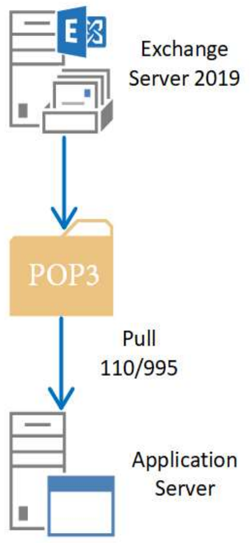
- 10 -
IMAP and POP

POP

POP3 is still commonly used by ISP's (Internet Service Providers) as a means for mail clients to retrieve mail from a mail server. POP3 utilizes port 110 (unencrypted, not secure or opportunistic TLS) or port 995 (secure, encrypted) for connecting to a server, so the client could potentially use either a secure or an insecure connection when retrieving emails. Generally speaking, Outlook users with an Exchange mailbox do not use POP3 nor is it recommended. MAPI, RPC over HTTP or MAPI over HTTP are the preferred methods to access a mailbox. POP3 is either used by older clients, clients on Operating Systems other than Windows or by applications that retrieve mail for a special purpose.



POP3 was created quite a long time ago and is beginning to show its age in the form of negative. The perception is that the number of apps or clients that connect using this method is declining. Here is a short list of limitations:

1. Download and Delete: POP3 clients initiate a download of messages from the server. Once downloaded the emails are removed from the server that hosts the emails. While some ISP's and POP3 providers have put in delays for when messages are removed, the messages will be removed and the only copy will be the local copy.
2. Limited client connectivity as only one device can download the emails.

For the purposes of scripting, we may not care about these deficiencies. Most Exchange engineers work with POP3 in three scenarios (1) setting up the initial configuration which means deciding on a login method, security (or no security) and then testing with the application needing the connection, (2) troubleshooting an existing POP3 setup or (3) checking for POP3 connections to see if that service needs to be moved to the new servers a client is deploying.

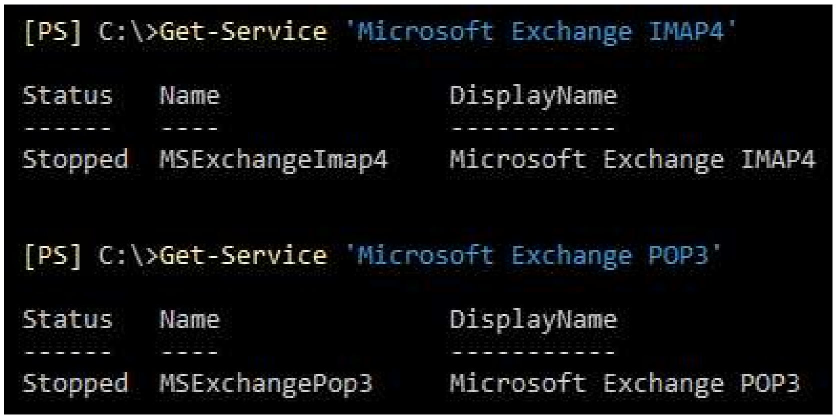
The following scenario we will cover the third scenario and build a script to look at the POP3 logs. By examining the POP3 logs of an Exchange server, we can see if the service is being used and even report back what servers or clients are using POP3 to pull email from the Exchange server.

Discovering POP3 Connections

Scenario - Migrating email from Exchange 2013 to Exchange 2019 (based on real world experiences)
Goal - Discover if apps are using POP3
Why? - Make sure all services are moved to 2019

How to accomplish the goal?
For our scenario we need to discover what applications, servers or computers are attempting to connect to Exchange 2013 so we can replicate these connections in Exchange 2019. First we need to start out with what we know about Exchange 2013's POP3 and IMAP feature. We know there are two services that Exchange uses to allow client connections. First one is called 'Microsoft Exchange POP3' and the second service is called 'Microsoft Exchange IMAP4'. We need to first check to see if these services are running on the server. We can do this with the Services.msc MMC console or we can do it with PowerShell.

Discovering properties on these services is easy with PowerShell:



```
[PS] C:\>Get-Service 'Microsoft Exchange IMAP4'

Status   Name               DisplayName
------   ----               -----------
Stopped  MSExchangeImap4    Microsoft Exchange IMAP4


[PS] C:\>Get-Service 'Microsoft Exchange POP3'

Status   Name               DisplayName
------   ----               -----------
Stopped  MSExchangePop3     Microsoft Exchange POP3
```

How do we know what PowerShell cmdlet to use? Using what was discussed previously in the book, we can run:

```
        Get-Command *Service*
```

Note: The use of the '*' wildcard symbol which confers any amount and type of characters before the word 'service'.

This will give us a list of PowerShell cmdlets with the word 'service' in it. Here is the list of cmdlets we can use:

```
        Get-Service         New-Service         Restart-Service
        Resume-Service      Set-Service         Start-Service
        Stop-Service        Suspend-Service
```

We can further improve our cmdlet search by just looking for GET cmdlets as these are informational:

Get-Command Get-*Service*

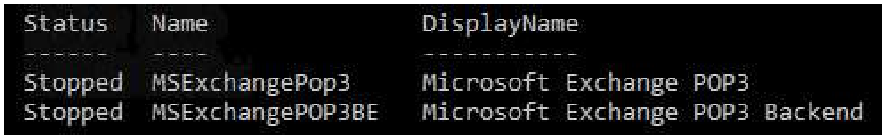
This gives us GET cmdlets with the word 'service' in it:

```
        Get-ClientAccessService             Get-FrontendTransportService
        Get-MailboxTransportService         Get-SMServerService
        Get-TransportService                Get-Service
```

From the above list we want 'Get-Service'. If you don't know the exact name of the service, try to perform similar search as we did for the Get-Service cmdlet, using the keyword 'POP':

```
        Get-Service *Pop*
```

On the Exchange 2013 server, one service is returned from the above query:



```
Status   Name               DisplayName
------   ----               -----------
Stopped  MSExchangePop3     Microsoft Exchange POP3
Stopped  MSExchangePOP3BE   Microsoft Exchange POP3 Backend
```

These cmdlets establish that POP3 services are running on the server. How do we find configuration settings which will lead us to connection information? To find the command for POP settings, try:

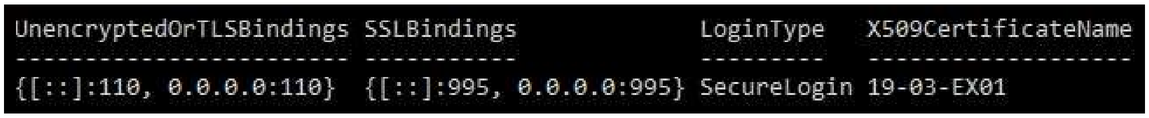
```
        Get-Command *Pop*
```

Which displays these relevant cmdlets:

```
Get-PopSettings               Set-PopSettings
Test-PopConnectivity                 Pop-Location
```

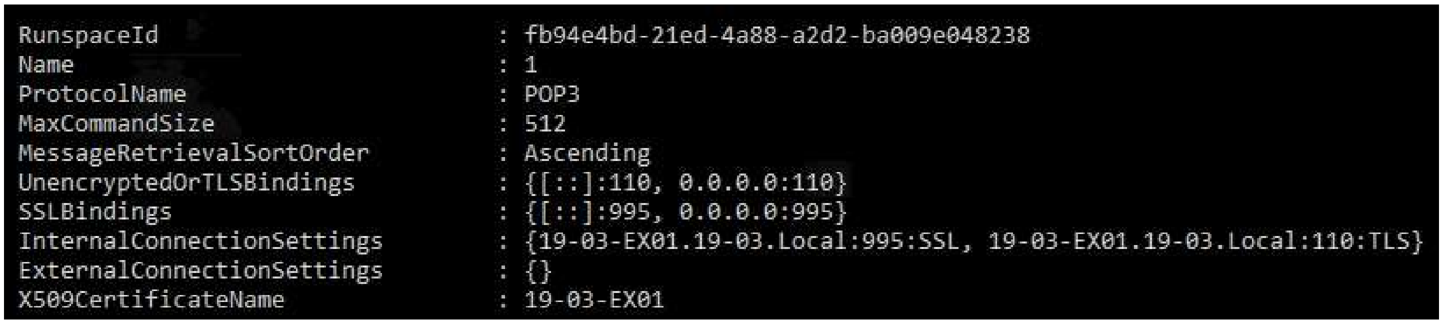
TIP: Different ways to display settings for POP3 are listed below.

