Azure Privilege Escalation Using Managed Identities

written by Karl Fosaaen | February 20, 2020

Azure Managed Identities are Azure AD objects that allow Azure virtual machines to act as users in an Azure subscription. While this may sound like a bad idea, AWS utilizes IAM instance profiles for EC2 and Lambda execution roles to accomplish very similar results, so it's not an uncommon practice across cloud providers. In my experience, they are not as commonly used as AWS EC2 roles, but Azure Managed Identities may be a potential option for privilege escalation in an Azure subscription.

TL;DR – Managed Identities on Azure VMs can be given excessive Azure permissions. Access to these VMs could lead to privilege escalation.

Much like other Azure AD objects, these managed identities can be granted IAM permissions for specific resources in the subscription (storage accounts, databases, etc.) or they can be given subscription level permissions (Reader, Contributor, Owner). If the identity is given a role (Contributor, Owner, etc.) or privileges higher than those granted to the users with access to the VM, users should be able to escalate privileges from the virtual machine.

Important note: Anyone with command execution rights on a Virtual Machine (VM), that has a Managed Identity, can execute commands as that managed identity from the VM.

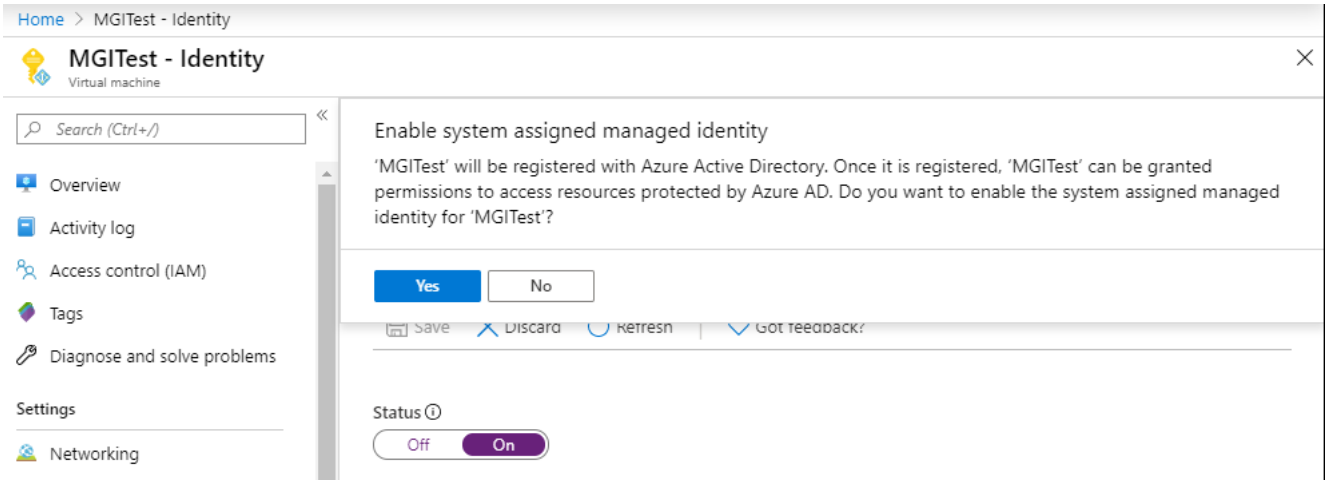
Here are some potential scenarios that could result in command execution rights on an Azure VM:

- Domain/Local user logins (RDP, PS Remoting, etc.)
- Application issues resulting in command execution
- Patch related issues resulting in command execution
- Azure IAM “Contributor” permissions on the VM
- Thick Application Breakouts

