LAB GUIDE

Analyzing Microsoft 365 through PowerShell

Breakout Session Three – Know your operators.

Overview

This breakout session focuses on making the evaluation of properties with a M365 tenant more dynamic rather than straight string evaluations. To do this, we consider input modifiers and comparison operators to manipulate input from PowerShell commands. With the right modifiers, we can evaluate any given input and ensure it aligns with the expected result.

Manipulators Worth Reviewing

Operators

-eq; straight comparison which evaluates that the left equals the right

- Useful when a specific return value is expected.

-ne; comparison which evaluates that the left does not equal the right

- Useful when multiple return states are valid, but one is not.

-in; comparison which evaluates if an input is found in a pre-defined list

- Useful when multiple return states are valid, but multiple are not.

-lt, -le; comparison which evaluates that the left is less than (or equal) the right

- Useful when looking for policies which should have a data no larger than X

-gt, ge; comparison which evaluate that the left is greater than (or equal) to the right

- Useful when looking at counts of items, policies, MFA devices, etc.

Modifiers

.Count; returns a count of items from an array.

- Useful when the total number of items is more important than their individual values.
- # of MFA methods is a great example.

.Length; returns the string length of a result.

- Useful when a default value is null or not set.

.ToString; returns the result as a string.

- This can be very helpful when dealing with certain Boolean-ish types. 1, True, and $True might all be used to show that a property is set to true. However, “2” -eq $True will evaluate as true which makes straight comparison more difficult. More on this below.

Snippets

The following snippets reflect some of my learnings from automating these kinds of controls. This is not meant to be an exhaustive list of examples but should go a long way in showing how to think about the data responses you may deal with across the M365 platform.

.Count, -lt example

The count example is great for making sure there are lists of things and for tracking changes in lists. As an area of security interest, it is recommended that users have multiple methods of MFA in place so that SSPR can always require MFA even when resetting one credential.

```
$Users = Get-MsolUser -All
ForEach($User in $Users) {
    If ($User.StrongAuthenticationMethods.Count -lt 2) {
        Write-Host "$($User.DisplayName) does not have sufficient MFA methods."
    }
}
```

.Length, -gt example

Some policies have no functional value, but cause issues when evaluated against a $Null statement. As a result, using the length parameter can provide a known sum when working with unset properties.

```
$CompanyInfo = Get-MsolCompanyInformation
If ($CompanyInfo.DirSyncServiceAccount.Length -gt 0) {
    Write-Host "Directory Sync Service Account:
    $($CompanyInfo.DirSyncServiceAccount)"
}
```

Disclaimer: this particular property responds to $Null, but the example is still valid.

-in/notin example

Using a list of known values can be important when there are multiple (in)correct options that could be returned from a single value.

```
$DeviceAdmins = @('Joe', 'Tony', 'Andy')
$DevRegPolicy = Get-MsolDeviceRegistrationServicePolicy
ForEach($User in $DevRegPolicy.Users) {
    If ($User -notin $DeviceAdmins) {
        Write-Host "Unexpected user allowed to AD Join $($User)"
    }
}
```

-ne example

Sometimes only a single value is bad from a list of potential returns. To save on processing time, you can check to see that the returned response is anything but the bad response.

```
$DevRegPolicy = Get-MsolDeviceRegistrationServicePolicy
If ($DevRegPolicy.AllowedToAzureAdJoin -ne "All") {
    Write-Host "You have a decent Azure AD Join Policy."
} Else {
    Write-Host "You should probably check your Azure AD Join Policy."
}
```