Format Table (FT) presents the settings for POP3:



```
UnencryptedOrTLSBindings SSLBindings              LoginType   X509CertificateName
------------------------ -----------              ---------   -------------------
{[::]:110, 0.0.0.0:110}  {[::]:995, 0.0.0.0:995} SecureLogin 19-03-EX01
```

While Format List (FL) presents the settings for POP3 as a list:

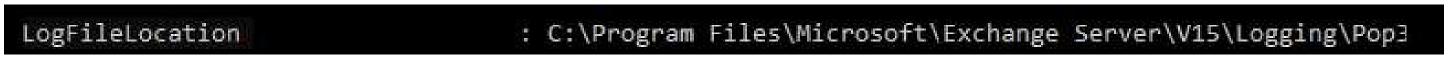


```
RunspaceId                         : fb94e4bd-21ed-4a88-a2d2-ba009e048238
Name                               : 1
ProtocolName                       : POP3
MaxCommandSize                     : 512
MessageRetrievalSortOrder          : Ascending
UnencryptedOrTLSBindings           : {[::]:110, 0.0.0.0:110}
SSLBindings                        : {[::]:995, 0.0.0.0:995}
InternalConnectionSettings         : {19-03-EX01.19-03.Local:995:SSL, 19-03-EX01.19-03.Local:110:TLS}
ExternalConnectionSettings         : {}
X509CertificateName                : 19-03-EX01
```

The command needed to find POP3 settings is:

```
Get-PopSettings
```

We need detailed information from each cmdlet. In order to get this, we need to add '| fl' to the end of each cmdlet. For POP3 the log location is revealed (in this particular Exchange installation) to be:



```
LogFileLocation                    : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Pop3
```

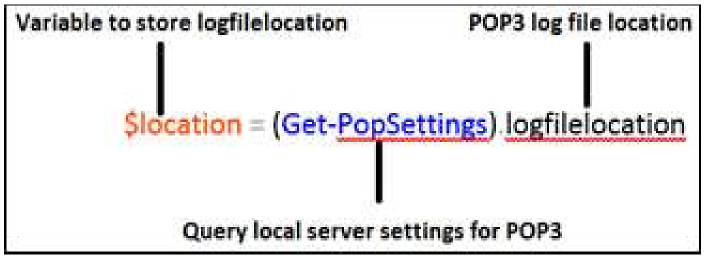
Make sure that Protocol logging (a verbose log of all connections on a particular protocol) is enabled, this is not a default setting. To enable logging for POP3 run this PowerShell cmdlet:

```
Set-PopSettings -ProtocolLogEnabled $True
```

Don't forget to restart the POP service, otherwise the log directories don't get created:

```
Restart-Service MSExchangePOP3
```

If there are log files in the directory, PowerShell can parse these files looking for relevant values. First, we need to get the location of the files to work with:



```
$location = (Get-PopSettings).logfilelocation
```

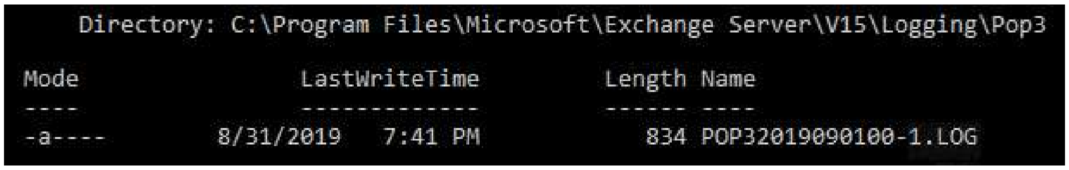
Variable to store logfilelocation
POP3 log file location
Query local server settings for POP3

Note: The .logfilelocation is a specific property retrieved by Get-PopSettings/

Once we have the location of the POP3 log files, we can get a full list of the files using the Get-ChildItem cmdlet (which provides 'child' items under a specified folder):

```
        Get-ChildItem $Location
```

The above cmdlet, using the file location stored in $location and provides this result:



```
    Directory: C:\Program Files\Microsoft\Exchange Server\V15\Logging\Pop3

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        8/31/2019   7:41 PM            834 POP32019090100-1.LOG
```

If there is more than one file present, we will need to be able to store the above information and analyze each of the POP3 log files for the CIP (Client IP Address). We will store the results in a variable called $files:

```
        $Files = Get-ChildItem $Location
```

We now have a list of files to work with. With PowerShell we can use the Foreach loop to examine each file. Code to execute on each file will be placed in the brackets '{ }'.

```
        Foreach ($File in $Files) { }
```

Once in the loop, we need to get the name of the file from the current line in the $files array:

```
        $Name = $File.Name
```

Note: '.name' is a property of the $File variable

We now have the file location ($location) and the name of the current POP3 file ($name) and need to store this into the $Csv variable so we can search each line for the CIP value:

```
        $Csv= Import-Csv (Join-Path $Location $Name)
```

Note: Join-Path allows us to join the two variables $Location (the path) and $Name (the child path) together with a '\' delimiter to make a completely new file path.

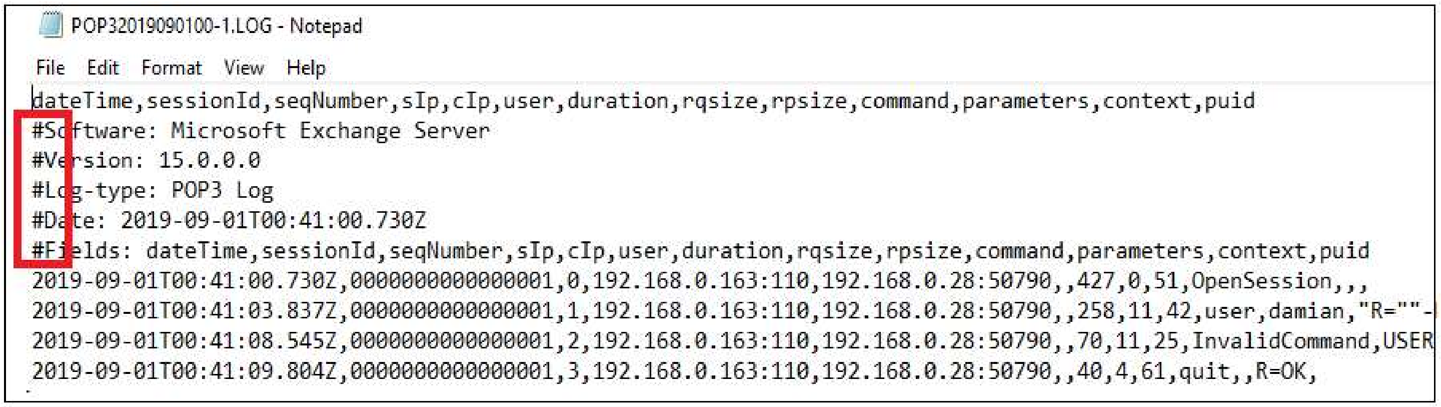
The $Csv variable will have a lot of lines to review, so we can loop using the Foreach command to review each line in the current POP3 log file:

```
        Foreach ($Line in $Csv) {
```

