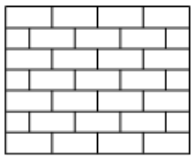
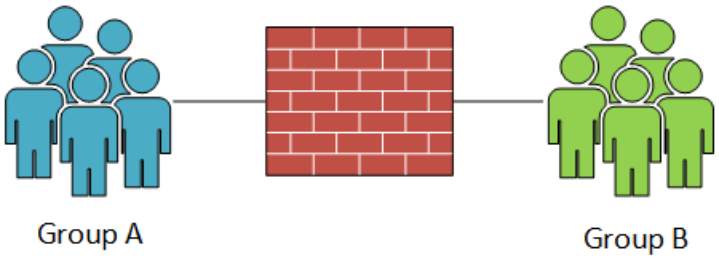


- 15 -
Information Barriers

Introduction

For its entire history, Office 365 did not provide technology to logically separate the people in an organization. Ideas like Ethical Firewalls did not exist in a native format. If we wanted this functionality, we would need a third-party product to provide the appropriate functionality. Recently Microsoft added a new concept called Information Barrier. While these cmdlets have been in the Security and Compliance Center since 2018, it's still not fully fleshed-out at the moment (this concept is limited to Teams communication) it is an important first step into this territory by Microsoft.

Not every organization will need a wall of separation like this. However, organizations like financial organizations or companies with heavy research and development teams may want to or have to (with regulations) restrict their users with whom they communicate within the company. In this chapter we will cover the requirements, prerequisites, steps for configuration and a detailed documentation script.



Information Barriers

https://docs.microsoft.com/en-us/office365/securitycompliance/information-barriers

What are Information Barriers?

Information Barriers are a logical construct that prevents communication between groups of people. Any of the people that are blocked from communicating need to be synced to Azure AD. The filters for users are based off of Azure AD users and the attributes that are allowed for filters. We can use one or more filters for filtering users. However, Microsoft recommends that these filters are not too complex.

Getting Started with Information Barriers

To use the Information Barrier functionality in the Security and Compliance Center, there are numerous prerequisites and caveats that you must deal with first. First, we'll start off with licensing requirements. From the official Microsoft documentation, the current licensing requirements can be found here:

https://docs.microsoft.com/en-us/office365/securitycompliance/information-barriers

Licensing Requirements

As with any feature in Office 365, Microsoft requires a license to enable the use of the feature. Information Barriers are certainly no different. When it comes to Information Barriers, you have a few choices:

- Microsoft 365 E5/A5

- Office 365 E5/A5/A3/A1
- Office 365 Advanced Compliance
- Microsoft 365 Compliance E5/A5
- Microsoft 365 Insider Risk Management

What you will notice is that you will need either an E5 or an add-on like Office 365 Advanced Compliance add-on license.

> **Office 365 Advanced Compliance:** contains these features Advanced eDiscovery, Advanced Data Governance, Privileged Access Management, Customer Key and Customer Lockbox

Once we have the right licensing in place, we can proceed to the correct permissions. From Microsoft's docs, the below security groups will have access to the Information Barrier configuration settings.

Permissions required

Permission requirements for Information Barriers are (one of the following):

- Microsoft 365 global administrator
- Office 365 global administrator
- Compliance administrator
- IB Compliance Management

The reason for this is that these groups all contain the correct Management Roles. There are two roles that are assigned to the groups. We can see the new Management Roles via PowerShell here:

```
Get-ManagementRole | where name -like '*ib compliance*'
```

```
Name                                RoleType
----                                --------
IB Compliance Management            IBComplianceManagement
View-Only IB Compliance Management ViewOnlyIBComplianceManagement
```

Take care in assigning users to any of the above Role Groups as they control communications between your end users in your tenant.

Filterable Attributes

Microsoft provides a list of user attributes that can be used in the user filters:

```
Co                  Company                                 Department
Description         Extension Attributes 1 to 15            Mail
Mail Nick Name      MSExch Ext. Custom Attr. 1 to 5         Member Of
Postal Code         Physical Delivery Office Name           Proxy Address
Street Address      Target Address                          Usage Location
User Principal Name
```

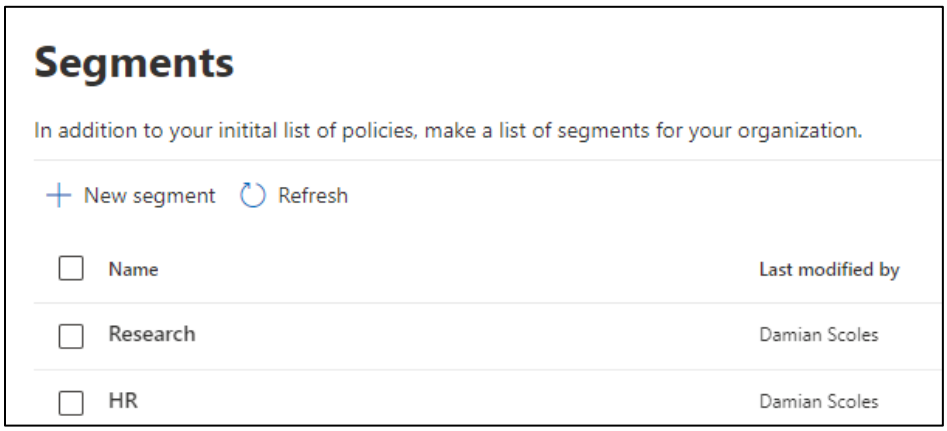
GUI or PowerShell?

Up until recently, the Information Barrier feature was not available in the Compliance Center and had to be managed with PowerShell. This situation has now changed and Information Barriers appear in the Compliance Center as a Preview feature. As this gets updated, we'll add more content to the book. For now, we will concentrate on the PowerShell side of this feature.

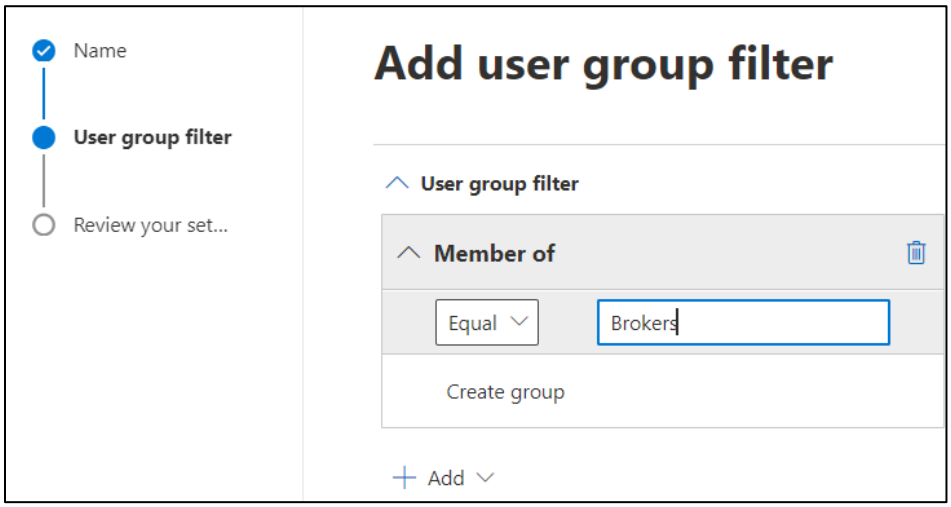
Information Barrier Segments

Withing the Compliance Center, a tab is now present for the Information Barriers Feature and is split into separate sub tabs for Segments, Policies and Policy Application. With this new interface we are able to create the same components we can in PowerShell or review the creating parts. First up we have Segments:

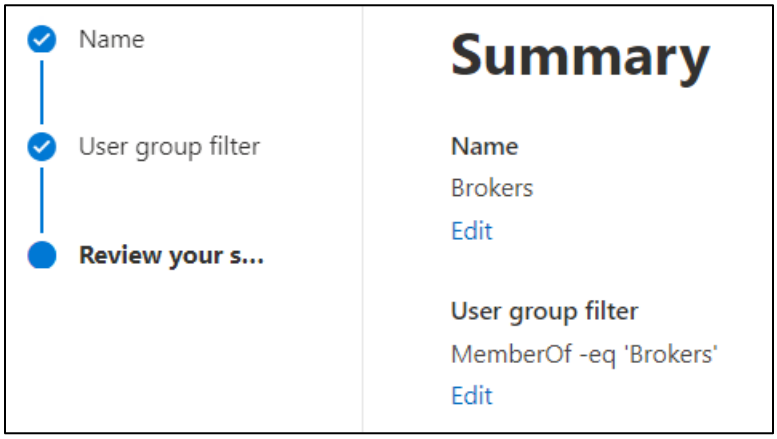
From here we can click on 'New Segment' and create a new Segment to be used by an Information Barrier Policy.



