IANS | Online Training

LAB GUIDE

Analyzing Microsoft 365 through PowerShell

Breakout Session Two – Setting foundation for Non-Interactive Logon

Overview

Running PowerShell scripts manually is good but running them on a scheduled task is even better. In this module, create a service principal to do much of the heavy lifting and a limited-permissioned user account to cover gaps in Microsoft's API/permission model.

IANS

Procedure

1. Open PowerShell and signon to Azure Ad with the global admin permissions:

   Code:
   ```
   Connect-AzureAd
   ```

```
Do Not Panic > Connect-AzureAd

Account                    Environment TenantId

-------                    ----------- --------
connor@peoplesdynamics.com AzureCloud  3a6daad1-6fd0-4543-8a8
```

2. Create a new application, called TenantScanner. This generates an application we can assign credentials to during the scan. Save the output to a variable for use in later commands.
   Code:
   ```
   $App = New-AzureADApplication -DisplayName TenantScanner
   ```
   ```
   Do Not Panic > $App = New-AzureADApplication -DisplayName TenantScanner
   ```

3. Create a self-signed certificate on your machine to assign to the application. This provides more security than a password by requiring access to the local certificate store rather than a string which is more easily transferred from system to system.
   Code:
   ```
   $Subject = “CN=TenantScanner“
   $CertStore = “Cert:\CurrentUser\My”
   $Cert = New-SelfSignedCertificate -Subject $Subject -CertStoreLocation $CertStore -KeyExportPolicy
   Exportable -KeySpec Signature -KeyLength 2048 -KeyAlgorithm RSA -HashAlgorithm SHA256
   ```

```
Do Not Panic > `
>> $Subject = "CN=TenantScanner";`
>> $CertStore = "Cert:\CurrentUser\My";`
>> $Cert = New-SelfSignedCertificate -Subject $Subject `
>>         -CertStoreLocation $CertStore `
>>         -KeyExportPolicy Exportable `
>>         -KeySpec Signature `
>>         -KeyLength 2048 `
>>         -KeyAlgorithm RSA `
>>         -HashAlgorithm SHA256
```

4. Export the certificate for use with Azure Ad.

   Code:

```
Export-Certificate -Cert $cert -FilePath .\TenantScanner.cer
```

```
Do Not Panic > Export-Certificate -Cert $cert -FilePath .\TenantScanner.cer


    Directory: C:\Users\ConnorPeoples


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----         11/1/2021   2:00 PM            782 TenantScanner.cer
```

5. Associate the certificate with the Application created in step 2. This allows certificate based signon for supported modules.

   Code:

```
New-AzureADApplicationKeyCredential `
 -ObjectId $App.ObjectId `
 -CustomKeyIdentifier "TenantScanner" `
 -StartDate $Cert.NotBefore `
 -EndDate $Cert.NotAfter `
 -Type AsymmetricX509Cert `
 -Usage Verify `
 -Value $([System.Convert]::ToBase64String($Cert.GetRawCertData()))
```

```
Do Not Panic > `
>> New-AzureADApplicationKeyCredential `
>>     -ObjectId $App.ObjectId `
>>     -CustomKeyIdentifier "TenantScanner" `
>>     -StartDate $Cert.NotBefore `
>>     -EndDate $Cert.NotAfter `
>>     -Type AsymmetricX509Cert `
>>     -Usage Verify `
>>     -Value $([System.Convert]::ToBase64String($Cert.GetRawCertData()))
```

6. Save the tenant and application information to local files which will be needed for signon later.

   Code:

```
$Tenant = Get-AzureADTenantDetail
$Tenant.ObjectId | Out-File TenantId.txt
$App.AppId | Out-File AppId.txt
$App.ObjectId | Out-File AppObjectId.txt
```

7. In a new PowerShell session, navigate to the same directory as before.
8. Create some easy variables based on the file contents for use during signon. Then, use those credentials to signon to AzureAd.
   Code:

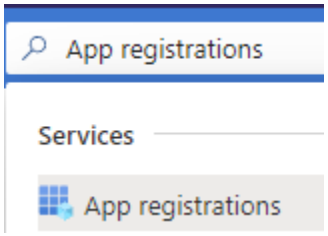
```
$Cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate("$((Get-
Location).Path)\TenantScanner.cer")
$ThumbPrint = $Cert.GetCertHashString()
$TenantId = Get-Content .\TenantId.txt
$AppId = Get-Content .\AppId.txt
Connect-AzureAd -TenantId $TenantId -ApplicationId $AppId -CertificateThumbprint $Thumbprint
```

```
Five Steps Ahead >  $Cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate("$((Get-Location).Path)\TenantScanner.cer");
>> $ThumbPrint = $Cert.GetCertHashString();
>> $TenantId = Get-Content .\TenantId.txt;
>> $AppId = Get-Content .\AppId.txt;
>> Connect-AzureAd -TenantId $TenantId -ApplicationId $AppId -CertificateThumbprint $Thumbprint
```

   Take note of those commands as you'll need them in our script later.