The POP3 log files contain some commented lines, with '#' at the very beginning of the lines. PowerShell can ignore these lines with some help:

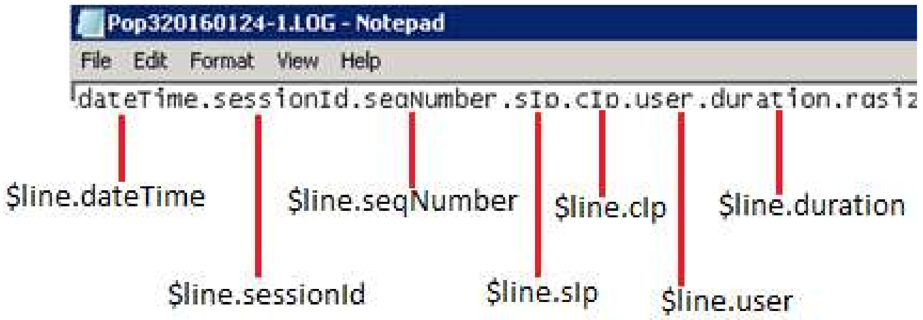
```
        If ($Line -Like "#") { }
```

The above line looks for a line with '#' in it and skips it due to the empty values. This process will happen as we loop through each line in the CSV. Below is a sample version of an Exchange 2013 POP3 log file. Notice the lines with a '#' in front of them:



```
dateTime,sessionId,seqNumber,sIp,cIp,user,duration,rqsize,rpsize,command,parameters,context,puid
#Software: Microsoft Exchange Server
#Version: 15.0.0.0
#Log-type: POP3 Log
#Date: 2019-09-01T00:41:00.730Z
#Fields: dateTime,sessionId,seqNumber,sIp,cIp,user,duration,rqsize,rpsize,command,parameters,context,puid
2019-09-01T00:41:00.730Z,0000000000000001,0,192.168.0.163:110,192.168.0.28:50790,,427,0,51,OpenSession,,,
2019-09-01T00:41:03.837Z,0000000000000001,1,192.168.0.163:110,192.168.0.28:50790,,258,11,42,user,damian,"R=""-
2019-09-01T00:41:08.545Z,0000000000000001,2,192.168.0.163:110,192.168.0.28:50790,,70,11,25,InvalidCommand,USER
2019-09-01T00:41:09.804Z,0000000000000001,3,192.168.0.163:110,192.168.0.28:50790,,40,4,61,quit,,R=OK,
```

Note: How variables store data:


Pop320160124-1.LOG - Notepad
File Edit Format View Help
dateTime.sessionId.seqNumber.sIp.cIp.user.duration.rqsiz
$line.dateTime
$line.seqNumber
$line.cIp
$line.duration
$line.sessionId
$line.sIp
$line.user

If the line in the CSV file does not have a '# in front of it, we can process it. To do so we can use an 'Else {' code section. This new section of code will work with lines with no '#' in front of it.

```
        Else {
```

Here we are isolating the CIP (Client IP) value. The $info variable will store the CIP value in the current line of the current log file:

```
        $Info = $Line.Cip
```

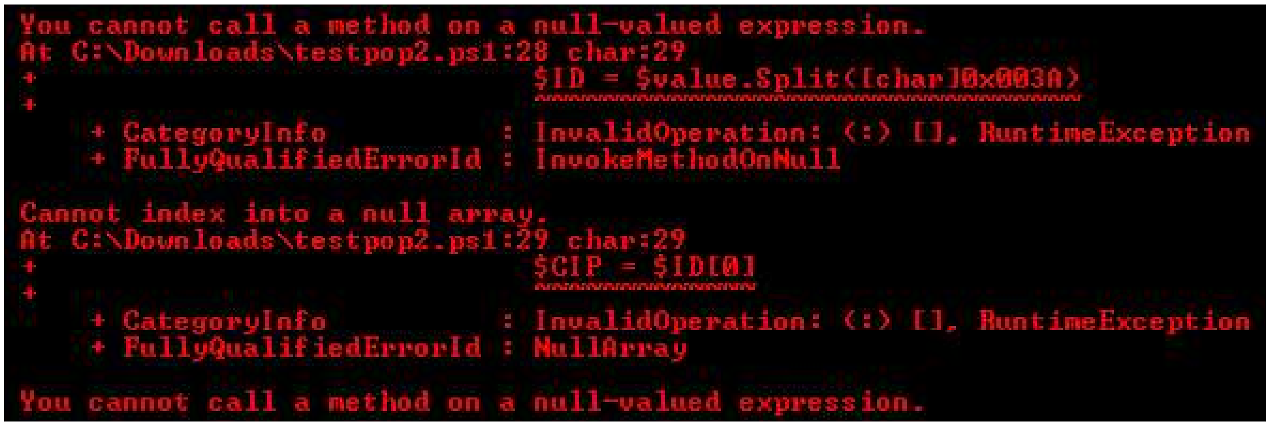
Part of troubleshooting the script is to remove bugs or workaround that data that is ingested into variables. In this case, $Info has a value of CIP for the first line. This line effectively says if $info does not have a value of 'cip' then do these steps.

```
        If ($Info -ne "Cip") {
```

This line will loop through all the values in the $info variable. Without this step, we would receive an error about indexing a null array as well a perform a certain operation on the data.

```
        Foreach ($Value in $Info) {
```

While working on this section of code, an error was generated in testing:



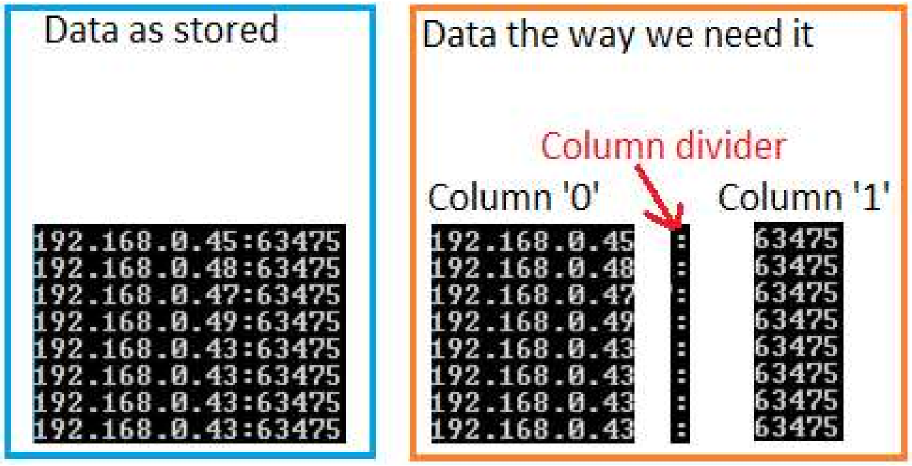
```
You cannot call a method on a null-valued expression.
At C:\Downloads\testpop2.ps1:28 char:29
+                             $ID = $value.Split([char]0x003A)
+                             ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvokeMethodOnNull

Cannot index into a null array.
At C:\Downloads\testpop2.ps1:29 char:29
+                             $CIP = $ID[0]
+                             ~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : NullArray

You cannot call a method on a null-valued expression.
```

PowerShell ran into an issue and the solution is to loop through the variable as it is an array of values. The 'Cannot index into a null array' is the clue. The fact that the variable is an array provides an opportunity to work with the data better in a loop. The Foreach loop will allow PowerShell to go through each line and process it correctly.

