Azure AD Default Configuration Blunders

Following the release of [two recent blogs](#_References) regarding Microsoft’s Azure Active Directory default configurations, we began digging a little further into the access an unprivileged user has inside any tenant running any of the default settings in their tenant.

What we’ve found is that the Default Enabled settings allow any unprivileged, authenticated user to create apps, create security groups, read all users’ full profiles, read all security groups’ full profiles, as well as read nearly all tenant settings, configurations, and information.

We started by [creating a test tenant](https://office365itpros.com/2021/04/15/test-development-office-365-tenant-test-new-features/) in Azure and populating it with sample data. We kept the default configurations, including the Microsoft Security Defaults configured on all tenants when they’re created. We then created a new, unprivileged user in the tenant and configured the MFA requirements.

Once the account was ready, we opened PowerShell and connected to two modules: AzureAD, and MSOnline.

Graphical user interface, text, email, website

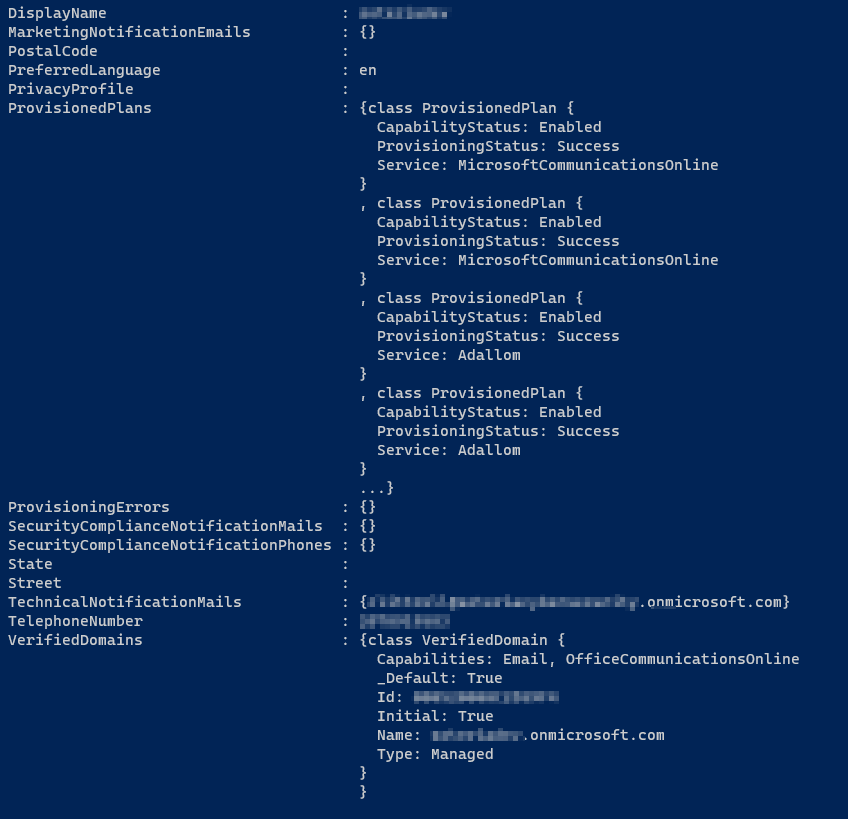
Description automatically generated

We needed to get to know the environment, so we started by getting the tenant information.

Get-AzureADTenantDetail | fl

Text

Description automatically generated



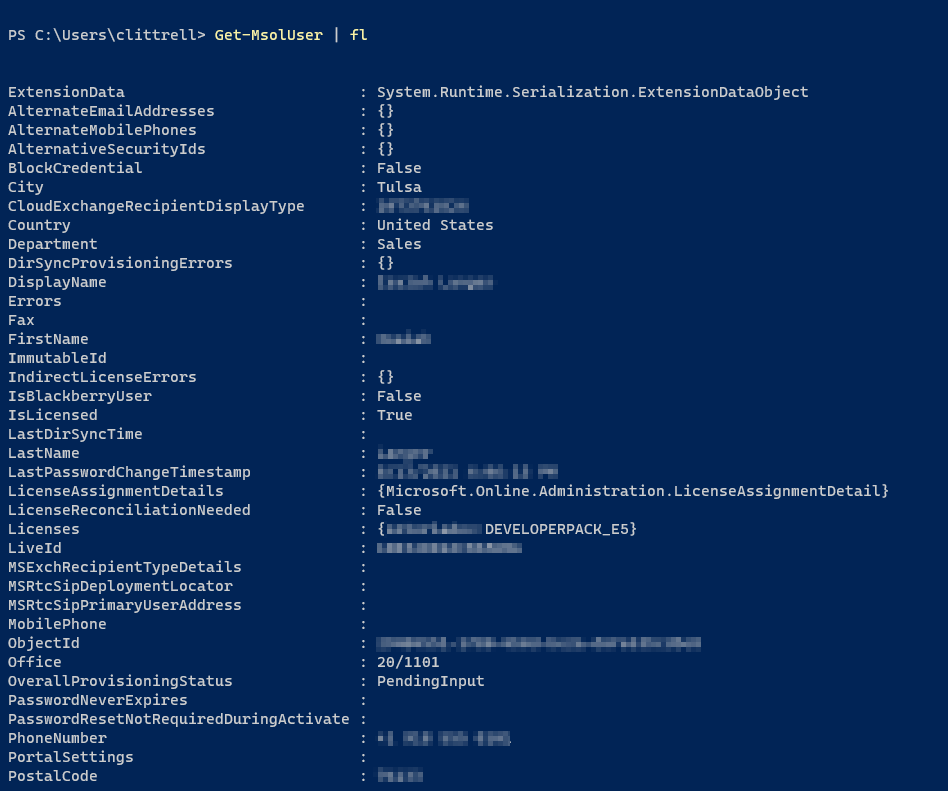
Next, we wanted to figure out just how much access we had with our shiny, new user account.

