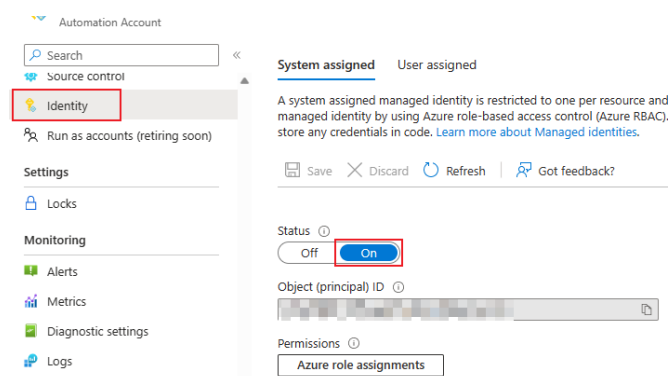# Set scope tag by domain of primary user with automation

This automation solves a very small specific use case. As soon as the mobile devices (IOS, Android) are registered in Intune and are not set up via enrollment type profiles, no more scope tags can be set based on device parameters. This is where this automation comes into play.

The script runs regularly and goes through all IOS & Android devices in Intune. There it takes the primary user of the device and resolves the domain to the scope tag via an API. Then the script sets the scope tag in Intune via the API.

Prerequisites

Managed Identity

On the Automation Account you must activate the managed identity. This can be achieved in the settings under "Identity":

Then you can add the managed identity of the Azure Automation Account to the Storage Account with the permissions of "Storage Account Contributor":

Selected role	Storage Account Contributor		
Assign access to	○ User, group, or service principal		
	● Managed identity		
Members	+ Select members		

Name	Object ID	Type
intune-automation		Automation Account ⓘ

Description	Optional

App Registration

After that you have to create an App Registration. The process to create and how the variables can be found out, you will find in this tutorial: Get app details and gr... | LNC DOCS (lucanoahcaprez.ch)

The permissions on the App Registration that are necessary for this automation are as follows:

- Device.ReadWrite.All
- User.Read.All
- DeviceManagementConfiguration.ReadWrite.All,
- DeviceManagementRBAC.ReadWrite.All
- DeviceManagementManagedDevices.ReadWrite.All