This line will allow us to split the $value variable into chunks, using the ':' character as the splitting point. 0x003A stands for the ':' character.

```
      $ID = $Value.Split([Char]0x003A)
```

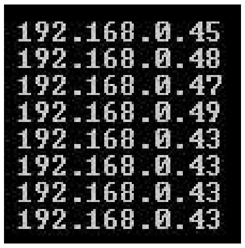
Data in the $Value variable prior to splitting:



$ID[0] is the first value stored in the $ID array. ($ID[1] would be the port number, which is column 1 above):

```
      $CIP = $ID[0]
```

This data looks like this:



```
192.168.0.45
192.168.0.48
192.168.0.47
192.168.0.49
192.168.0.43
192.168.0.43
192.168.0.43
192.168.0.43
```

Here we are combining all of the values found in $Cip and storing them in an array called $CipResults:

```
$CipResults += $Cip
```

Because we are storing variables in the $CipResults variable, we also need to define the variable as an array. It is not uncommon to have to define this after a script is almost complete especially if the data type to be stored in a variable are not known. The best practice is to place these lines at the top of the script:

```
$CipResults = @()
```

After the script reads all the POP3 log files and stores the information in a variable (this part could take time depending on the number of log files and size of the files that were are searching through). PowerShell can now display a list of the CIP values:

```
$CipResults | Sort -Unique
```

This line will take the values in $CipResults, find all unique values and sort them. If for example, if the data set looked like this:

```
192.168.0.43,192.168.0.43,192.168.0.43,192.168.0.45,192.168.0.46,192.168.0.46,192.168.0.43
,192.168.0.47
```

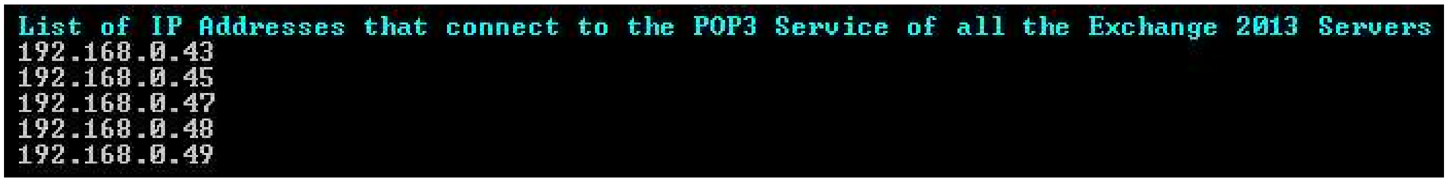
You would see this displayed at the end:

```
192.168.0.43 192.168.0.45 192.168.0.46 192.168.0.47
```

Final Script

```
# Define Variables
$CipResults = @()
# Get Files for parsing
$Location = (Get-PopSettings).LogfileLocation
$Files = Get-ChildItem $Location
# Loop for each file to get IP Addresses
Foreach ($File in $Files) {
$Name = $File.Name
$Csv= Import-Csv (Join-Path $location $name)
Foreach ($Line in $Csv) {
If ($Line -Like "#") { }
Else {
# Get the Client IP
$Info = $Line.Cip
If ($Info -ne "Cip") {
Foreach ($Value in $Info) {
# Client IP also contains the port number which we will remove here
$ID = $Value.Split([Char]0x003A)
$CIP = $ID[0]
$Cipresults += $cip
}
}
}
}
}
# Optional - Remove Duplicates
Write-host "List of IP Addresses that connect to the POP3 Service of all the Exchange 2013
 Servers." $cipResults | Sort -Unique
```

Results produced by the script:



```
List of IP Addresses that connect to the POP3 Service of all the Exchange 2013 Servers
192.168.0.43
192.168.0.45
192.168.0.47
192.168.0.48
192.168.0.49
```

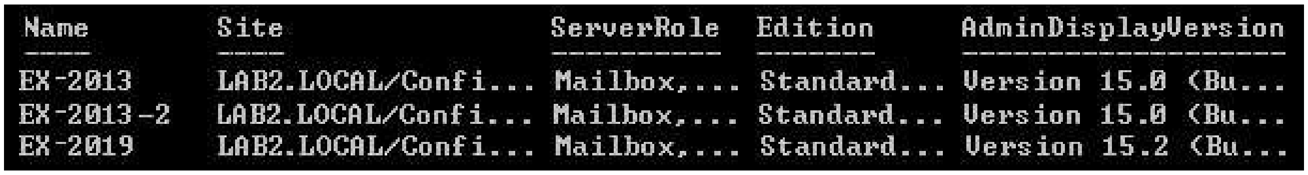
Limitations and How to Overcome Them

The PowerShell script used for determining POP3 Connections is written to query one local server. What if the Exchange organization contains a large number of servers? In a large migration, that could be a problem. Do we want to run the script separately on each server? Or do we want to use PowerShell to handle multiple queries?

How do we overcome this problem?

First, since this scenario is built off Exchange 2013 (in a migration to Exchange 2019), we'll assume we need to check each Exchange 2013 server for the same information. A common solution when dealing with a large number of servers is running a loop with the code run against each server. There are exceptions, but in general expanding the same code to be run against multiple objects is one of the main advantages of PowerShell. First, a list of Exchange Servers is needed in order to know what servers are being queried in the script:

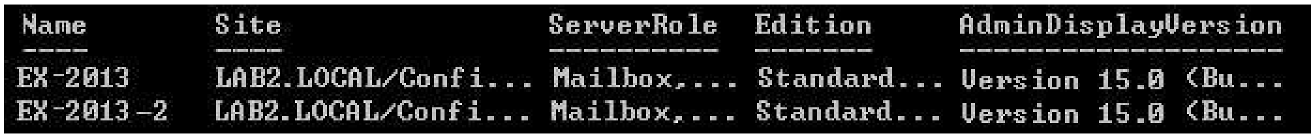
```
Get-ExchangeServer
```



```
Name        Site                  ServerRole  Edition     AdminDisplayVersion
----        ----                  ----------  -------     -------------------
EX-2013     LAB2.LOCAL/Confi...  Mailbox,...  Standard... Version 15.0 (Bu...
EX-2013-2   LAB2.LOCAL/Confi...  Mailbox,...  Standard... Version 15.0 (Bu...
EX-2019     LAB2.LOCAL/Confi...  Mailbox,...  Standard... Version 15.2 (Bu...
```

Unfortunately, we get all Exchange Servers even the Exchange Server 2019 servers. We need to filter those servers out and leave only 2013 servers behind. Exchange 2013 Servers have the value of "Version 15.0 (Bu...)" for the AdminDisplayVersion property (last column in the above screenshot). To get just those Exchange 2013 servers the Get-ExchangeServer cmdlet, is used and a filter is employed to get the results for that property, with that value:

```
Get-ExchangeServer | Where {$_.AdminDisplayVersion -Like "Version 15.0*"}
```



```
Name        Site                  ServerRole  Edition     AdminDisplayVersion
----        ----                  ----------  -------     -------------------
EX-2013     LAB2.LOCAL/Confi...  Mailbox,...  Standard... Version 15.0 (Bu...
EX-2013-2   LAB2.LOCAL/Confi...  Mailbox,...  Standard... Version 15.0 (Bu...
```

Now that the list contains just Exchange 2013 servers the 'Name' property is all we need. Refining the results again, we place brackets '(' ')' around the previous one-liner which allow us to pick a property from the servers, in this case 'name':

```
(Get-ExchangeServer | Where {$_.AdminDisplayVersion -Like "Version 15.0*"}).Name
```

Next will be to store the server names in a variable called '$Servers':