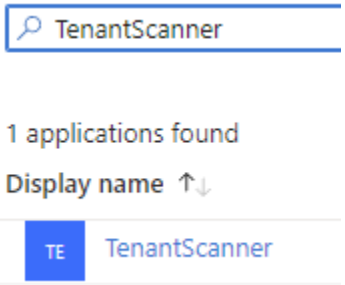
9. Now, we need to establish some permissions in the backend for the user account and for the service principal to access the Microsoft services. Login to https://portal.azure.com/.

10. In the top, search for “App registrations” and select “App Registrations” from Services.

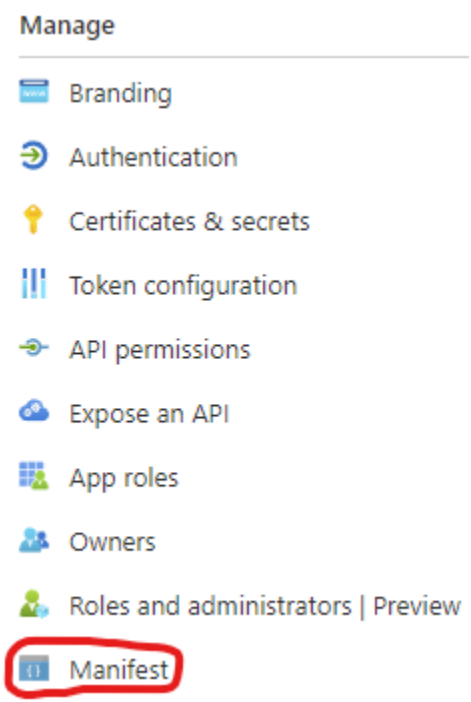


11. Select the “All Applications” tab in the content pane.

12. Search for “TenantScanner” and select it from the list of apps.



13. Open “Manifest” on the left column navigation

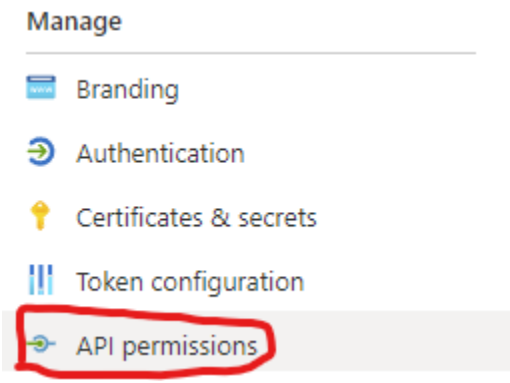


14. Update the requiredResourceAccess to enable ExchangeOnline access, then press save. The updated JSON is below:

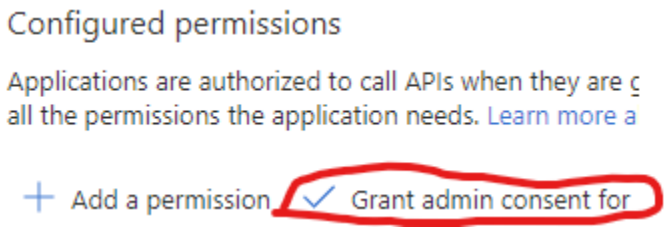
```
"requiredResourceAccess": [
    {
        "resourceAppId": "00000002-0000-0ff1-ce00-000000000000",
        "resourceAccess": [
            {
                "id": "dc50a0fb-09a3-484d-be87-e023b12c6440",
                "type": "Role"
            }
        ]
```

```
    }
],
```