PowerShell Script

With the following script this automation can be achieved. This is intended to be executed in a runbook. Accordingly, it is important that the variables are added to the runbook and automation account.

```
Param(
    $Device,
    $Domain
)

$Success = @()
$NoPM = @()
$NoScopeTag = @()
$NoShortName = @()
$NoOwner = @()
$NoDevice = @()
```

```powershell
#Azure Login with Identity
try
{
    # "Logging in to Azure..."
    Connect-AzAccount -Identity
}
catch {
    Write-Error -Message $_.Exception
    throw $_.Exception
}


$resourcegroupname = "<yourresourcegroupname>"
$storageaccountname = "<yourstorageaccountname>"
$storagetablename = "<yourstoragetablename>"


#storage account/table connection
$storageaccountkey = (Get-AzStorageAccountKey -ResourceGroupName $resourcegroupname -AccountName
$storageaccountname | where {$_.keyname -eq "key1"}).value
$storageContext = New-AzStorageContext -StorageAccountName $storageaccountname -StorageAccountKey
$storageaccountkey
$table = Get-AzStorageTable -name $storagetablename -context $storageContext
$hostnames = get-AzTableRow -table $table.CloudTable


#Collect Credential Data for Graph API
$tenantId = Get-AutomationVariable -Name "<yourunbooktenantidvariable>"
$ClientId = Get-AutomationVariable -Name "<yourunbookclientidvariable>"
$CredentialObject = Get-AutomationPSCredential -Name "<yourrunbookclientsecretsecret>"
$ClientSecret = $CredentialObject.GetNetworkcredential().password
$WebhookURI = Get-AutomationVariable -Name "<yourteamswebhookuri>"
$FunctionURL = Get-AutomationVariable -Name "<yourfunctionapiurl>"


$Body = @{
    "tenant" = $TenantId
    "client_id" = $ClientId
    "scope" = "https://graph.microsoft.com/.default"
    "client_secret" = $ClientSecret
    "grant_type" = "client_credentials"
}


$Params = @{
```

```powershell
    "Uri" = "https://login.microsoftonline.com/$TenantId/oauth2/v2.0/token"
    "Method" = "Post"
    "Body" = $Body
    "ContentType" = "application/x-www-form-urlencoded"
}
$AuthResponse = Invoke-RestMethod @Params

$Header = @{
    "Authorization" = "Bearer $($AuthResponse.access_token)"
    #"ConsistencyLevel" = "eventual"
}

Function Send-Logs(){

    param (
        [String]$LogType,
        [Hashtable]$LogBodyList
    )


    $LogBodyJSON = @"
    {
        "logtype": "$LogType",
        "logbody": {}
    }
"@

    $LogBodyObject = ConvertFrom-JSON $LogBodyJSON
    Foreach($Log in $LogBodyList.GetEnumerator()){
        $LogBodyObject.logbody | Add-Member -MemberType NoteProperty -Name $log.key -Value $log.value
    }
    $Body = ConvertTo-JSON $LogBodyObject

    $Response = Invoke-Restmethod -uri $FunctionURL -Body $Body -Method POST -ContentType
"application/json"
    return $Response
}

function Send-ToTeams {
    $CurrentTime = Get-Date
```

```powershell
    $Body = @{
"@context" = "https://schema.org/extensions"
"@type" = "MessageCard"
"themeColor" = "880808"
"title" = "Automation completed:"
    "text" = @"
Parameter:
<ul><li>Successful assignments in Intune: $($SuccessChanged.Count)</li><li>Already correct assignment in
Intune: $($SuccessAlready.Count)</li><li>Scope tag not found in Intune:
$($NoScopeTag.Count)</li><li>Shortname by domain not found in storage table:
$($NoShortName.Count)</li><li>Owner not found in Intune: $($NoOwner.Count)</li></ul>
"@
}

    $JsonBody = $Body | ConvertTo-JSON

    Invoke-RestMethod -Method Post -Body $JsonBody -Uri $WebhookURI -Headers @{"content-type" =
"application/json; charset=UTF-8"} | out-null
}

#get all Scope Tags
$uri = "https://graph.microsoft.com/beta/deviceManagement/roleScopeTags"
$Results = Invoke-RestMethod -Method GET -Uri $uri -ContentType "application/json" -Headers $header
$scopeTags = $results.value

if($device){

    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices/$($device)" -ContentType
"Application/Json" -Header $Header
    $managedDevices += $Results
    # $managedDevices
}elseif($Domain){
    Add-Type -AssemblyName System.Web

    $encodedURL = [System.Web.HttpUtility]::UrlEncode($Domain)

    #get android and ios managed devices
    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=(contains(activationlockbypass
```

```
code,%20%27$encodedURL%27))%20and%20startswith(operatingSystem,'ios')" -ContentType "Application/Json"
-Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }
    $managedDevices += $ResultsValue


    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=(contains(activationlockbypass
code,%20%27$encodedURL%27))%20and%20startswith(operatingSystem,'android')" -ContentType
"Application/Json" -Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }

    $managedDevices += $ResultsValue

    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices/$($device)" -ContentType
"Application/Json" -Header $Header
    $managedDevices += $Results
    # $managedDevices
```

```
}else{
    #get android and ios managed devices
    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'io
s')" -ContentType "Application/Json" -Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }
    $managedDevices += $ResultsValue


    $Results = Invoke-RestMethod -Method GET -Uri
"https://graph.microsoft.com/v1.0/deviceManagement/managedDevices?`$filter=startswith(operatingSystem,'an
droid')" -ContentType "Application/Json" -Header $Header
    $ResultsValue = $Results.value
    if ($results."@odata.nextLink" -ne $null) {
        $NextPageUri = $results."@odata.nextLink"
        ##While there is a next page, query it and loop, append results
        While ($NextPageUri -ne $null) {
            $NextPageRequest = (Invoke-RestMethod -Headers $Header -Uri $NextPageURI -Method Get)
            $NxtPageData = $NextPageRequest.Value
            $NextPageUri = $NextPageRequest."@odata.nextLink"
            $ResultsValue = $ResultsValue + $NxtPageData
        }
    }

    $managedDevices += $ResultsValue
}

write-output "Intune Devices: $($managedDevices.count)"

#compare and collect MEID devices, which are managed by intune
```

```powershell
foreach($managedDevice in $managedDevices){
    #configure hostname if device has an owner
    if($managedDevice.userPrincipalName){
        #get domainname of user
        $dn = ($managedDevice.userprincipalname -split "@")[1]
        $shortname = ($hostnames | where {$_.dn -eq $dn}).shortname
        if($shortname.count -gt 1){
            $shortname = $shortname[0]
        }
        if($shortname){
            #check if scope tag exists based on azure table
            $scopeTagName = "SCT-INT-$shortname-INTUNE-*-PROD"
            $scopeTag = $scopeTags | where {$_.displayName -like $scopeTagName}
            if($scopeTag){

                $uri =
"https://graph.microsoft.com/beta/deviceManagement/managedDevices('$($managedDevice.id)')"
                $DeviceObject = Invoke-RestMethod -Uri $uri -Headers $header -Method GET -ContentType
"application/json"
                if($DeviceObject.roleScopeTagIds -contains $scopeTag.id){
                    $SuccessAlready += $managedDevice.id
                }
                else{
                    $assignBody = @{
                        roleScopeTagIds = @("$($scopeTag.id)")
                    }

                    $JSON = $assignBody | ConvertTo-Json
                    Invoke-RestMethod -Uri $uri -Headers $header -Method Patch -Body $JSON -ContentType
"application/json"
                    $SuccessChanged += $managedDevice.id
                }

            } else {
                write-warning "$($managedDevice.id) - Scope Tag $scopeTagName does not exist!"
                $NoScopeTag += $managedDevice.id
            }
        } else {
            write-warning "$($managedDevice.id) - Domain $dn not in Storage Table!"
            $NoShortName += $managedDevice.id
```

```
        }
    } else {
        write-warning "$($managedDevice.id) - Device has no Owners"
        $NoOwner += $managedDevice.id
    }
}


$Logs = @{
    "successchangedassignment"="$($SuccessChanged.count)"
    "successalreadyassigned"="$($SuccessAlready.count)"
    "errornoscopetag"="$($NoScopeTag.count)"
    "errornoshortname"="$($NoShortName.count)"
    "errornoowner"="$($NoOwner.count)"
}


Send-Logs -LogType "MobileDeviceScopingByPrimaryUserExecutions" -LogBodyList $Logs


Send-ToTeams
```