

Greater Region 9 Seminar

Microsoft 365 API Scripting

September 23, 2026 – Detroit, MI

Joey Cusimano, ICF – STACS Software Engineer – Baltimore, MD

Session Agenda

- PowerShell 7
- Microsoft Graph
 - Modules
 - Scopes
 - Example Modules
- Data Handling
- Azure Resource Graph

- NOTE: Local administrative privileges are required if installing PowerShell 7.



PowerShell 7

- PowerShell 5: default, packaged, only gets security updates
- PowerShell 7: Faster, modern .NET architecture, actively receiving features
- PowerShell 7 requires manual installation (w/ admin privileges)
 - PS > winget install --id Microsoft.PowerShell --source winget (“winget source update” first!)
 - msi package from <https://github.com/PowerShell/PowerShell/>
 - Windows store
- PowerShell 7 does not replace 5.1
 - Choose when opening new tab and can set default
 - Scripts written in one won't work in the other
 - Just like Python 2 and 3
 - No magic “convert” function, but can use Microsoft tool “PSScriptAnalyzer” to help identify version conflicts
 - If making portable scripts (i.e. to be run on workstations), consider 5.1

Microsoft Graph

- REST API endpoint for Microsoft products
- Authenticate w/user account (or service account) set up in Entra
- Returns JSON objects you can work with in any scripting/programming language
 - Produce readable text file for showing/emailing
 - Produce .csv for sorting, filtering, etc. in Excel
 - Work with large data sets more easily than the UI can process/handle them
- No extra cost unless marked as “metered”, then billed monthly per 1,000 objects.

Microsoft Graph - Modules

- Libraries/packages that contain code for specific uses (i.e., M365 API calls)
- Modules must be installed (once) and imported (in every script before their use)
- Good practice: do a check-and-install at the beginning of your script while importing

```
# Install modules if necessary
$modules = @('Microsoft.Graph')
foreach ($module in $modules) {
    if (-not (Get-Module -ListAvailable -Name $module)) {
        Install-Module -Name $module -Force -AllowClobber -Scope CurrentUser
    }
    Import-Module $module
}
```

- Key Examples
 - ExchangeOnlineManagement: retrieve mail forwarding rules (& more)
 - Microsoft.Graph: retrieve user sign-in logs (& more)

Microsoft Graph - Scopes

- Attempt to connect to the API with the specified permission set
 - Success – can use privileged functions
 - Failure – insufficient roles/permissions in Azure/M365 (or you fail the auth prompt, or no valid cached auth token is available, possibly due to incorrect password, password reset, expired password, disabled account...)
- Example: view user's sign-in logs requires two:
 - Directory.Read.All (because it's user/directory data)
 - AuditLog.Read.All (because it's audit data)
- Good practice: define them as part of setup right before connecting

```
# Connect to Microsoft Graph with required scopes
$scopes = @('AuditLog.Read.All', 'Directory.Read.All')
Connect-MgGraph -Scopes $scopes -NoWelcome
```

Example Modules

- Get-MgUser
 - Good way to check if a given username exists before continuing the script and passing it to another module
 - Can retrieve user's GUID or display name for other modules or inclusion on reports

```
PS C:\Users\37246\OneDrive - ICF\Files\Scripts> & .\training_m365_single_user_logins.ps1 secops@ndc.org
⌚ Start Time: 09/17/2026 15:58:16

DisplayName      Id      Mail      UserPrincipalName
-----
                96beede5-6df0-4d25-9296-6bec3e62d31f  @      @
```

- Get-MgAuditLogSignIn
 - Retention period of logs limited to your tenancy
 - Entra ID P1 or P2 License (Business Premium has P1 included!!!): 30 days
 - No Entra ID P1/P2 license: 7 days

Example Modules (cont.)

- Get-MgAuditLogSignIn (cont.)
 - Can get success logs only
 - Can get error logs to troubleshoot or monitor failed access attempts
 - Great for forensics if an incident is suspected or confirmed – remember, preserve logs past 30 days outside of M365 before they are lost!!!

UserPrincipalName	LastSignInLocal	LastSignInLocation	LastSignInIP
@	9/17/2026 6:31:19 AM	Glen Burnie, Maryland, US	68.134.201.
@	9/15/2026 12:20:43 PM	Glen Burnie, Maryland, US	68.134.201.
@	9/8/2026 2:22:22 PM	Glen Burnie, Maryland, US	68.134.201.
@	9/8/2026 11:21:10 AM	Glen Burnie, Maryland, US	68.134.201.
@	9/1/2026 4:24:35 PM	Severn, Maryland, US	68.134.201.
@	9/1/2026 10:30:18 AM	Severn, Maryland, US	68.134.201.
@	9/1/2026 10:12:27 AM	Severn, Maryland, US	68.134.201.
@	8/20/2026 11:05:45 AM	Glen Burnie, Maryland, US	68.134.201.
@	8/19/2026 11:24:32 AM	Glen Burnie, Maryland, US	68.134.201.