Provide a name and click Next.



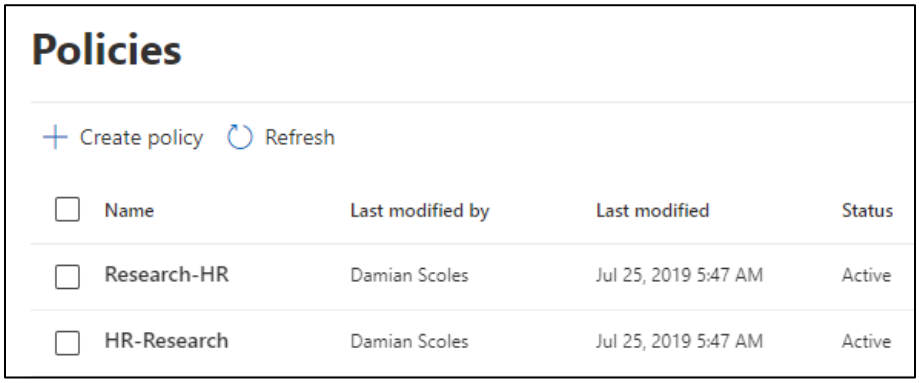
Add membership rules, making sure that the desired objects exist already.



Make sure to have a pre-existing group before completing this rule as an error will be generated and the same error will generate if the specified match is not a Proxy Address for the same group. With a Segment in place, we can now create a policy to apply to users in the Tenant.

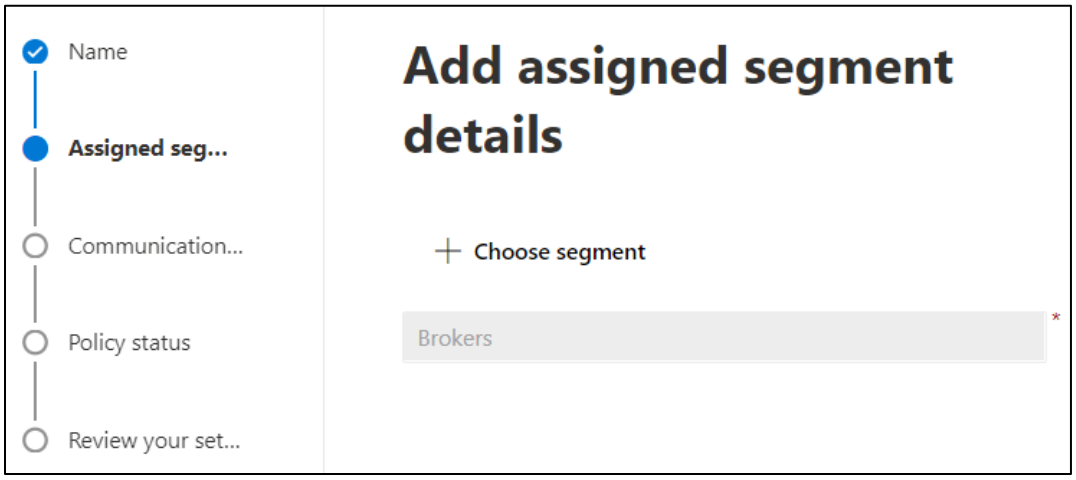
Information Barrier Policies

Another tab in Information Barriers show the policies that apply to users:

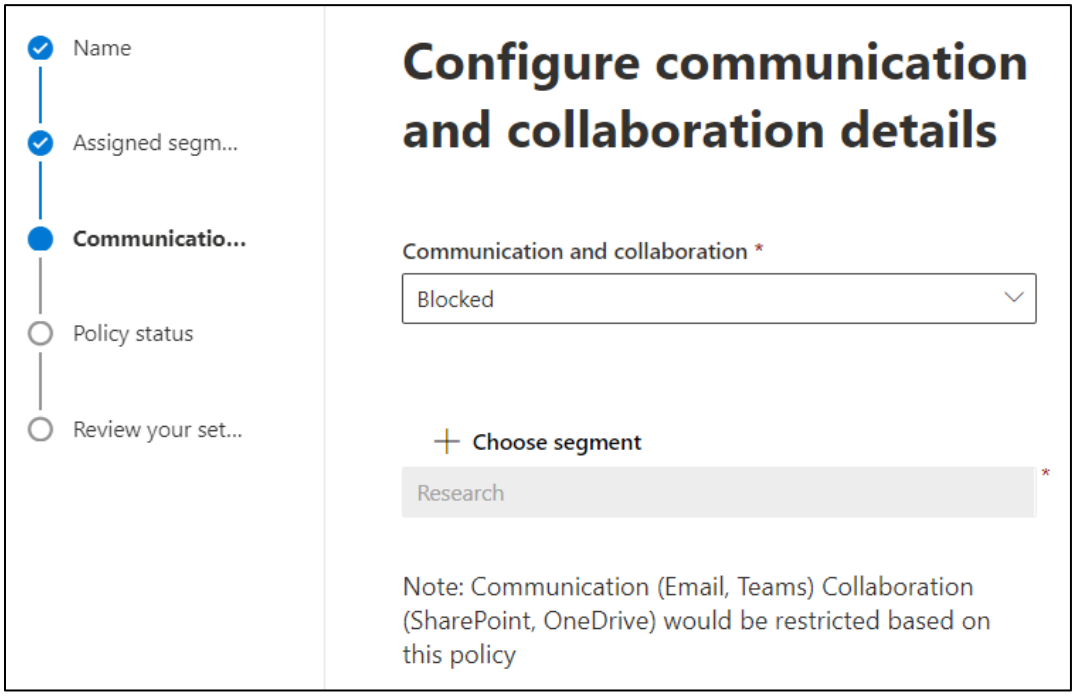


Clicking on 'Create Policy:

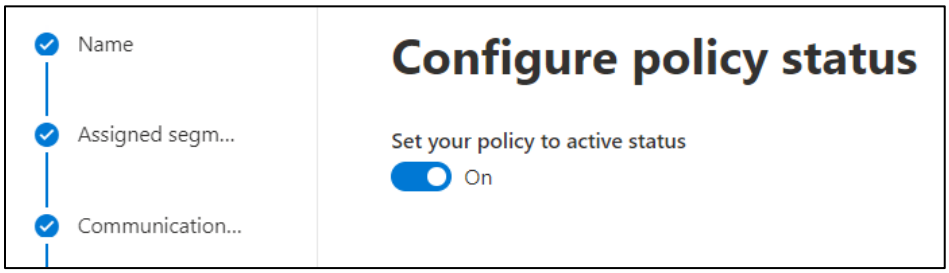
Add the previously created Segment:



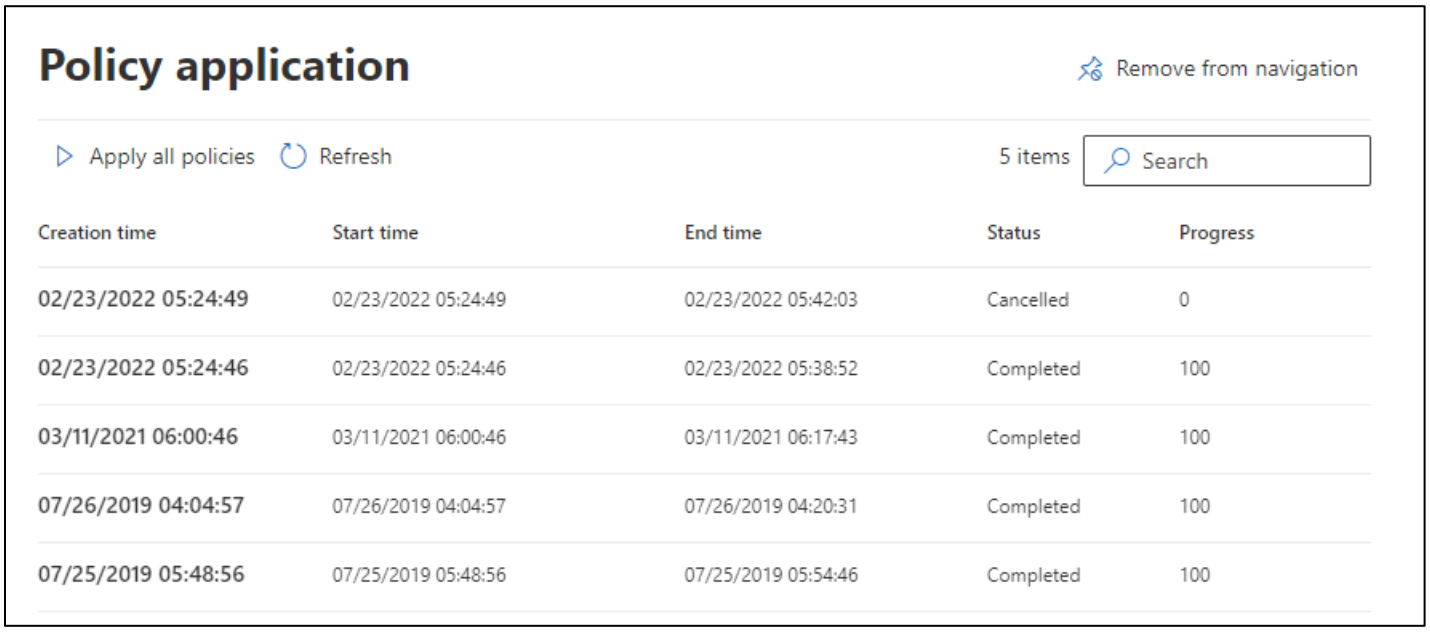
Pick an action to either allow or block communications:



Turn the Policy on:



Then click 'Next' and 'Submit' and your new Information Barrier Policy is in place. For this scenario, HR and Brokers would have restrictive communication with each other over Email, Teams, SharePoint and One Drive.



Lastly, we have a view of our Information barrier policies and their application in a tenant. Currently this appears to be more of an activity tracker and perhaps more detail and functionality will be revealed over time.

PowerShell

Using the PowerShell examples here should help guide you when implementing Information Barriers in your tenant. Another note, we are limited at this time to this subset of restrictions in Office 365:

Restrictions in Teams

As mentioned earlier, Information Barriers provides limited functionality as of the writing of this book. At this time, the current list of what is restricted is listed below:

Searching for a user
Starting a chat session with someone
Inviting someone to join a meeting
Placing a call
Access to file through sharing link
Adding a member to a team
Starting a group chat
Sharing a screen
Sharing a file with another user

A complete list of behaviors can be found at this Microsoft link, as they could change with little or no notice.

Prerequisites

Before we work with any Information Barrier PowerShell, we need to satisfy a few requirements. Most of these requirements are easy to meet, while one may not be as easy to accomplish. Below is the official list from Microsoft on Information Barriers:

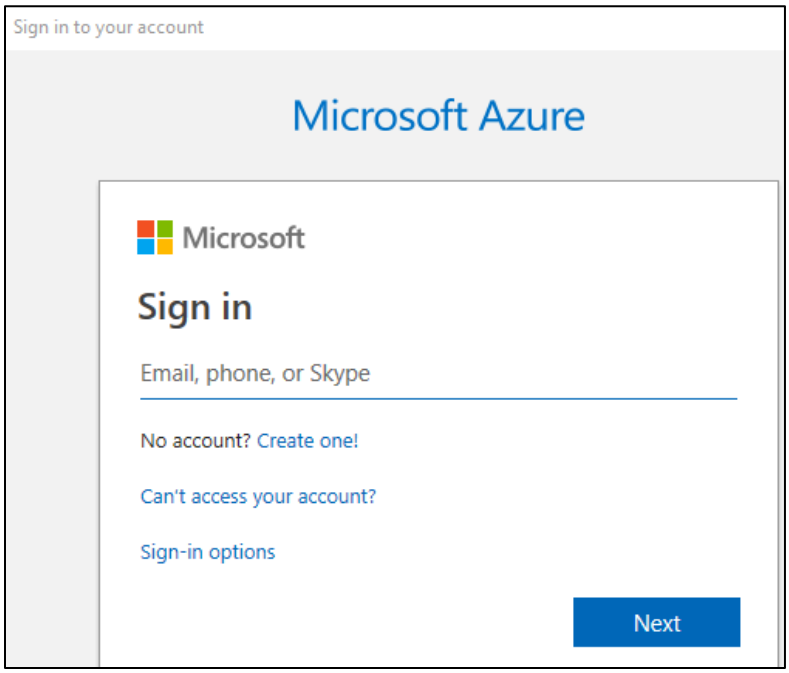
AD Replicated - make sure any attribute you want to filter by is replicating correctly to your tenant
Scoped Directory Search - https://docs.microsoft.com/en-us/MicrosoftTeams/teams-scoped-directory-search
Audit Logging On - https://docs.microsoft.com/en-us/office365/securitycompliance/turn-audit-log-search-on-or-off
No Exchange Online Address Book Polices - Remove prior to enabling Information Barriers (**)
PowerShell - Security and Compliance Center and Azure PowerShell modules
Administrative Consent - Follow a documented Microsoft process to enable access

(**) Note: This is an interesting requirement. For most organizations, the use of Address Book Policies is zero. However, in larger or complex environments, Address Book Polices are quite useful. If your organization has these in place now, planning will be needed prior to removing these to implement Information Barriers.

Administrative Consent

Administrative Consent grants certain access rights to your tenant for the Information Barrier Processor App and is applied globally to a tenant. For this task, we need to run some cmdlets in Azure (AZ PowerShell module required) in order to enable the use of Information Barriers:

```
Login-AzAccount
```




```
Account           : damian@practicalpowershell.com
SubscriptionName  : Microsoft Azure
SubscriptionId    : [redacted]
TenantId          : [redacted]
Environment       : AzureCloud
```

Type this into the PowerShell window (the ID represents the Information Barrier Processor App ID):

$AppId = "bcf62038-e005-436d-b970-2a472f8c1982"

```
PS C:\> $appId="bcf62038-e005-436d-b970-2a472f8c1982"
```