15. Select “API Permission” from the left navigation column.

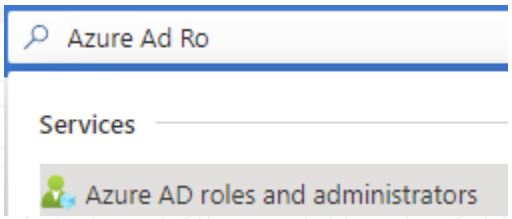


16. Select “Grant admin consent for …” to approve the ExchangeOnline access provided in step 22.

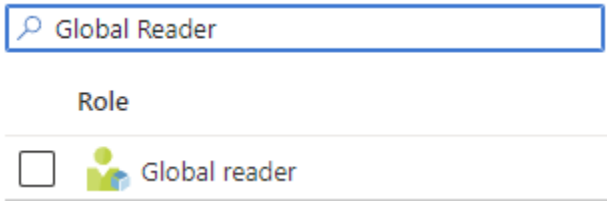


17. Click “Yes” on the confirmation dialog. The service principal can now access ExchangeOnline.

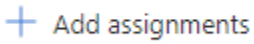
18. In the top of the Azure Portal, search for “Azure Ad roles and administrators”, and select it from the Services list.



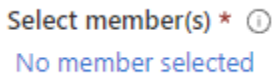
19. Search for “Global reader” and select it from the list.



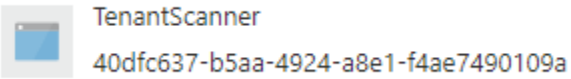
20. On the top of the content pane, select “Add assignments”



21. Select “No Members Selected”

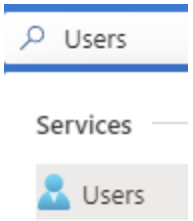


22. In the Add Assignments dialog, search for TenantScanner and select the service principal.

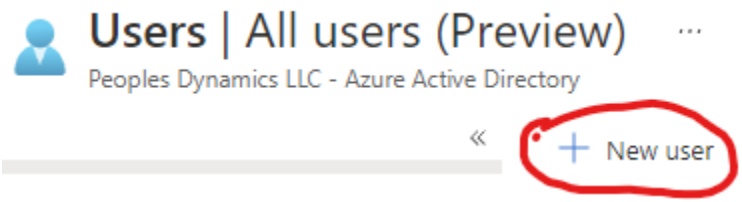


23. Select “Add” to finish adding the assignment. The service principal now has global reader rights to the organization.

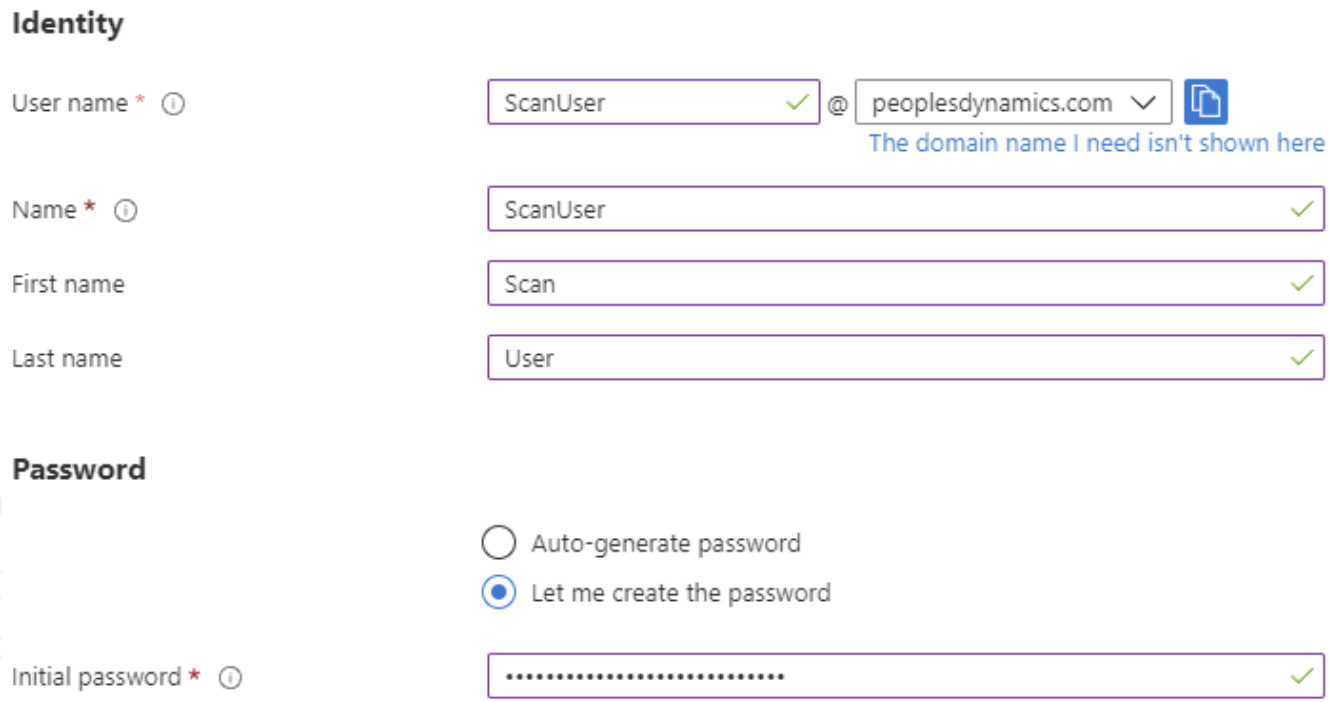
24. The next step is to set up the scan user account which has limited access to run headlessly. In the top search bar, search for and select “Users.”