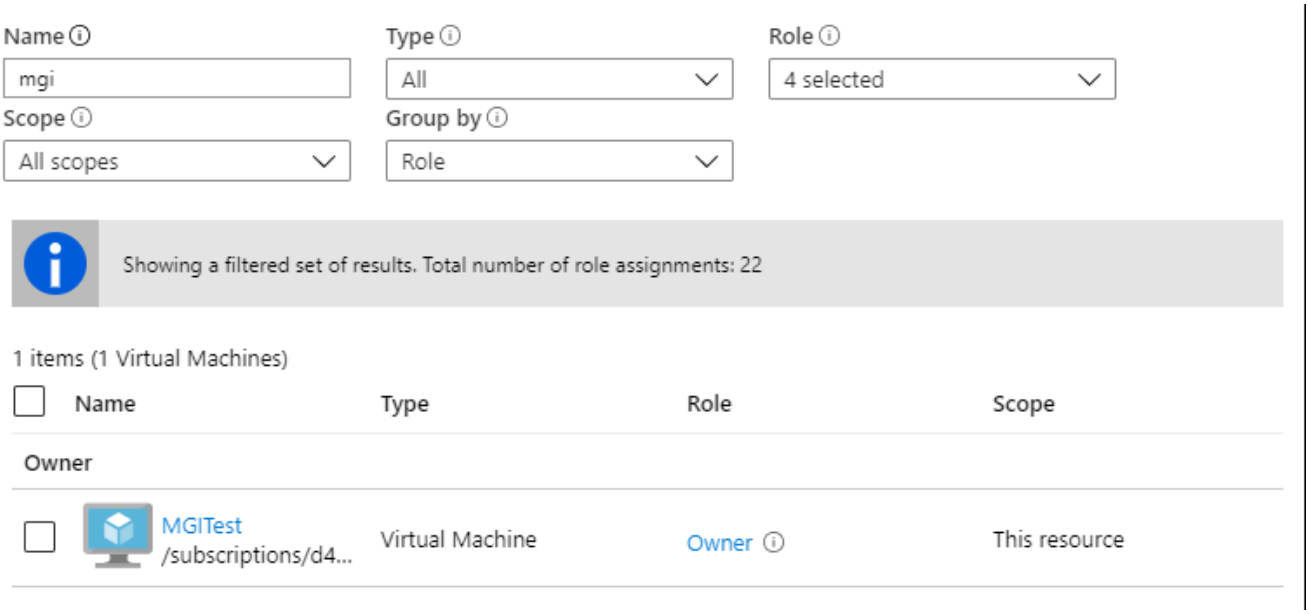
Identifying Managed Identities

In the Azure portal, there are a couple of different places where you will be able to identify managed identities. The first option is the Virtual Machine section. Under each VM,

there will be an “Identity” tab that will show the status of that VM’s managed identity.



Alternatively, you will be able to note managed identities in any Access Control (IAM) tabs where a managed identity has rights. In this example, the MGITest identity has Owner rights on the resource in question (a subscription).



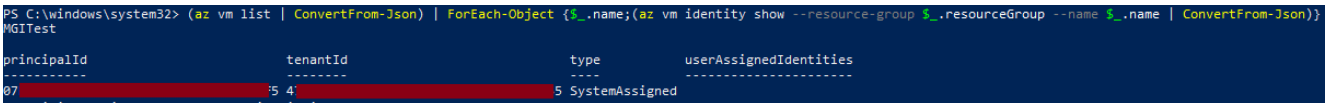
From the AZ CLI – AzureAD User

To identify managed identities as an authenticated AzureAD user on the CLI, I normally get a list of the VMs (az vm list) and pipe that into the command to show identities.

Here’s the full one-liner that I use (in an authenticated AZ CLI session) to identify managed identities in a subscription.

```
(az vm list | ConvertFrom-Json) | ForEach-Object {$_.name;(az vm identity show --resource-group $_.resourceGroup --name $_.name | ConvertFrom-Json)}
```

Since the principalId (a GUID) isn't the easiest thing to use to identify the specific managed identity, I print the VM name ($_.name) first to help figure out which VM (MGITest) owns the identity.



From the AZ CLI – On the VM

Let's assume that you have a session (RDP, PS Remoting, etc.) on the Azure VM and you want to check if the VM has a managed identity. If the AZ CLI is installed, you can use the "**az login –identity**" command to authenticate as the VM to the CLI. If this is successful, you have confirmed that you have access to a Managed Identity.