$SP = Get-AzureRmADServicePrincipal -ServicePrincipalName $AppId

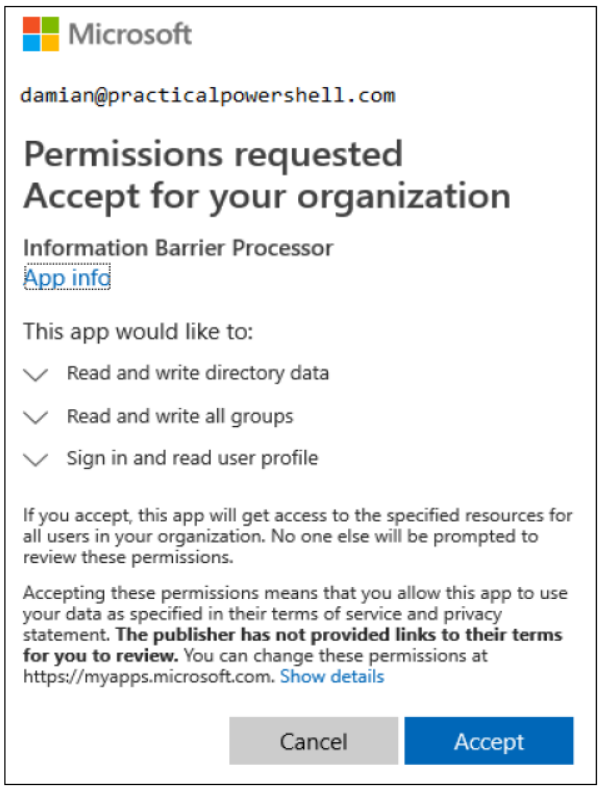
```
PS C:\> $sp=Get-AzureRmADServicePrincipal -ServicePrincipalName $appId
```

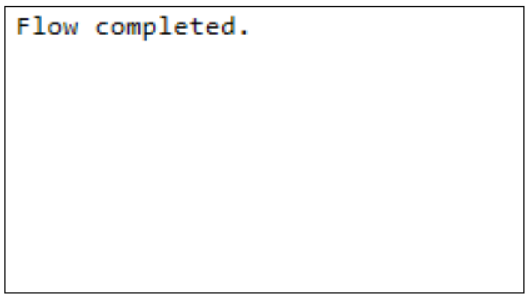
If ($SP -eq $Null) { New-AzureRmADServicePrincipal -ApplicationId $AppId }

```
PS C:\> if ($sp -eq $null) { New-AzureRmADServicePrincipal -ApplicationId $appId }


ServicePrincipalNames : {bcf62038-e005-436d-b970-2a472f8c1982, https://policyprocessor.microsoft.com}
ApplicationId         : bcf62038-e005-436d-b970-2a472f8c1982
DisplayName           : Information Barrier Processor
Id                    : eccbb8b9-45a0-40cf-8b30-bc388c102c08
AdfsId                :
Type                  : ServicePrincipal
```

```
Start-Process https://login.microsoftonline.com/common/adminconsent?client_id=$appId
```





Note: You may need to re-authenticate for the Permissions request step above.

Make sure to close any open PowerShell windows, otherwise an error will be thrown when working with the Information Barrier PowerShell cmdlets:

```
PS C:\> New-InformationBarrierPolicy -Name 'Research-HR' -AssignedSegment Research -SegmentsBlocked HR
Creating a new session for implicit remoting of "New-InformationBarrierPolicy" command...
New-PSSession : [nam05b.ps.compliance.protection.outlook.com] Connecting to remote server
nam05b.ps.compliance.protection.outlook.com failed with the following error message : Access is denied. For more
information, see the about_Remote_Troubleshooting Help topic.
At C:\Users\DScoles\AppData\Local\Temp\tmp_wzt1ndzv.qbw\tmp_wzt1ndzv.qbw.psm1:137 char:17
+                & $script:NewPSSession `
+                ~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OpenError: (System.Manageme....RemoteRunspace:RemoteRunspace) [New-PSSession], PSRemotin
   gTransportException
    + FullyQualifiedErrorId : AccessDenied,PSSessionOpenFailed
Exception calling "GetSteppablePipeline" with "1" argument(s): "No session has been associated with this implicit
remoting module."
At C:\Users\DScoles\AppData\Local\Temp\tmp_wzt1ndzv.qbw\tmp_wzt1ndzv.qbw.psm1:12522 char:13
+            $steppablePipeline = $scriptCmd.GetSteppablePipeline($myI ...
+            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : NotSpecified: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : RuntimeException
```

PowerShell

Before proceeding, close your PowerShell session to the SCC and reconnect to work with the new changes. Now that we have the basic requirements and background information on Information Barriers out of the way, let's take a look at what we can do in PowerShell. First, we need a list of cmdlets that contain the noun 'Barrier' in them. We'll run this cmdlet:

```
Get-Command *barrier*
```

This provides a list of cmdlets related to the new Information Barrier (cmdlets displayed may vary depending on permissions granted):

```
Get-InformationBarrierPoliciesApplicationStatus    Get-InformationBarrierPolicy
Get-InformationBarrierRecipientStatus              Get-InformationBarrierReportDetails
Get-InformationBarrierReportSummary                New-InformationBarrierPolicy
Remove-InformationBarrierPolicy                    Set-InformationBarrierPolicy
Start-InformationBarrierPoliciesApplication        Stop-InformationBarrierPoliciesApplication
Test-InformationBarrierPolicy
```

In addition to the above cmdlets, there is another set of cmdlets that are used to create the segmentation that is used by Information Barriers to create the logical separation. These are cmdlets used to create these Organization Segments:

```
Get-Command *segment
```

Which provides us with these cmdlets:

```
Get-OrganizationSegment          New-OrganizationSegment
Remove-OrganizationSegment       Set-OrganizationSegment
```

By default there are no Information Barriers or Organization Segments in your tenant, so we should create these first. For the New-OrganizationSegment cmdlet, let's review the Get-Help and look at the examples provided for the cmdlet. What we find is that because the cmdlet is so new, there are no examples listed in the Get-Help for the cmdlet. Microsoft Docs also does not have anything online for the cmdlet. So where can we find an example of how to use the cmdlet? We can look here: https://docs.microsoft.com/en-us/office365/securitycompliance/information-barriers-policies

Using the examples provided by this link, we can construct a one-liner to create our Organization Segments:

```
New-OrganizationSegment -Name 'Research' -UserGroupFilter "Department -eq 'Research'"
```

```
RunspaceId          : 15c6b834-4383-405c-8173-0736f130894f
Type                : OrganizationSegment
UserGroupFilter     : Department -eq 'Research'
ExoSegmentId        : 39ee3d20-47d5-4bc7-9479-e95534f86fd3
ObjectVersion       : 349447f7-7de4-4fc5-f5a1-08d712126e74
CreatedBy           : Damian Scoles
LastModifiedBy      : Damian Scoles
Comment             :
ModificationTimeUtc : 7/26/2019 9:44:26 PM
CreationTimeUtc     : 7/25/2019 5:18:42 AM
Identity            : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com/Configuration/Research
Id                  : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com/Configuration/Research
ExchangeVersion     : 0.20 (15.0.0.0)
Name                : Research
DistinguishedName   : CN=Research,CN=Configuration,CN=[redacted].onmicrosoft.com,OU=Microsoft Exchange Hosted Organizations,DC=FFO,DC=extest,DC=microsoft,DC=com
ObjectCategory      :
ObjectClass         : {msExchUnifiedPolicy}
WhenChanged         : 7/26/2019 4:44:26 PM
WhenCreated         : 7/25/2019 12:18:42 AM
WhenChangedUTC      : 7/26/2019 9:44:26 PM
WhenCreatedUTC      : 7/25/2019 5:18:42 AM
ExchangeObjectId    : 9eae29a4-4ce1-47c1-9eb7-031c0e46e717
OrganizationId      : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com - FFO.extest.microsoft.com/Microsoft Exchange Hosted
                      Organizations/[redacted].onmicrosoft.com/Configuration
Guid                : 9eae29a4-4ce1-47c1-9eb7-031c0e46e717
OriginatingServer   :
IsValid             : True
ObjectState         : Unchanged
```