Error Code	Meaning
0	Success
50034	User account not found in tenant
50053	Account locked (Smart Lockout)
50055	Password expired
50057	Account disabled
50058	No user session / silent sign-in failed
50076	MFA required
50126	Invalid username or password
53003	Blocked by Conditional Access policy

Data Handling

- Some modules like Get-MgAuditSignIn take specific params including conditional statements:
 - -Filter "userPrincipalName eq '\$Username' and status/errorCode eq 0"
 - -Top 20
- Objects can be manipulated w/PowerShell after returned by the module:
 - | Sort-Object -Property createdDateTime -Descending
 - | Where-Object { \$_.status.errorCode -eq 0 }
 - | Select-Object -First {x}
- Tip: prefer the former, but save a more expansive query to save for seeing the full set
 - Retrieves only what you need, keeping report content and filesize low and scripts finishing more quickly
 - Allows finding other attributes to add to the script later

Tips and Tricks - Troubleshooting

- Dump variables to console or file and analyze JSON separately (especially nested properties)
 - VSCode: format JSON file with (Alt +Z), has colors (winget install Microsoft.VisualStudioCode --source winget)
 - Notepad++: JSON viewer plugin (winget install Notepad++.Notepad++ --source winget)
 - Most programming IDEs have a plugin or native support just to format .json files
 - Turn on word-wrap for readability if the results are a long, single line
 - Ensure your data is saved as a file.json or the editor's formatting function may not work properly
- Include code to handle blank values when data is unreliable
 - Example: FixedVersion on securityresources is often empty, code should handle it appropriately
- Cross-reference the UI to validate results (links are good to include in SOP)
 - [Sign-in logs \(Entra\)](#) – Monitoring & Health -> Sign-In Logs
 - [Vulnerability findings \(inc. software updates\)](#) (Defender) – Exposure Mgmt -> Vuln Mgmt -> Vulnerabilities

Azure Resource Graph

- Browser-based UI for writing/running queries in KQL to get JSON data as output
- Download results as CSV that can be processed with scripts
- Example: locate unpatched software flagged on Azure VMs

The screenshot shows the Azure Resource Graph Explorer interface. At the top, the breadcrumb is "Dashboard > Microsoft Defender for Cloud | Recommendations". The main title is "Azure Resource Graph Explorer". Below the title, there are two tabs: "Software Updates" (active) and "[Schema] securityresources". The interface includes a search bar, a "Run" button, and "Save" and "Share" options. On the left, there is a "Categories" and "Tables" section with a search bar and a list of resources: "advisorresources", "aksresources", and "alertsmanagementresources". The main area displays a KQL query:

```
1 securityresources
2 | where type == "microsoft.security/assessments"
3 | where properties.metadata.recommendationCategory == "SoftwareUpdate"
4 | extend VMResourceId = tostring(properties.resourceDetails.Id)
5 | extend VMName = extract(@"virtualMachines/([^\s/]+)", 1, VMResourceId)
6 | extend SoftwareName = tostring(properties.additionalData.SoftwareName)
7 | extend StatusChangeDate = tostring(properties.status.statusChangeDate)
8 | extend ScannerDetails =
9   parse_json(tostring(properties.additionalData.ScannersDetails))
```

Azure Resource Graph – Example Output

```
=====
Host: Workstation_2
=====
(2026-09-04 15:35) chrome: ["152.0.7977.83"] → 153.0.8010.37

=====
Host: Workstation_1
=====
(2026-09-10 02:31) chrome: ["152.0.7977.83"] → 153.0.8010.37
(2026-09-09 13:01) sql_server_2019: ["15.0.4480.2"] → 15.0.4490.9

Servers requiring OS updates (reboot schedule: every 3rd Saturday at 3AM EST):
[Workstation_1] (2026-09-15 00:20) - windows_server_2022 (["10.0.20348.5499"])
[Workstation_2] (2026-09-15 00:20) - windows_server_2019 (["10.0.17763.9121"])
[Workstation_3] (2026-09-15 00:20) - windows_server_2019 (["10.0.17763.9121"])

Other Required Updates with Undetermined Target Versions:
[Workstation_1] (2026-08-31 13:46) - entra_connect (["2.6.3.0"])

[Workstation_2] (2026-09-15 00:20) - .net (["8.0.29.0"])
[Workstation_2] (2026-09-09 13:01) - .net_framework (["4.7.2.0", "3.5.0.0", "4.0.0.0", "3.0.0.0", "2.0.0.0"])
[Workstation_2] (2026-09-15 00:20) - asp.net_core (["8.0.29.0"])

[Workstation_3] (2026-09-09 13:02) - .net_framework (["4.7.2.0", "4.0.0.0", "3.0.0.0", "2.0.0.0", "3.5.0.0"])

[Workstation_4] (2026-09-15 00:20) - .net (["9.0.19.0"])
[Workstation_4] (2026-09-09 13:12) - .net_framework (["2.0.0.0", "4.7.2.0", "3.0.0.0", "3.5.0.0", "4.0.0.0"])
[Workstation_4] (2026-09-15 00:20) - asp.net_core (["9.0.19.0"])
```

Other Scripting Examples

- Get all mail forwarding rules in exchange (useful after a compromise!).
- Get all sign in logs for all users.
- Get users who currently don't have MFA setup.
 - Useful for that push to get everyone to do it – but you can be more heavy-handed in the policies as well.
- Get all public IP addresses in an Azure tenancy across all devices.
 - Identifies unknown devices, retired devices that are taking up space, and provides a list to give for IP scanning.

Scripts can also perform edit/update/delete actions, but this must be done with CAUTION.

- Revoke all current login sessions.
- Force a password change for a user (or all users).