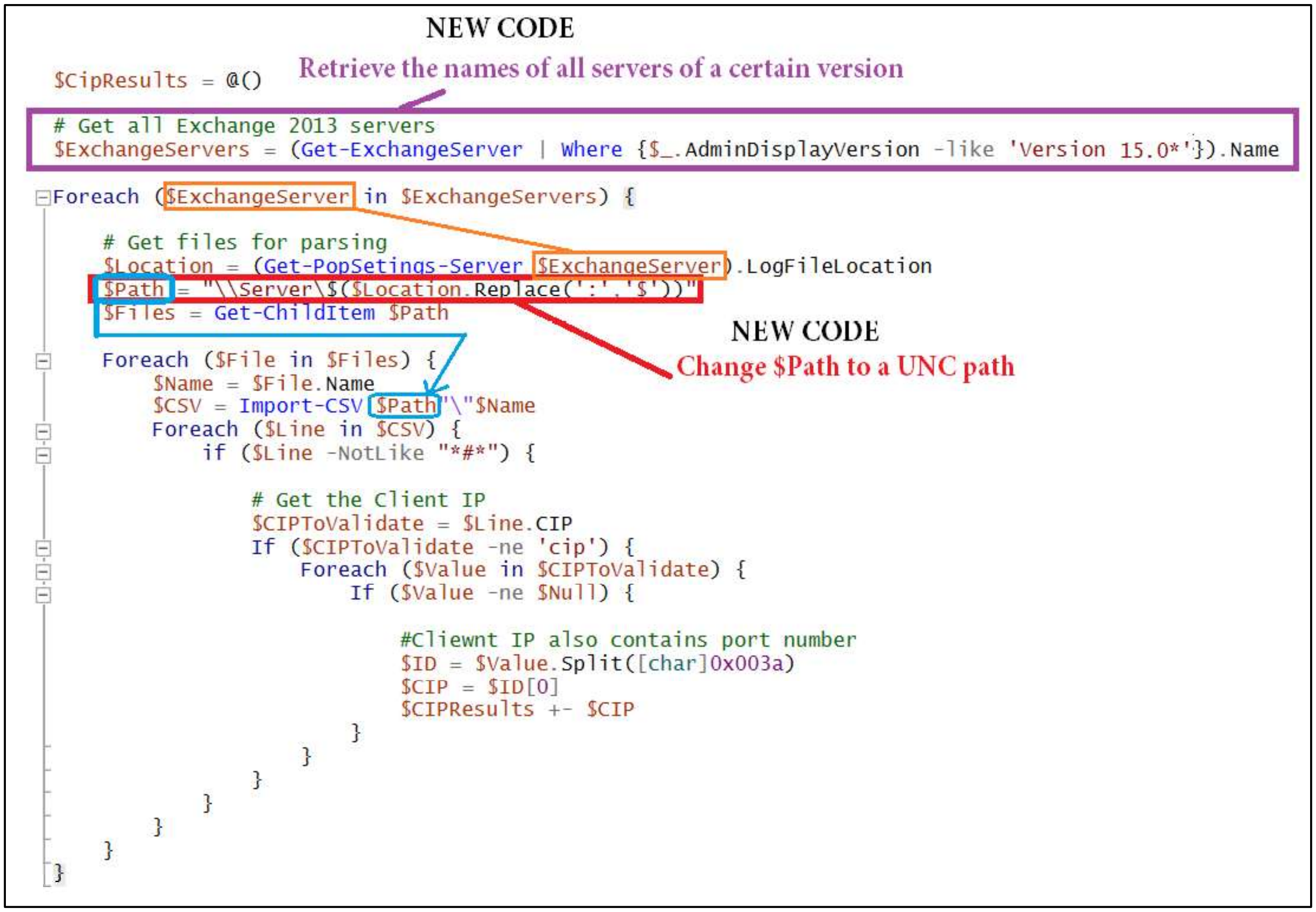
```
$Servers = (Get-ExchangeServer | Where {$_.AdminDisplayVersion -Like "Version 15.0*"}).Name
```

Then a loop is necessary to process a series of PowerShell cmdlets for each Exchange 2013 server name stored in $Server. The loop will read in the $servers variable and store the current line of values in a variable called $Server (notice the singular vs plural). Inside the loop there will be code for the POP3 log analysis.

```
      Foreach ($Server in $Servers) { … insert code here .. }
```

The next step is to integrate the $Server variable into the script so that each time the Foreach loop reads another name, the correct settings can be read for the server. Not only do we need to do that, but the file name path will need to be adjusted to a UNC path in order to be read properly:

Note: new multi-server script shown below:


NEW CODE
Retrieve the names of all servers of a certain version
NEW CODE
Change $Path to a UNC path

```
$CipResults = @()

# Get all Exchange 2013 servers
$ExchangeServers = (Get-ExchangeServer | Where {$_.AdminDisplayVersion -like 'Version 15.0*'}).Name

Foreach ($ExchangeServer in $ExchangeServers) {

    # Get files for parsing
    $Location = (Get-PopSetings-Server $ExchangeServer).LogFileLocation
    $Path = "\\Server\$($Location.Replace(':','$'))"
    $Files = Get-ChildItem $Path

    Foreach ($File in $Files) {
        $Name = $File.Name
        $CSV = Import-CSV $Path"\"$Name
        Foreach ($Line in $CSV) {
            if ($Line -NotLike "*#*") {

                # Get the Client IP
                $CIPToValidate = $Line.CIP
                If ($CIPToValidate -ne 'cip') {
                    Foreach ($Value in $CIPToValidate) {
                        If ($Value -ne $Null) {

                            #Cliewnt IP also contains port number
                            $ID = $Value.Split([char]0x003a)
                            $CIP = $ID[0]
                            $CIPResults += $CIP
                        }
                    }
                }
            }
        }
    }
}
```

The results will work the same as the single server script in that multiple servers can now be queried.

Last limitation is there is the 'X' Factor that we do not know how many or how large the POP3 log files would be. For a typical migration, the last 30 days for connection detection should be sufficient. In this case we would only review logs that are <= 30 days old. To handle this, we would set a time limit using the Get-Date cmdlet and subtracting 30 days from the current date. We reuse the $path variable which refers to the file location on a remote server.

```
      $Limit = (Get-Date).AddDays(-30)
      Get-ChildItem $Path -Recurse | ? { -Not $_.PSIsContainer -And $_.CreationTime -lt $limit} |
Remove-Item
```

Conclusion for this Script

While the above script was written for Exchange Server 2013, it was written for a migration to Exchange Server 2019. The same code in the final script works on Exchange 2013, 2016 and 2019. This was verified in multiple test labs prior to putting this code in our book.
Summary of cmdlets used:

```
Get-ExchangeServer
Get-POPSettings
Get-ChildItem
Import-Csv
Write-Host
```

IMAP4

We've covered POP3, but now IMAP4 deserves attention as we are more likely to find clients connecting with IMAP4 versus POP3. In the POP3 section we were able to track down the logs and write a script to find any client connections using that protocol. Let's see what it takes to accomplish the same task with IMAP4. We will skip some of the above steps as we've 'worked out the kinks' in the way the script needs to run.

IMAP is a more modern protocol and tends to be the protocol used by mail servers that are not Exchange Servers. IMAP also operates in a different manner than the previous POP3 protocol. It does not download email from the mail server and delete it from the server. This allows for multiple clients (think multiple computers and mobile devices) to connect to the same email account and not have to worry about mail being missing. In our BYOD and multiple device age, the IMAP4 protocol provides a superior end user experience to POP3.

IMAP uses ports 143 (insecure, unencrypted or opportunistic TLS) and 993 (secure and encrypted). Commonly an ISP's (like Google) uses IMAP, as do many mail servers that run on Mac servers, Exchange server and Office 365 can enable it as well.