Running “Get-MsolCompanyInformation | fl” as any unprivileged, authenticated user allows you to determine how much access and information you can garnish from a tenant. Alternatively, you can run “Get-AzureADMSAuthorizationPolicy | select -ExpandProperty defaultuserrolepermissions | fl” and see if all users can create security groups, view all user information, create apps, add users, etc...

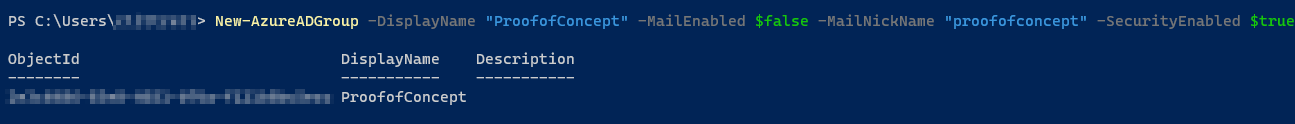


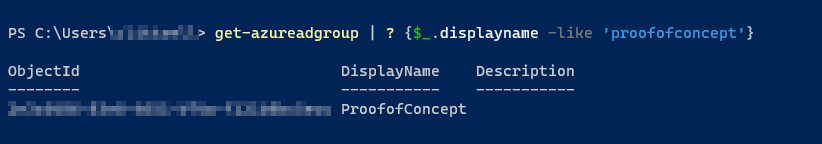
As a standard, unprivileged user we see that we can read other users’ accounts in Azure AD, and we can create new Security Groups.

Running “Get-MsolUser” as any unprivileged, authenticated user allows you to read the full details of any user's account - this is a default setting that is Enabled



Now, let’s have some fun and create a new Security Group



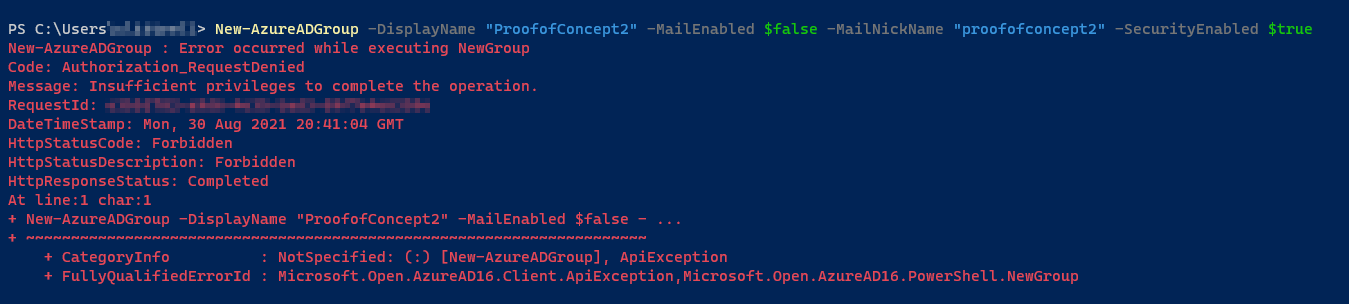


The above can be mitigated by running “Set-MsolCompanySettings -UsersPermissionToReadOtherUsersEnabled $false” and “Set-MsolCompanySettings -UsersPermissionToCreateGroupsEnabled $false” respectively, as a Global Admin.

Once those settings are disabled, the user should see an “Access Denied” message when attempting either action.

Text, website

Description automatically generated



The AzureAD PowerShell module also bypasses any MFA requirements – even if MFA is enabled and enforced on all accounts, just an FYI.