From here, your best bet is to list out your permissions for the current subscription:

```
az role assignment list -–assignee ((az account list | ConvertFrom-Json).id)
```

Alternatively, you can enumerate through other resources in the subscription and check your rights on those IDs/Resource Groups/etc:

```
az resource list
```

```
az role assignment list --scope "/subscriptions/SUB_ID_GOES_HERE/PATH_TO_RESOURCE_GROUP/OR_RESOURCE_PATH"
```

From the Azure Metadata Service

If you don't have the AZ CLI on the VM that you have access to, you can still use PowerShell to make calls out to the

Azure AD OAuth token service to get a token to use with the Azure REST APIs. While it's not as handy as the AZ CLI, it may be your only option.

To do this, invoke a web request to 169.254.169.254 for the oauth2 API with the following command:

```
Invoke-WebRequest -Uri 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https://management.azure.com/' -Method GET -Headers @{Metadata="true"} -UseBasicParsing
```

If this returns an actual token, then you have a Managed Identity to work with. This token can then be used with the REST APIs to take actions in Azure. A simple proof of concept for this is included in the demo section below.

You can think of this method as similar to gathering AWS credentials from the metadata service from an EC2 host. Plenty has been written on that subject, but here's a good primer blog for further reading.

Limitations

Microsoft does limit the specific services that accept managed identities as authentication – Microsoft Documentation Page

Due to the current service limitations, the escalation options can be a bit limited, but you should have some options.

Privilege Escalation

Once we have access to a Managed Identity, and have confirmed the rights of that identity, then we can start escalating our privileges. Below are a few scenarios (descending by level of permissions) that you may find yourself in with a Managed Identity.

- **Identity is a Subscription Owner**
  - Add a guest account to the subscription

        - Add that guest as an Owner
    - Add an existing domain user to the subscription as an Owner
        - See the demo below
- **Identity is a Subscription Contributor**
    - Virtual Machine Lateral Movement
        - Managed Identity can execute commands on another VMs via Azure CLI or APIs
    - Storage Account Access
        - Read files from Storage Accounts
        - See Cloud Shell Privilege Escalation
    - Configuration Access
        - Dump configuration information for services containing credentials
            - See Get-AzurePasswords
- **Identity has rights to other subscriptions**
    - Pivot to other subscription, evaluate permissions
- **Identity has access to Key Vaults**
    - Query Key Vaults for credential and secrets data
        - Local admin creds
        - Domain creds
        - See related Automation Account Scenario
- **Identity is a Subscription Reader**
    - Subscription Information Enumeration
        - List out available resources, users, etc for further use in privilege escalation

For more information on Azure privilege escalation techniques, check out my DerbyCon 9 talk:

Secondary Access Scenarios

You may not always have direct command execution on a virtual machine, but you may be able to indirectly run commands via Automation Account Runbooks.

I have seen subscriptions where a user does not have

contributor (or command execution) rights on a VM, but they have Runbook creation and run rights on an Automation account. This automation account has subscription contributor rights, which allows the lesser privileged user to run commands on the VM through the Runbook. While this in itself is a privilege inheritance issue (See previous Key Vault blog), it can be abused by the previously outlined process to escalate privileges on the subscription.

Proof of Concept Code

Below is a basic PowerShell proof of concept that uses the Azure REST APIs to add a specific user to the subscription Owners group using a Managed Identity.