Similar to POP3, most engineers deal with IMAP4 for initial setup, troubleshooting and migrations. Like POP3, we will start with discovering IMAP4 connections.

Script Re-Usability

IMAP is similar enough in Exchange that the script used for POP3 can be re-used to handle IMAP4 logs on the same Exchange servers. After reviewing the code for the script, the only lines that needed to be changes were:

```
$Location = (Get-POPSettings -Server $Server).LogFileLocation
```

Which becomes:

```
$Location = (Get-IMAPSettings -Server $Server).LogFileLocation
```

And then the description line at the bottom of the script:

```
Write-Host "List of IP Addresses that connect to the POP3 Service of all the Exchange 2013
 Servers."
```

Which becomes:

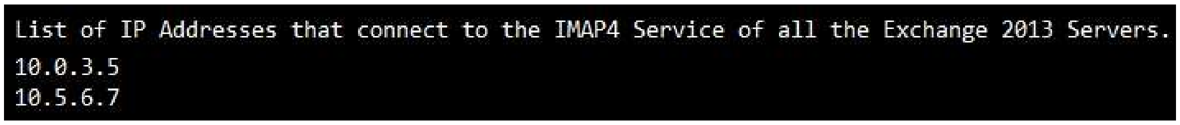
```
Write-Host "List of IP Addresses that connect to the IMAP4 Service of all the Exchange 2013
 Servers."
```

Really, that was it. The rest of the lines in the script dealt with querying server names, parsing log files and summarizing results. The similarities were key in the area of log files that POP3 and IMAP4 use. This very little effort was needed to create two completed scripts, which report on two different protocol connections for Exchange Servers.

IMAP4 Script Code

```
# Define Variables
$CipResults = @()
$Location = (Get-IMAPSettings).LogFileLocation
$Files = Get-ChildItem $Location
# Loop for each file to get IP Addresses
Foreach ($File in $Files) {
$Name = $File.Name
$Csv = Import-Csv $Location"\"$Name
Foreach ($Line in $Csv) {
If ($Line -NotLike "*#*") {
# Get the Client IP
$CIPToValidate = $Line.Cip
If ($CIPToValidate -ne "cip") {
Foreach ($Value in $CIPToValidate) {
# Client IP also contains the port number which we will remove here
$ID = $Value.Split([Char]0x003A)
$CIP = $ID[0]
$CipResults += $Cip
}
}
}
}
}
# Optional - Remove Duplicates
Write-Host "List of IP Addresses that connect to the IMAP4 Service of all the Exchange 2013
 Servers."
$CipResults | Sort -Unique
```

Results for the IMAP version of the script:



```
List of IP Addresses that connect to the IMAP4 Service of all the Exchange 2013 Servers.
10.0.3.5
10.5.6.7
```

POP3 and IMAP4 Reporting

While finding the connections made by clients or servers to Exchange Servers can be useful for long term maintenance or documenting settings for these services. The documentation can be as simple as a formatted list report created in PowerShell as complex as a complete export of settings with these settings extracted and then reformatted into an HTML report for server documentation purposes.

POP3 Setting

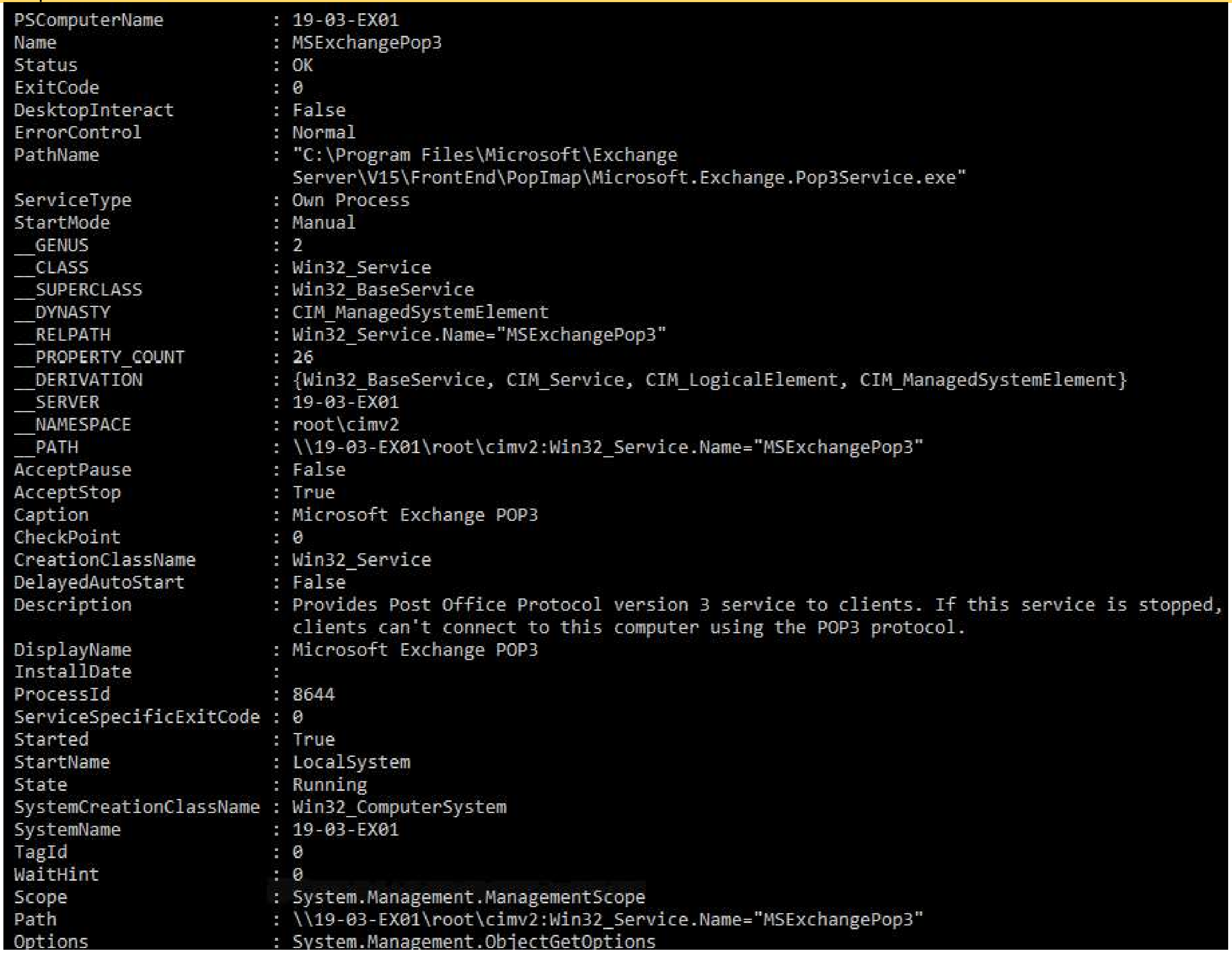
Going back to the beginning of this chapter we know we can run these commands to begin our analysis:

```
Get-Service *Pop*
Get-POPSettings
```

The Get-Service command is good, but it is not nearly detailed enough for providing information. Using WMI or CIM queries are a better bet as more information is stored:

```
Get-WmiObject Win32_Service -Property * | Where {$_.Name -Like "*pop*"} | Select *
```

Note: Notice the use of the 'Win32_Service' class in the above command. To find the appropriate class see the WMI/CIM chapter.