Revisiting those default configuration settings, we found that there are a few others that may be interesting to review. Using the command “Get-AzureADMSAuthorzationPolicy” gives you some visibility into default user authorization configurations.

Graphical user interface, application

Description automatically generated

The “AllowEmailVerifiedUsersToJoinOrganization” default setting is Enabled. [According to Microsoft](https://docs.microsoft.com/en-us/azure/active-directory/enterprise-users/directory-self-service-signup), this allows self-service creation of an account where the user does not exist, but the email domain is listed in the accepted domains.

This could potentially allow us to create a new user with any of the domain suffixes that a tenant owns or lists as "Accepted Domains" (eg, [thisuserdoesnotexist@secondaryorg.com](mailto:thisuserdoesnotexist@secondaryorg.com)).

It’s possible that leveraging that, with default configuration settings in place, we could then potentially read all user account information via Azure AD PowerShell, paving the way for account takeover, administrative account enumeration, or any number of other malicious actions.

So now that we’ve found these holes, can we do anything about them?

Short answer – yes and no.

# Recommendations:

While we understand that not every finding can be remediated at this time, we have compiled a list of items from other resources and our own findings that should help shore up your defenses.

* Restrict Access to Azure AD Administration Portal
  + Log in to the Azure portal using a Global Administrator account
  + Navigate to Azure Active Directory
  + Navigate to User Settings
  + Toggle the “Restrict access to Azure AD administration portal” to Yes
* Restrict Access to invite Guests into the tenant – it is recommended to only allow Admins and Guest Inviter roles to invite Guests
  + Log in to the Azure portal using a Global Administrator account
  + Navigate to Azure Active Directory
  + Navigate to External Identities
  + Navigate to External collaboration settings
  + Toggle “Members can invite” to No
  + Toggle “Guests can invite” to No

* Restrict Access to Create Security Groups and Microsoft 365 Groups
  + Log in to the Azure portal using a Global Administrator account
  + Navigate to Azure Active Directory
  + Navigate to User Settings
  + Toggle the “Users can register applications” to No
  + Using PowerShell log in to the MSOnline module and run
    - Set-MsolCompanySettings -UsersPermissionToCreateGroupsEnabled $false
* Restrict Access to App Registrations, Application Consent, and Adding Gallery Apps
  + Using PowerShell log in to the MSOnline module and run
    - Set-MsolCompanySettings -UsersPermissionToUserConsentToAppEnabled $false
* Disable Legacy Authentication when possible
  + This can be configured via Conditional Access policies. To create a Conditional Access policy, log in to Azure Active Directory, Navigate to Security, and then to Conditional Access. Click New Policy and configure the policy with the below options.
    - Users – All (Create exclusions as needed)
    - Cloud Apps – Office 365 Exchange Online
    - Conditions
      * Client Apps
        + Select Mobile apps and Desktop Clients
        + Select Other Clients
      * Device Platforms – Any
      * Locations – Any
    - Block Access
  + Using Powershell, log in to the ExchangeOnlineManagement module
    - Get-Mailbox | Set-CasMailbox -PopEnabled $false -ImapEnabled $false  
      -SmtpClientAuthenticationDisabled $true -ActiveSyncEnabled $false
  + To prevent needing to repeat this process, the following can be run in PowerShell
    - Get-CASMailboxPlan | Set-CASMailboxPlan -PopEnabled $false -ImapEnabled $false  
      -SmtpClientAuthenticationDisabled $true -ActiveSyncEnabled $false
* Set the UsersPermissionToReadOtherUsersEnabled to False
  + Using PowerShell log in to the MSOnline module and run
    - Set-MsolCompanySettings -UsersPermissionToReadOtherUsersEnabled $false
* Verify that Self Service Join should be available, otherwise Set AllowEmailVerifiedUsersToJoinOrganization to False
  + Using PowerShell log in to the MSOnline module and run
    - Set-MsolCompanySettings –AllowEmailVerifiedUsersToJoinOrganization $false
* Unless needed, it would be recommended to also set AllowedToSignupEmailBasedSubscriptions to False
  + Using PowerShell log in to the MSOnline module and run
    - Set-MsolCompanySettings -AllowedToSignupEmailBasedSubscriptions $false

Beyond the settings and mitigations outlined in the blog posts that spurred this project, partial mitigation can be accomplished by leveraging Conditional Access policies that restrict access to various portions of the tenant’s administrative consoles and commands.

A Conditional Access policy applied to All Users (be sure to exclude your administrative accounts or groups) with the following settings will restrict access for users and attackers alike, preventing resources from being created on the organizations behalf.

To create a Conditional Access policy, log in to Azure Active Directory, Navigate to Security, and then to Conditional Access. Click New Policy and configure the policy with the below options.