For our example, we need to create one more Organization Segment otherwise we won't be able to create any Information Barriers:

```
New-OrganizationSegment -Name 'HR' -UserGroupFilter "Department -eq 'HR'"
```

```
RunspaceId          : 15c6b834-4383-405c-8173-0736f130894f
Type                : OrganizationSegment
UserGroupFilter     : Department -eq 'HR'
ExoSegmentId        : b2a7bd2b-9ede-4a4d-86b1-ede1ba25f962
ObjectVersion       : 81c5c9ff-61ea-4716-33ce-08d710bf7351
CreatedBy           : Damian Scoles
LastModifiedBy      : Damian Scoles
Comment             :
ModificationTimeUtc : 7/25/2019 5:17:55 AM
CreationTimeUtc     : 7/25/2019 5:17:55 AM
Identity            : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com/Configuration/HR
Id                  : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com/Configuration/HR
ExchangeVersion     : 0.20 (15.0.0.0)
Name                : HR
DistinguishedName   : CN=HR,CN=Configuration,CN=[redacted].onmicrosoft.com,OU=Microsoft Exchange Hosted Organizations,DC=FFO,DC=extest,DC=microsoft,DC=com
ObjectCategory      :
ObjectClass         : {msExchUnifiedPolicy}
WhenChanged         : 7/25/2019 12:17:55 AM
WhenCreated         : 7/25/2019 12:17:55 AM
WhenChangedUTC      : 7/25/2019 5:17:55 AM
WhenCreatedUTC      : 7/25/2019 5:17:55 AM
ExchangeObjectId    : 52af790c-f5bb-4a63-96be-0bd6bb6d3e29
OrganizationId      : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com - FFO.extest.microsoft.com/Microsoft Exchange Hosted
                      Organizations/[redacted].onmicrosoft.com/Configuration
Guid                : 52af790c-f5bb-4a63-96be-0bd6bb6d3e29
OriginatingServer   :
IsValid             : True
ObjectState         : New
```

Now that we have multiple organization segments to work with, we can build out some Information Barriers. The first cmdlet we need to use is 'New-InformationBarrierPolicy. Similar to the New-OrganizationSegment, there are no examples listed in the Get-Help of the cmdlet. However, there is a Microsoft Docs page for the cmdlet. We can bring up the Online version like so:

```
Get-Help New-InformationBarrierPolicy -Online
```

From the same page that listed examples for New-OrganizationSegment, we also find examples for the New-InformationBarrier cmdlet. So, following this, let's work through a sample scenario. We have two defined Organization Segments - HR and Research.

Per a corporate directive, we need to construct Information Barriers between these two segments. When we create the barriers, we also need to set the state to Inactive, per Microsoft best practices for Information Barrier creation. The theory on that is that we can create barriers, activate the Information Barrier app that runs in Azure and then apply them once we

are sure the policies are correct in place. So for creating the base barriers, we need these parameters - Name, AssignedSegment, SegmentBlocked and State. From this we will start with blocking the HR segment from the Research segment:

```
New-InformationBarrierPolicy -Name 'Research-HR' -AssignedSegment Research -SegmentsBlocked
 HR -State Inactive
```

```
RunspaceId           : 2e07d4cf-d7a4-4d5c-b3e5-3b2902cddc6e
Type                 : InformationBarrier
AssignedSegment      : Research
SegmentsAllowed      : {}
ExoPolicyId          : 69a9440c-4e3d-492d-8c48-d85297146576
SegmentsBlocked      : {HR}
SegmentAllowedFilter :
BlockVisibility      : True
BlockCommunication   : True
State                : Inactive
ObjectVersion        : 8c6df40e-e692-47cd-bfe5-08d710c2df48
CreatedBy            : Damian Scoles
LastModifiedBy       : Damian Scoles
Comment              :
ModificationTimeUtc  : 7/25/2019 5:42:25 AM
CreationTimeUtc      : 7/25/2019 5:42:25 AM
Identity             : FFO.extest.microsoft.com/Microsoft Exchange Hosted
                       Organizations/[redacted].onmicrosoft.com/Configuration/Research-HR
Id                   : FFO.extest.microsoft.com/Microsoft Exchange Hosted
                       Organizations/[redacted].onmicrosoft.com/Configuration/Research-HR
ExchangeVersion      : 0.20 (15.0.0.0)
Name                 : Research-HR
DistinguishedName    : CN=Research-HR,CN=Configuration,CN=[redacted].onmicrosoft.com,OU=Microsoft Exchange Hosted
                       Organizations,DC=FFO,DC=extest,DC=microsoft,DC=com
ObjectCategory       :
ObjectClass          : {msExchUnifiedPolicy}
WhenChanged          : 7/25/2019 12:42:25 AM
WhenCreated          : 7/25/2019 12:42:25 AM
WhenChangedUTC       : 7/25/2019 5:42:25 AM
WhenCreatedUTC       : 7/25/2019 5:42:25 AM
ExchangeObjectId     : 4809c4b8-e60d-4ffe-a7eb-d52254aa193f
OrganizationId       : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com -
                       FFO.extest.microsoft.com/Microsoft Exchange Hosted
                       Organizations/[redacted].onmicrosoft.com/Configuration
Guid                 : 4809c4b8-e60d-4ffe-a7eb-d52254aa193f
OriginatingServer    :
IsValid              : True
ObjectState          : New

WARNING: Your changes will take into affect after you run Start-InformationBarrierPoliciesApplication cmdlet.
Start-InformationBarrierPoliciesApplication cmdlet only applies Active state policies.
```

We then need to create a rule where we block Research from HR:

```
New-InformationBarrierPolicy -Name 'HR-Research' -AssignedSegment HR -SegmentsBlocked
 Research -State Inactive
```

```
RunspaceId           : 2e07d4cf-d7a4-4d5c-b3e5-3b2902cddc6e
Type                 : InformationBarrier
AssignedSegment      : HR
SegmentsAllowed      : {}
ExoPolicyId          : 1b4679d1-3abd-4ae7-8d7d-e86731222a45
SegmentsBlocked      : {Research}
SegmentAllowedFilter :
BlockVisibility      : True
BlockCommunication   : True
State                : Inactive
ObjectVersion        : b44f97ea-459f-442a-e3d7-08d710c32add
CreatedBy            : Damian Scoles
```

```
LastModifiedBy       : Damian Scoles
Comment              :
ModificationTimeUtc  : 7/25/2019 5:44:32 AM
CreationTimeUtc      : 7/25/2019 5:44:32 AM
Identity             : FFO.extest.microsoft.com/Microsoft Exchange Hosted
                       Organizations/[redacted].onmicrosoft.com/Configuration/HR-Research
Id                   : FFO.extest.microsoft.com/Microsoft Exchange Hosted
                       Organizations/[redacted].onmicrosoft.com/Configuration/HR-Research
ExchangeVersion      : 0.20 (15.0.0.0)
Name                 : HR-Research
DistinguishedName    : CN=HR-Research,CN=Configuration,CN=[redacted].onmicrosoft.com,OU=Microsoft Exchange Hosted
                       Organizations,DC=FFO,DC=extest,DC=microsoft,DC=com
ObjectCategory       :
ObjectClass          : {msExchUnifiedPolicy}
WhenChanged          : 7/25/2019 12:44:31 AM
WhenCreated          : 7/25/2019 12:44:31 AM
WhenChangedUTC       : 7/25/2019 5:44:31 AM
WhenCreatedUTC       : 7/25/2019 5:44:31 AM
ExchangeObjectId     : 0dae728c-72c0-4a54-926e-ef1500dcb3d9
OrganizationId       : FFO.extest.microsoft.com/Microsoft Exchange Hosted Organizations/[redacted].onmicrosoft.com
                       FFO.extest.microsoft.com/Microsoft Exchange Hosted
                       Organizations/[redacted].onmicrosoft.com/Configuration
Guid                 : 0dae728c-72c0-4a54-926e-ef1500dcb3d9
OriginatingServer    :
IsValid              : True
ObjectState          : New
```

With those two cmdlets we now have two Information Barriers ready to be used. Let's verify that we can see these in our tenant:

```
Get-InformationBarrierPolicy | Ft Name,Guid
```

```
PS C:\> Get-InformationBarrierPolicy | ft name,guid

Name        Guid
----        ----
Research-HR 4809c4b8-e60d-4ffe-a7eb-d52254aa193f
HR-Research 0dae728c-72c0-4a54-926e-ef1500dcb3d9
```