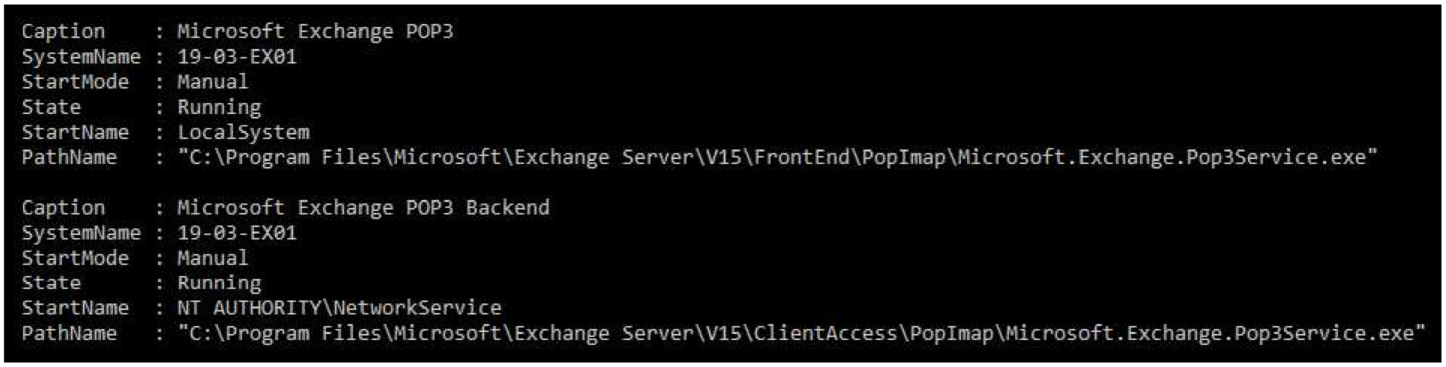


```
PSComputerName          : 19-03-EX01
Name                    : MSExchangePop3
Status                  : OK
ExitCode                : 0
DesktopInteract         : False
ErrorControl            : Normal
PathName                : "C:\Program Files\Microsoft\Exchange
                          Server\V15\FrontEnd\PopImap\Microsoft.Exchange.Pop3Service.exe"
ServiceType             : Own Process
StartMode               : Manual
__GENUS                 : 2
__CLASS                 : Win32_Service
__SUPERCLASS            : Win32_BaseService
__DYNASTY               : CIM_ManagedSystemElement
__RELPATH               : Win32_Service.Name="MSExchangePop3"
__PROPERTY_COUNT        : 26
__DERIVATION            : {Win32_BaseService, CIM_Service, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                : 19-03-EX01
__NAMESPACE             : root\cimv2
__PATH                  : \\19-03-EX01\root\cimv2:Win32_Service.Name="MSExchangePop3"
AcceptPause             : False
AcceptStop              : True
Caption                 : Microsoft Exchange POP3
CheckPoint              : 0
CreationClassName       : Win32_Service
DelayedAutoStart        : False
Description             : Provides Post Office Protocol version 3 service to clients. If this service is stopped,
                          clients can't connect to this computer using the POP3 protocol.
DisplayName             : Microsoft Exchange POP3
InstallDate             :
ProcessId               : 8644
ServiceSpecificExitCode : 0
Started                 : True
StartName               : LocalSystem
State                   : Running
SystemCreationClassName : Win32_ComputerSystem
SystemName              : 19-03-EX01
TagId                   : 0
WaitHint                : 0
Scope                   : System.Management.ManagementScope
Path                    : \\19-03-EX01\root\cimv2:Win32_Service.Name="MSExchangePop3"
Options                 : System.Management.ObjectGetOptions
```

While the amount of information may seem overwhelming, the Get-WmiObject command enables us to pick the needed details. Take some time to review each line as this information may come handy with future queries. The service account, if there is a service account, is shown as "StartName" and the path to the executable that is linked to the service on the server is listed as "PathName".

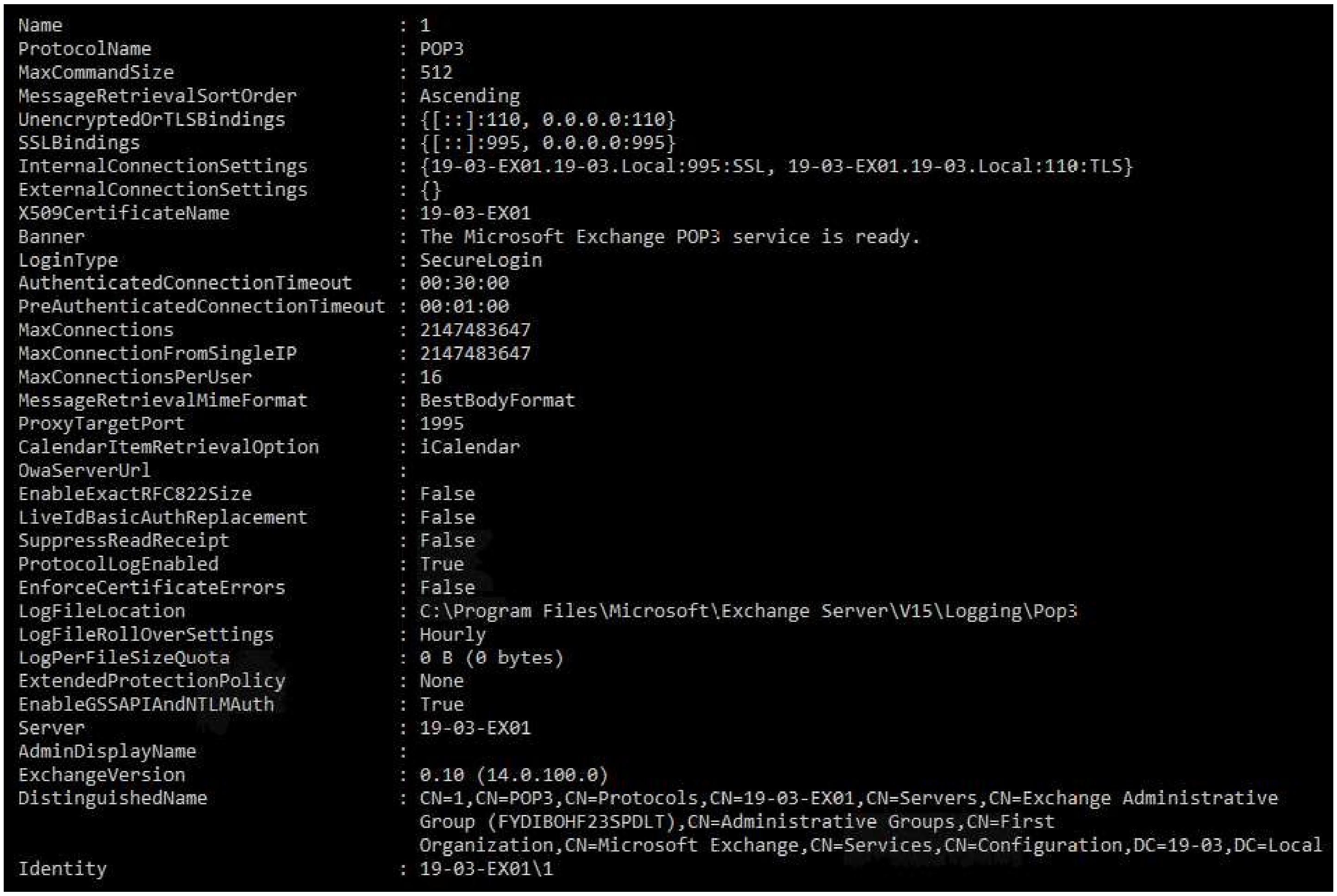
```
Get-WmiObject Win32_Service -Property * | Where {$_.Name -Like "*pop*"} | Select Caption,
 SystemName, StartMode, State, StartName, PathName | fl
```