Policy Settings:

* Users – All and Exclude Directory Role Global Administrator (and/or other admin roles)
* Cloud Apps – Microsoft Azure Management
* Conditions
  + Device Platforms – Any
  + Locations – Any
* Block Access

This policy, per [Microsoft’s documentation](https://docs.microsoft.com/en-us/azure/role-based-access-control/conditional-access-azure-management) will restrict user access to the following items:

* Azure portal
* Azure Resource Manager provider
* Classic Service Management APIs
* Azure PowerShell
* Visual Studio subscriptions administrator portal
* Azure DevOps
* Azure Data Factory portal
* Azure Event Hubs
* Azure Service Bus
* [Azure SQL Database](https://docs.microsoft.com/en-us/azure/azure-sql/database/conditional-access-configure)
* SQL Managed Instance
* Azure Synapse

However, it does not apply to Azure AD PowerShell, which calls Microsoft Graph.

Another possible option is to block the MSOnline module entirely – this is an all or nothing setting, and as such it is not a recommended step just yet. The below code from [o365blog](https://o365blog.com/post/limit-user-access/) performs the checks and sets the block.

# Get the access token

Get-AADIntAccessTokenForMSGraph -SaveToCache

# Disable the Msol PowerShell module access

Disable-AADIntTenantMsolAccess

# Check the settings after 10 seconds or so.

Get-AADIntTenantAuthPolicy | Select block\*

Since most of the commands we’ve leveraged in this post call AzureAD, and AzureAD’s PowerShell module calls the Graph API, this setting still leaves those holes wide open.

Microsoft does not make it easy to find ways to restrict access to Graph API’s, and after some research and community collaboration, we were pointed to Microsoft documentation and an existing repository [used in the Education space](https://docs.microsoft.com/en-us/schooldatasync/blocking-powershell-for-edu). Using the information provided in those scripts, we determined a method of restricting the PowerShell access for both Azure Active Directory, and Microsoft Graph. Steps for restricting this access are as follows:

* Define the app you want to restrict
  + To do this, you will need to find the ObjectID or the ApplicationID of the application. This can be found in the Sign-in logs (see screenshot below) after connecting to the application/module. For the two applications we are concerned with:
    - Azure Active Directory - 1b730954-1685-4b74-9bfd-dac224a7b894
    - Microsoft Graph - 14d82eec-204b-4c2f-b7e8-296a70dab67e
* Define the users or roles you want to grant access to
  + You may opt for an Azure AD Directory Role (eg, Global Administrator)
  + You may use a CSV file of Admins and Users containing their UPN (User Principal Name)
* This method requires that the application has a Service Principal, if one does not exist, you can create it for the specific application
  + Check for the Service Principal
    - $servicePrincipal = Get-AzureADServicePrincipal -Filter "appId eq '$appId'"
  + Create the Service Principal if it does not exist
    - if (-not $servicePrincipal) {$servicePrincipal = New-AzureADServicePrincipal -AppId $appId}
  + Set the Service Principal to require Assignment
    - Set-AzureADServicePrincipal -ObjectId $global:servicePrincipal.ObjectId -AppRoleAssignmentRequired $true
  + Set the Service Principal Assignment
    - New-AzureADServiceAppRoleAssignment -ObjectId $global:servicePrincipal.ObjectId -ResourceId $servicePrincipal.ObjectId -Id ([Guid]::Empty.ToString()) -PrincipalId $admin.ObjectID

Once completed, unauthorized users should be greeted with an error messaged after verifying their credentials.

Graphical user interface, text, application

Description automatically generated

Since remediation methods like those above may require approval, or may even not be feasible in your environment, detection should be the next viable method to ensure your environment is safe.

Detections for various SIEM solutions should focus on the Azure Sign-in logs keying on the User or Username and Application or Application ID fields.

Configuring a detection and alert for any users not explicitly authorized logging in via the Azure Active Directory PowerShell or Microsoft Graph application can alert you to Compromised Accounts, Insider Threats, and Unauthorized Access.

Text

Description automatically generated

# References:

1. <https://www.pentestpartners.com/security-blog/azure-ad-attack-of-the-default-config/>
2. <https://github.com/jonathandata1/microsoft_azure_personally_identifiable_info_leak>
3. <https://docs.microsoft.com/en-us/azure/active-directory/enterprise-users/directory-self-service-signup>
4. <https://office365itpros.com/2021/04/15/test-development-office-365-tenant-test-new-features/>
5. <https://o365blog.com/post/limit-user-access/>
6. <https://docs.microsoft.com/en-us/azure/role-based-access-control/conditional-access-azure-management>
7. <https://docs.microsoft.com/en-us/schooldatasync/blocking-powershell-for-edu>