Once we are sure that all of our settings are good and that all of the barriers we want to create are done, we can change the state of all of our policies to active like so: (Using the GUID's from above):

Set-InformationBarrierPolicy -Identity 4809c4b8-e60d-4ffe-a7eb-d52254aa193f -State Active
Set-InformationBarrierPolicy -Identity 0dae728c-72c0-4a54-926e-ef1500dcb3d9 -State Active

Note: Make sure to use the GUIDs like the example above, otherwise piping Get-InformationBarrier to Set-InformationBarrier will fail.

```
PS C:\> Set-InformationBarrierPolicy -Identity 4809c4b8-e60d-4ffe-a7eb-d52254aa193f -State Active
WARNING: Your changes will take into affect after you run Start-InformationBarrierPoliciesApplication cmdlet.
Start-InformationBarrierPoliciesApplication cmdlet only applies Active state policies.
PS C:\> Set-InformationBarrierPolicy -Identity 0dae728c-72c0-4a54-926e-ef1500dcb3d9 -State Active
WARNING: Your changes will take into affect after you run Start-InformationBarrierPoliciesApplication cmdlet.
Start-InformationBarrierPoliciesApplication cmdlet only applies Active state policies.
```

To finalize the process we have one last cmdlet that needs to be run. This cmdlet will kick off the process where accounts will be segmented off:

```
Start-InformationBarrierPoliciesApplication
```

```
PS C:\> Start-InformationBarrierPoliciesApplication
WARNING: It may take several hours for the application to finish. Please check the status using
Get-InformationBarrierPoliciesApplicationStatus cmdlet. Execution of New/Set cmdlets will be prevented until start/stop
 is finished.


RunspaceId              : 2e07d4cf-d7a4-4d5c-b3e5-3b2902cddc6e
Identity                : 5ea0bd1f-29ab-4c36-8300-7f7d0745ba55
CreatedBy               : Damian Scoles
CancelledBy             :
Type                    : ExoApplyIBPolicyJob
ApplicationCreationTime : 07/25/2019 05:48:56
ApplicationEndTime      :
ApplicationStartTime    : 07/25/2019 05:48:56
TotalBatches            : 0
ProcessedBatches        : 0
PercentProgress         : 0
TotalRecipients         : 0
SuccessfulRecipients    : 0
FailedRecipients        : 0
FailureCategory         : None
Status                  : NotStarted
IsValid                 : True
ObjectState             : Unchanged
```

Microsoft notes that this process does take time to touch each user object. The process itself takes up to 30 minutes to start and can process up to 5,000 users an hour. For larger organizations this could mean up to and past a 24 hour delay before all Information Barrier policies are applied.

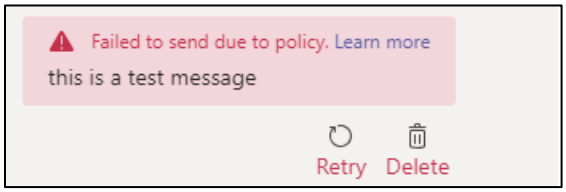
If we want to validate the progress of the Application, we get use 'Get-InformationBarrierPoliciesApplicationStatus' to do so:

```
RunspaceId              : 2e07d4cf-d7a4-4d5c-b3e5-3b2902cddc6e
Identity                : 5ea0bd1f-29ab-4c36-8300-7f7d0745ba55
CreatedBy               : Damian Scoles
CancelledBy             :
Type                    : ExoApplyIBPolicyJob
ApplicationCreationTime : 07/25/2019 05:48:56
ApplicationEndTime      :
ApplicationStartTime    : 07/25/2019 05:48:56
TotalBatches            : 0
ProcessedBatches        : 0
PercentProgress         : 0
TotalRecipients         : 0
SuccessfulRecipients    : 0
FailedRecipients        : 0
FailureCategory         : None
Status                  : NotStarted
IsValid                 : True
ObjectState             : Unchanged
```

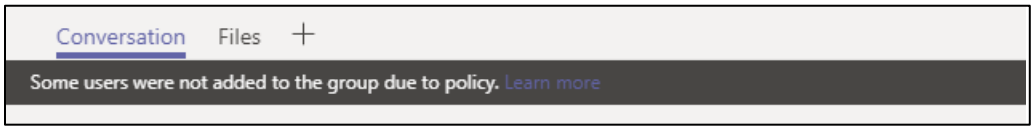
Real World Experience

In our scenario we walked through with PowerShell in the past few pages, we have multiple users assigned a barrier policy, two Organization Segments defined and policies that block communication between the two. How can we validate the settings work, outside of PowerShell? Well, we can have the users log into their Teams clients and see what communications are allowed or blocked.

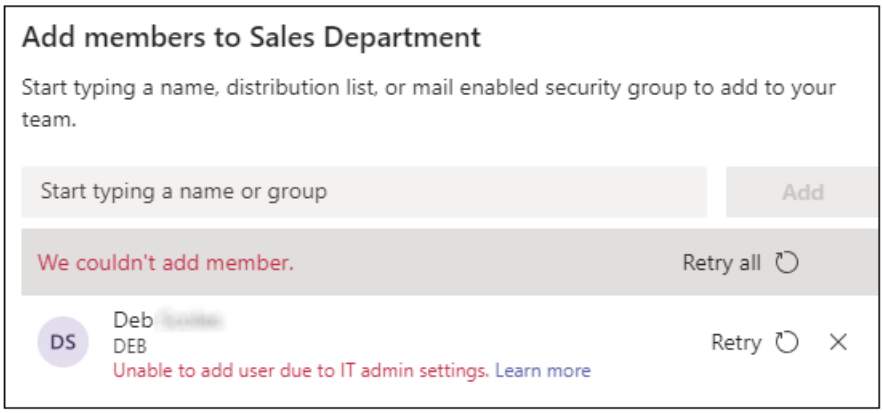
For the first test, we have a user in one segment who is trying to IM a user in the other segment, however, we get this message from either user that tries to IM the other in their Teams client:



If we have an existing conversation with a user that is in the same segment, then we try to add another user in another segment, we are blocked from doing so:



Last option to try is to add a user, in a different segment, to a Team to collaborate with. However, as we should know, this will fail:



Caveats to Blocking

Information Barriers are not perfect or even complete in their attempt to block communications between teammates or co-workers. Information Barriers are restricted to Teams only and only parts of Teams. We still have Exchange Online that can be shared between people. In order to handle these workloads, we could set up alerts and DLP to monitor and/or restrict sharing of information.

Documenting Settings (Script)

Now that we have our Information Barriers in place and we have users segmented in Office 365, how do we know this is working? There are a series of 'Get' PowerShell cmdlets that have to deal with Organization Segments and Information Barriers. We will utilize these for our documentation.

PowerShell

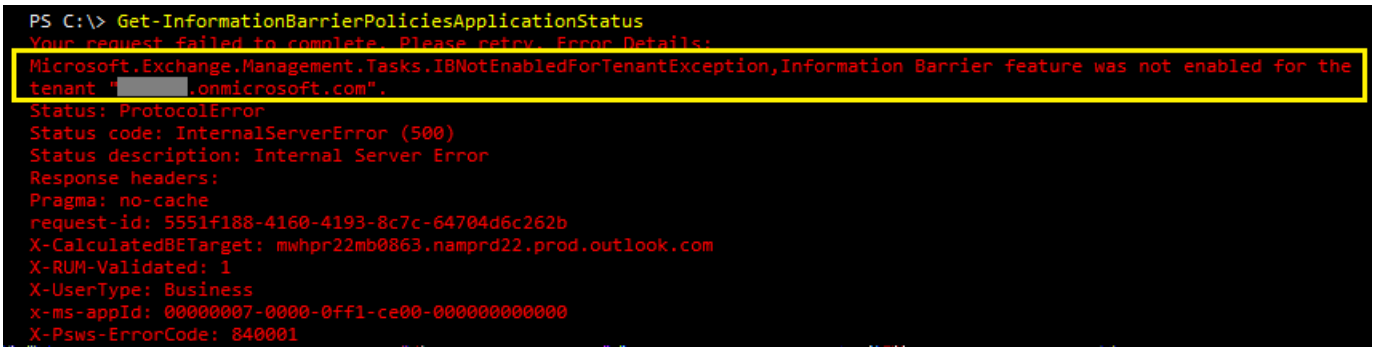
Audit logs are great for post-change monitoring and reporting for management and IT security to keep an eye on the Information Barrier changes. We can again use them for compliance reasons. Beyond auditing we may need to confirm how the Information Barrier is configured and produce this as well as the actual audit report for compliance. When it comes to documenting the settings in Information Barriers, we need to think in terms of what it takes to configure the barrier, policies, segments and apply them to users. Since we have no other interface with which to check and document these settings, we'll just use PowerShell to do this for us.