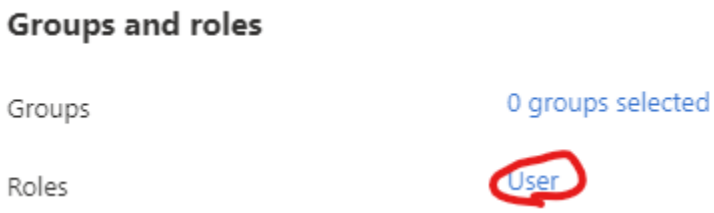
25. Select “New User” from the content pane.



26. Create the identity for the user and set the password.



27. Select the roles button to assign roles.



28. Add the following roles:
    a. Global Reader
    b. SharePoint Administrator

29. Select "Create" on the bottom of the page. If you are notified that role creation failed, try adding the role directly to the user after creation. https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-users-assign-role-azure-portal

30. With the Service Principal and limited user account created, it's time to get back into Powershell. Open the prompt for PowerShell and run a logon command with the new scan user credentials

```
Connect-MsolService
```

Note: This will force you to change your password.

31. Let's now focus on creating user credentials for re-use. Start by gathering the credentials of the user account you want to automatically call Msol, SharePoint or Microsoft Teams endpoints.
    Code:

```
$Credential = Get-Credential
```

```
Do Not Panic > $Credential = Get-Credential
```

32. Export the credentials to file for future reference. This uses a secure string representation of the password rather than the cleartext password. We encourage looking into your organization's best practices to align secret storage with their policies.

Code:

```
$Credential.UserName | Out-File username.txt
$Secret = $Credential.Password | ConvertFrom-SecureString
$Secret | Out-File secret.txt
```

```
Do Not Panic > `
>> $Credential.UserName | Out-File username.txt; `
>> $Secret = $Credential.Password | ConvertFrom-SecureString; `
>> $Secret | Out-File usersecret.txt
```

33. Using the stored credentials, access the MSOL command.

Code:

```
$Username = Get-Content .\username.txt
$Secret = Get-Content .\secret.txt | ConvertTo-SecureString
$Cred = New-Object System.Management.Automation.PSCredential -ArgumentList ($Username, $Secret)
Connect-MsolService -Credential $Cred
```

```
Do Not Panic > `
>> $Username = Get-Content .\Username.txt;`
>> $Secret = Get-Content .\secret.txt | ConvertTo-SecureString;`
>> $Cred = New-Object System.Management.Automation.PSCredential -ArgumentList ($Username, $Secret); `
>> Connect-MsolService -Credential $Cred
```

34. Now both the service principal and user account are configured and available locally for automation scanning. The breakdown of when to use each is below:

Module	Login Command	Identity	Test Command
AzureAd	Connect-AzureAd	TenantScanner	Get-AzureAdApplication
AzureRm	Login-AzureRmAccount	TenantScanner	Get-AzureRmAdApplication
ExchangeOnline	Connect-ExchangeOnline	TenantScanner	Get-OrganizationConfig
Msol	Connect-MsolService	ScanUser	Get-MsolDomain
SharePoint	Connect-SpoService -Url	ScanUser	Get-SpoTenant
Microsoft Teams	Connect-MicrosoftTeams	ScanUser	Get-CsTeamsMeetingBroadcastPolicy

35. Leveraging the credentials, now create a powershell script that logs into each service listed above and validates it can call each test command.