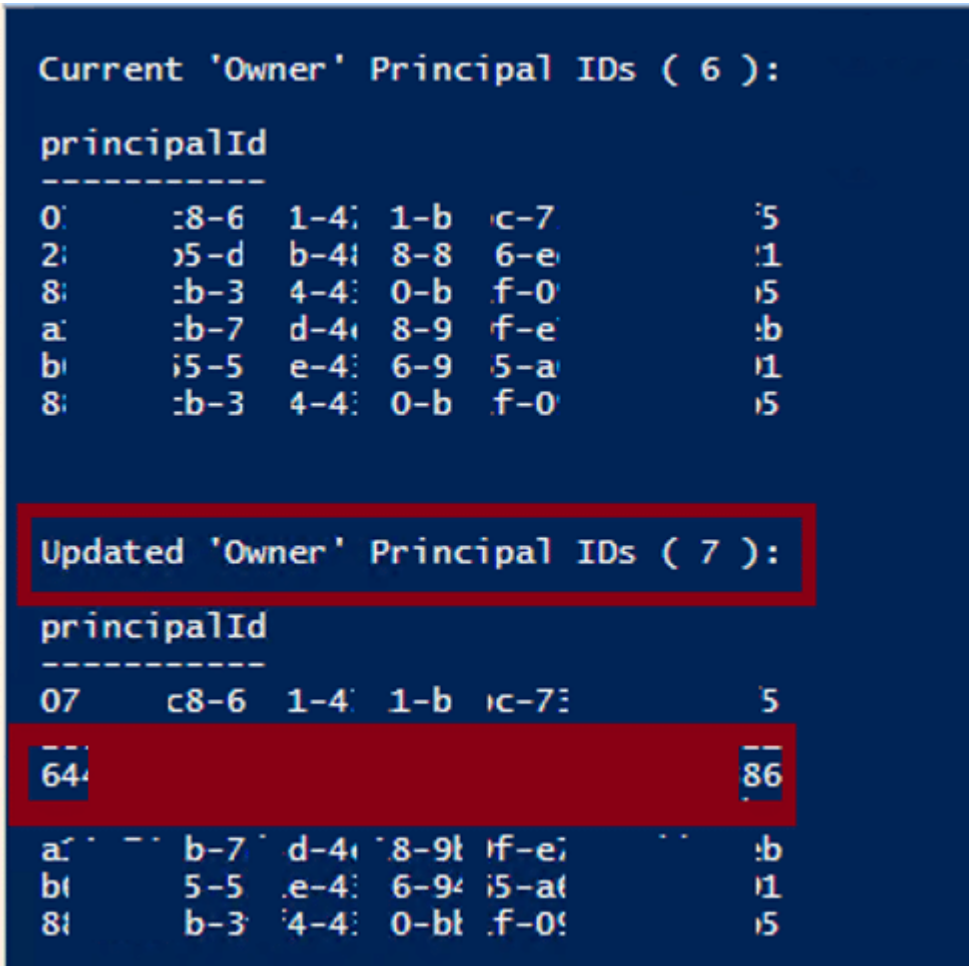
Proof of Concept Code Sample

All the code is commented, but the overall script process is as follows:

1. Query the metadata service for the subscription ID
2. Request an OAuth token from the metadata service
3. Query the REST APIs for a list of roles, and find the subscription “Owner” GUID
4. Add a specific user (see below) to the subscription “Owners” IAM role

The provided code sample can be modified (See: “CHANGE-ME-TO-AN-ID”) to add a specific ID to the subscription Owners group.

While this is a little difficult to demo, we can see in the screen shot below that a new principal ID (starting with 64) was added to the owners group as part of the script execution.



2/25/20 – Update:

I created a secondary PoC function that will probably be more practical for everyday use. The function uses the same methods to get a token, but it uses the REST APIs to list out all of the available storage account keys. So if you have access to an Azure VM that is configured with Contributor or Storage Account Contributor, you should be able to list out all of the keys. These keys can then be used (See Azure Storage Explorer) to remotely access the storage accounts and give you further access to the subscription.

You can find the sample code here – get-MIStorageKeys.ps1

Conclusion

I have been in a fair number of Azure environments and can say that managed identities are not heavily used. But if a VM is configured with an overly permissive Managed Identity, this

might be a handy way to escalate. I have actually seen this exact scenario (Managed Identity as an Owner) in a client environment, so it does happen.

From a permissions management perspective, you may have a valid reason for using managed identities, but double check how this identity might be misused by anyone with access (intentional or not) to the system.