Where to start?

For an overall status check on the Information Barrier feature in the Security and Compliance Center, we should run the 'Get-InformationBarrierPoliciesApplicationStatus' cmdlet in order to validate that the Barrier Application has been started with the 'Start-InformationBarrierPoliciesApplication' cmdlet. If the 'Start-InformationBarrierPoliciesApplication' cmdlet has not been run, we may run into errors when running this cmdlet:

```
Get-InformationBarrierPoliciesApplicationStatus
```

Typical errors when configuration has not begun:



Because we may get this result, we need to accommodate for this possibility. The easiest way is to use a Try and Catch {} code block. If the cmdlet fails, we can stop documenting because no configuration is probably available.

Code Sample

```
$InfoBarrier = $True
Try {
$Test = Get-InformationBarrierPoliciesApplicationStatus -ErrorAction STOP
} Catch {
$InfoBarrier = $False
}
```

Notice that there is a variable called $InfoBarrier which is set to True at the start of the block. This is set to True because we assume the configuration is in place. If, however, the cmdlet fails, triggering the STOP and then running the code in the Catch section, the variable is then set to False. This means no configuration is running and we can skip checking settings. If however, the $InfoBarrier variable is still True, we can proceed to the next section of code. In order to properly document the findings, we will need some sort of output file that will be used by the script. We can define the output file in a variable block in the code like so:

```
$Path = (Get-Item -Path ".\" -Verbose).FullName
$File = "InformationBarrierDocumentation.Txt"
$Destination = $Path+"\"+$File
```

With the above lines, we tell PowerShell to get the current path, combine it with a predefined file name which together will become the destination file path for the script to export its findings to. Sample destination:

```
D:\Scripts\SCC\InformationBarrier\InformationBarrierDocumentation.Txt
```

Now that we have this, we can move on to documenting. In the next block, we'll begin by writing to the documentation file with a description line (Line 1), a separator line (Line 2) and a blank line for formatting (Line 3). Following this we'll run the Get-InformationBarrierPoliciesApplicationStatus in its default format-list format. The code is here:

```
$Line = 'Information Barrier Policy Application Status' | Out-File $Destination -Append
$Line = '-----------------------------------------------' | Out-File $Destination -Append
$Line = ' ' | Out-File $Destination -Append
$Line = Get-InformationBarrierPoliciesApplicationStatus | Out-File $Destination -Append
```

```
$Line = ‘ ‘ | Out-File $Destination -Append
```

The output from these lines of code will look something like this:

```
Information Barrier Policy Application Status
----------------------------------------------


RunspaceId              : b9aab4f8-b6de-4188-8bd5-52d7a5c9da88
Identity                : 5ea0bd1f-29ab-4c36-8300-7f7d0745ba55
CreatedBy               : Damian Scoles
CancelledBy             :
Type                    : ExoApplyIBPolicyJob
ApplicationCreationTime : 07/25/2019 05:48:56
ApplicationEndTime      : 07/25/2019 05:54:46
ApplicationStartTime    : 07/25/2019 05:48:56
TotalBatches            : 1
ProcessedBatches        : 1
PercentProgress         : 100
TotalRecipients         : 512
SuccessfulRecipients    : 512
FailedRecipients        : 0
FailureCategory         : None
Status                  : Completed
IsValid                 : True
ObjectState             : Unchanged
```

Now that we have the application documented, we can now pull information on each Information Barrier Policy. For consistency, we’ll use a similar block of code, with a descriptive line, a line of ‘-’ for separation and a blank line for formatting. Then, we’ll use ‘Get-InformationBarrierPolicy ‘ cmdlet, which by default will produce output in a list format, which may not be as useful for a summary table. For this cmdlet, we’ll look at that output, pick which fields we want and then run a one-liner that will produce a table of values per policy:

```
$Line = ‘Information Barrier Policies’ | Out-File $Destination -Append
$Line = ‘----------------------------’ | Out-File $Destination -Append
$Line = ‘ ‘ | Out-File $Destination -Append
$Line = Get-InformationBarrierPolicy | ft Name, Type, AssignedSegment, SegmentsAllowed,
 SegmentsBlocked, SegmentsAllowedFilter, BlockVisibility, BlockCommunication, State,
 CreatedBy, CreationTimeUTC | Out-File $Destination -Append
```

The output from this code block is as follows:

```
Information Barrier Policies
----------------------------

Name        Type               AssignedSegment SegmentsAllowed SegmentsBlocked SegmentsAllowedFilter BlockVisibility BlockCommunication State  CreatedBy     CreationTimeUtc
----        ----               --------------- --------------- --------------- -------------------- --------------- ------------------ -----  ---------     ---------------
Research-HR InformationBarrier Research        {}              {HR}                                 True              True Active Damian Scoles 7/25/2019 5:42:25 AM
HR-Research InformationBarrier HR              {}              {Research}                           True              True Active Damian Scoles 7/25/2019 5:44:32 AM
```

The above is a concise summary table of our barrier policies. Some fields are necessarily empty because there is nothing in them. However, if the information were added later, we would have columns ready to populate. In a larger, more complex environment this would cover the cross-segment filter that could occur.

Next in the list items to document are the Organization Segments. It’s one of the cmdlets / sections that does not specifically mention that it is a part of Information Barriers. Similar to the other cmdlets, Get-OrganizationSegment also outputs in a list

format. As such, we will pick some relevant fields and then use a table format to export the results of the cmdlet to the $Destination file. Also, same formatting before as the other two code blocks for consistency:

```
$Line = ‘ ‘ | Out-File $Destination -Append
$Line = ‘Organization Segment(s)’ | Out-File $Destination -Append
$Line = ‘----------------------------’ | Out-File $Destination -Append
$Line = Get-OrganizationSegment | Ft Name, Type, UserGroupFilter, ObjectClass, CreatedBy
 | Out-File $Destination -Append
```

Output from the script looks like this:

```
Organization Segment(s)
---------------------------

Name     Type                UserGroupFilter           ObjectClass           CreatedBy
----     ----                ---------------           -----------           ---------
Research OrganizationSegment Department -eq 'Research' {msExchUnifiedPolicy} Damian Scoles
HR       OrganizationSegment Department -eq 'HR'       {msExchUnifiedPolicy} Damian Scoles
```

What about users in your tenant? How do we check the users to see if they are getting the policy applied? This turns out to be possible, if a bit harder when it comes to actually creating a report for the users. First, we need our description and formatting for the $Destination file, following the same format as all other sections:

```
$Line = ‘ ‘ | Out-File $Destination -Append
$Line = ‘Information Barrier Recipient Status’ | Out-File $Destination -Append
$Line = ‘-----------------------------------’ | Out-File $Destination -Append
```

Next, we’ll need to set some variables for this section of code. The first will be a variable to store information about all of the Organization Segments. The next two will be arrays (with the ‘@()’) which will be used to store more complex information:

```
$OrgSegments = Get-OrganizationSegment
$AllOrgFilters = @()
$AllRecipients = @()
```

