses +
$shared_mailbox
            $body = [PSCustomObject] @{ emailAddresses =
$user.emailAddresses } | ConvertTo-Json
            $patch_response = Invoke-RestMethod -Uri $apiUrl -
Method Patch -Headers $headers -Body $body
```

```
        }
    }
}
#delete all the shared mailbox that are functional users. Iterate
all functional_users
foreach($functional in $mailbox_list_values)
{
    $id = $functional.id

    $apiUrl = "<youresgAddress>/api/v2/user/$id"
    # Define headers
    $headers = @{
        'Accept' = 'application/json'
        'X-ESG-Auth-Token' = 'apiToken'
    }

$response = Invoke-RestMethod -Uri $apiUrl -Method Get -Headers
$headers

#Functional user role, is it needed to check if the retrieved user
is functional, il that case it will remove it (After having
remapped it)
$functionalUser = "/api/v2/user-role/499"
$otherFunctionalUser = "/api/v2/user-role/400"
if($response.role -eq $functionalUser)
{
$apiUrl = "<youresgAddress>/api/v2/user/$id"
$apiUrl
# Define headers
$headers = @{
'Accept' = 'application/json'
'X-ESG-Auth-Token' = 'apiToken'
}
$response = Invoke-RestMethod -Uri $apiUrl -Method DELETE -Headers
$headers
Write-Host "Deleting the user ..."
}
if($response.role -eq $otherFunctionalUser)
{
$apiUrl = "<youresgAddress>/api/v2/user/$id"
$apiUrl
```

```
# Define headers
$headers = @{
'Accept' = 'application/json'
'X-ESG-Auth-Token' = 'apiToken'
}
$response = Invoke-RestMethod -Uri $apiUrl -Method DELETE -Headers
$headers
Write-Host "Deleting the user ..."
}

}

Write-Host  "all shared mailbox are now mapped"
```