Next, we will go through each Organization Segment and store just the User Group Filter property of each Organization Segment. This will allow us to search for users with this criterion. We will use the ‘$AllOrgFilters’ array that we defined above.

```
Foreach ($OrgSegment in $OrgSegments) {
$OrgFilter = $OrgSegment.UserGroupFilter
$AllOrgFilters += $OrgFilter
}
```

Next, we will add a line to our output line to help identify the columns that will present in the output that is generated from the user queries:

```
$Line = “Segment,Alias,Department,DisplayName,ExOPolicyID,IsValid” | Out-File $Destination
 -Append
```

Next, we will loop through each Organization Segment, which we will use to pull information from which will be used to find users:

```
    Foreach ($OrgFilter in $AllOrgFilters) {
```

Simple, single attribute filter: (Pulled from the $OrgFilter variable and stored in the array)

```
UserGroupFilter     : Department -eq 'Research'
```

The above is the easiest to deal with and the easiest to code for. However, what if there were two attributes:

```
UserGroupFilter     : Department -eq 'Research' -and PostalCode -eq '60606'
```

Harder to deal with, because now we have more than one set of criteria. If the filter gets more complete though, we will need to code some sort of logic to handle that. Now, there is a caveat to doing this work. You may not be able to cover all scenarios. The code below is a sample code, for you, the reader, to use as a base for building more complex scripts. So let's take the UserGroupFilter and see what we can do handle a more complex filter.

First, we will take the filter above with the '-and' in it. We can use this as our basis to begin building. Now when we run the script and we are working with the filter, it is stored in the $OrgFilter (from the previous Foreach loop beginning). Let's use the example where $OrgFilter = "Department -eq 'Research' -and PostalCode -eq '60606'". First, we need to split this into two values. Typically in PowerShell we would use a couple of options for this, a '-Split' parameter or we could use the '.Split' function of the variable itself. So let's do that first:

```
    $OrgFilterValues = $Orgfilter -split ' -and '
```

Notice the trailing and preceding spaces in the split string. We do this to remove all spaces that are not needed. Now we have these values:

```
PS C:\> $TestSplit[0]
Department -eq 'Research'

PS C:\> $TestSplit[1]
PostalCode -eq '60606'
```

Next, we need to loop through each of the Org Filters we have stored in the $OrgFilterValues variable:

```
    Foreach ($OrgFilterValue in $OrgFilterValues) {
```

First, we'll remove all of the spaces that are present to prevent any mismatches later:

```
    $SplitOrgFilterValues = $OrgFilterValue.Split(' ')
```

Within this loop we can now pull out the three criteria needed for finding users that are to be involved in Information Barriers. These criteria are Attribute, Operator and the Value to be found:

Attribute - what user attribute we are looking to match [The script defines this in the $Attribute variable]
Operator - are we looking for an equal or not equal operator [The script defines this in the $Operator variable]
Value - what we are looking for in the attribute [The script defines this in the $Value variable]

Then we will store the three items in separate variables:

```
    $Attribute = $SplitOrgFilterValues[0]
```

```
$Operator = $SplitOrgFilterValues[1]
$Value = $SplitOrgFilterValues[2] -replace (“’”,””)
```

To make this script more efficient, we will implement a function to handle each name query in the script. The reason we are using a function is that this will be a good-sized repeatable set of code. By doing so we can eliminate duplicating code sections and keep the number of code lines down making the script smaller and more efficient. Let’s review what variables or values we need for the names query.

Since we are using a function, we will want to pass these variables to the function. Let’s first decide on a function name. When we name it, we should name it something that is recognizable. For this exercise, let’s use ‘GetNames’ for our function name. So in order to pass the variables, we need a line like so:

```
GetNames $Attribute $Operator $Value
```

We are passing these in order, just to make it easy on ourselves. It is NOT required for the first example, but would be for the second Function example. Now, the Function will need to know how to handle it. So we have a couple of choices, both will work:

```
Function GetNames ($Attribute,$Operator,$Value) {
}
```

Alternatively, we can use a different method to perform the same task:

```
Function GetNames {
Param(
[parameter(position=0)]
$Attribute,
[parameter(position=1)]
$Operator,
[parameter(position=2)]
$Value
)
}
```

For our code sample we will use the first option as it requires fewer lines of code:

```
Function GetNames ($Attribute,$Operator,$Value ) {
```

Next, we will define our variables. First one will be used for a Try and Catch later for error handling in case a query fails:

```
$NamesQuery = $True
```

Then we search for users with the ‘Get-User’ cmdlet. We pull all three parameters into this. On the outside we have two If statements, one for if the $Operator is ‘-eq’ and the other if the $Operator is ‘-ne’. Then inside the If statement, we perform a Try and Catch on the Get-Name filter using the $Attribute, operator and $Value parameters pulled in from the function. If the query is successful, all names are stored in the $Names variable. If it fails, the $NamesQuery variable is changed to $False.

```
If ($Operator -eq ‘-eq’) {
Try {
$Names = (Get-User -ErrorAction STOP | where {$_.($Attribute) -eq $Value}).DisplayName
```

```
} Catch {
$NamesQuery = $False
}
}
If ($Operator -eq '-ne') {
Try {
$Names = (Get-User -ErrorAction STOP | where {$_.($Attribute) -ne $Value}).DisplayName
} Catch {
$NamesQuery = $False
}
}
```

Now that we have a list of names, we can prepare them for output to a chart. First, we check the $NamesQuery variable. If it is set to $False, we skip this loop as we have no users to work with. If, however, the $NamesQuery variable is still $True, this means that we have a list of users to work with.

```
If ($NamesQuery) {
```

Assuming we were able to query names in the environment, we now need to loop through the $Names variable with a Foreach loop:

```
Foreach ($Name in $Names) {
```

First, we pull the Information Policy of the user and store it in a $RecipientStatus variable:

```
$RecipientStatus = Get-InformationBarrierRecipientStatus $Name | select Alias, $Attribute,
 DisplayName, ExoPolicyID, IsValid
```

Next, we retrieve the ExOPolicyID value for the Policy. This requires some manipulation as the value has details we don't need. The value we need is at the end of a string of values, separated by '/' characters. We can use '.Split' to get this:

```
$ExOPolicyIDTemp = ($RecipientStatus).ExoPolicyID
$ExOPolicyID = $ExOPolicyIDTemp.split('/')[1]
```

With the ExoPolicyID, we can pull the AssignedSegment value:

```
$Segment = (Get-InformationBarrierPolicy -ExoPolicyId $ExOPolicyID).AssignedSegment
```

Now we grab information on the recipient - Alias, Department, Display Name and if it is valid:

```
$Alias = $RecipientStatus.Alias
$Department = $RecipientStatus.Department
$DisplayName = $RecipientStatus.DisplayName
$IsValid = $RecipientStatus.IsValid
```

Now we take this information, place it into one row and use 'Out-File' to export the data to a file:

```
# Export results:
$Line = "$Segment,$Alias,$Department,$DisplayName,$ExOPolicyID,$IsValid" | Out-File
 $Destination -Append
```

Output from this section looks like this:

```
Information Barrier Recipient Status
------------------------------------
Segment,Alias,Department,DisplayName,ExOPolicyID,IsValid
Research,Damian,Research,Damian Scoles,69a9440c-4e3d-492d-8c48-d85297146576,True
Research,John,Research,John Smith,69a9440c-4e3d-492d-8c48-d85297146576,True
```

Conclusion

Information Barriers can be an important aspect of separation of duties in Microsoft 365. While not all workloads are covered, Teams, Email, SharePoint and OneDrive communications / collaboration will be restricted with this feature. Microsoft affords administrators two separate methods for configuration which will make it easier for an organization's implementation. In this chapter we covered both PowerShell and the GUI methodologies for managing the Information Barrier feature. In the next chapter we will cover Insider Risk Management.