This cmdlet reveals some key information about the POP3 services on an Exchange 2019 server:



```
Caption    : Microsoft Exchange POP3
SystemName : 19-03-EX01
StartMode  : Manual
State      : Running
StartName  : LocalSystem
PathName   : "C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\PopImap\Microsoft.Exchange.Pop3Service.exe"

Caption    : Microsoft Exchange POP3 Backend
SystemName : 19-03-EX01
StartMode  : Manual
State      : Running
StartName  : NT AUTHORITY\NetworkService
PathName   : "C:\Program Files\Microsoft\Exchange Server\V15\ClientAccess\PopImap\Microsoft.Exchange.Pop3Service.exe"
```

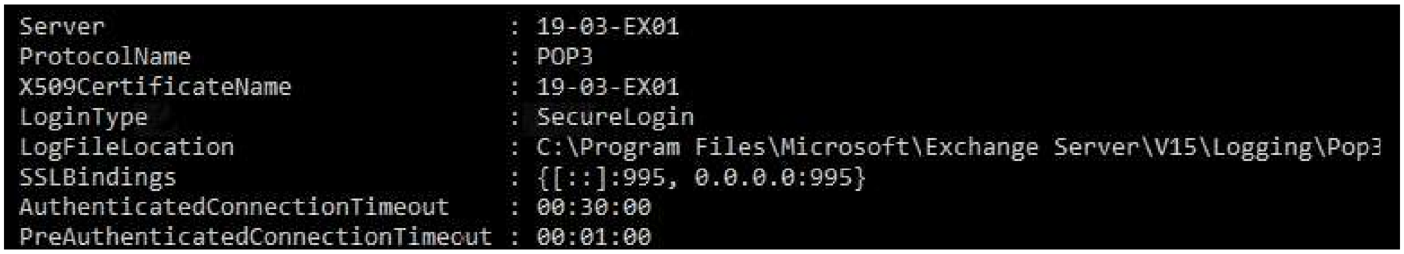
Next the 'Get-PopSettings | fl' one-liner will provide the remaining information on POP3 for the Exchange server:



```
Name                              : 1
ProtocolName                      : POP3
MaxCommandSize                    : 512
MessageRetrievalSortOrder         : Ascending
UnencryptedOrTLSBindings          : {[::]:110, 0.0.0.0:110}
SSLBindings                       : {[::]:995, 0.0.0.0:995}
InternalConnectionSettings        : {19-03-EX01.19-03.Local:995:SSL, 19-03-EX01.19-03.Local:110:TLS}
ExternalConnectionSettings        : {}
X509CertificateName               : 19-03-EX01
Banner                            : The Microsoft Exchange POP3 service is ready.
LoginType                         : SecureLogin
AuthenticatedConnectionTimeout    : 00:30:00
PreAuthenticatedConnectionTimeout : 00:01:00
MaxConnections                    : 2147483647
MaxConnectionFromSingleIP         : 2147483647
MaxConnectionsPerUser             : 16
MessageRetrievalMimeFormat        : BestBodyFormat
ProxyTargetPort                   : 1995
CalendarItemRetrievalOption       : iCalendar
OwaServerUrl                      :
EnableExactRFC822Size             : False
LiveIdBasicAuthReplacement        : False
SuppressReadReceipt               : False
ProtocolLogEnabled                : True
EnforceCertificateErrors          : False
LogFileLocation                   : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Pop3
LogFileRollOverSettings           : Hourly
LogPerFileSizeQuota               : 0 B (0 bytes)
ExtendedProtectionPolicy          : None
EnableGSSAPIAndNTLMAuth           : True
Server                            : 19-03-EX01
AdminDisplayName                  :
ExchangeVersion                   : 0.10 (14.0.100.0)
DistinguishedName                 : CN=1,CN=POP3,CN=Protocols,CN=19-03-EX01,CN=Servers,CN=Exchange Administrative
                                    Group (FYDIBOHF23SPDLT),CN=Administrative Groups,CN=First
                                    Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=19-03,DC=Local
Identity                          : 19-03-EX01\1
```

Depending on what the POP3 server is used for or what clients are connecting, settings can be chosen for documentation purposes. Here is a sample of the above services:

```
        Get-POPSettings | Fl Server,ProtocolName,X509*,Login*,LogFileL*,Unen,SSL*,*authe*
```

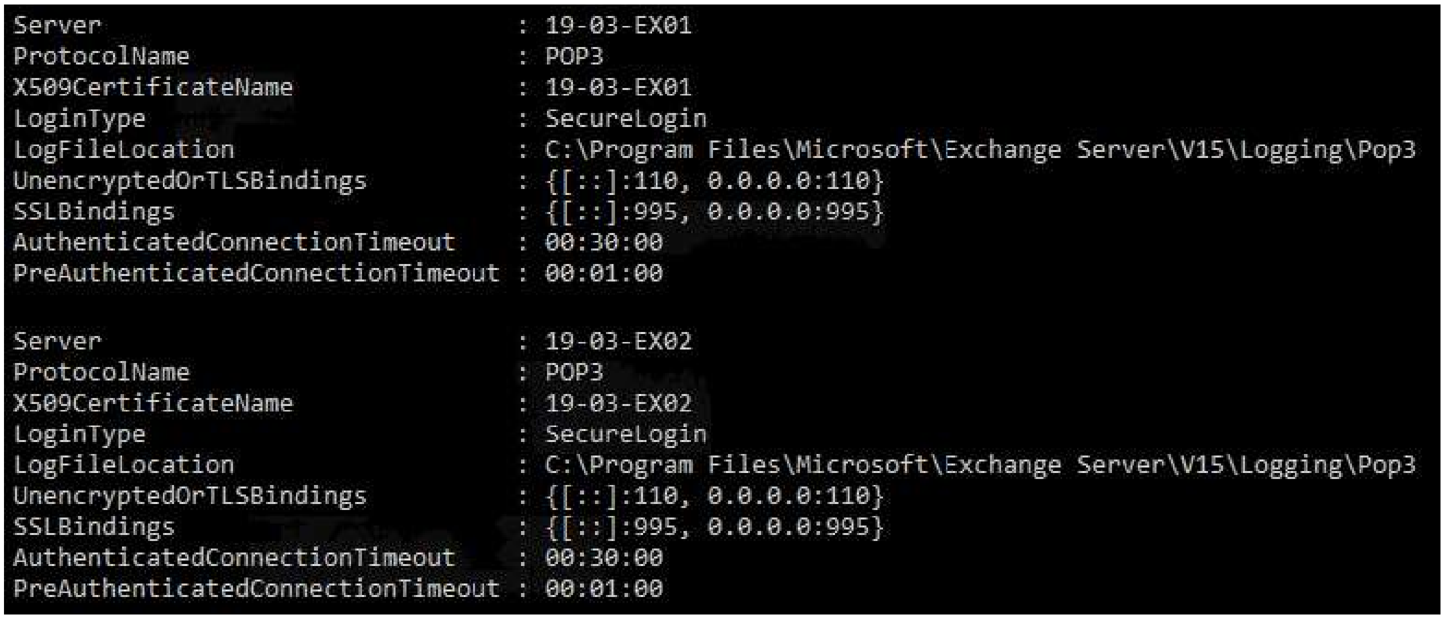


```
Server                            : 19-03-EX01
ProtocolName                      : POP3
X509CertificateName               : 19-03-EX01
LoginType                         : SecureLogin
LogFileLocation                   : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Pop3
SSLBindings                       : {[::]:995, 0.0.0.0:995}
AuthenticatedConnectionTimeout    : 00:30:00
PreAuthenticatedConnectionTimeout : 00:01:00
```

If there are multiple Exchange 2019 servers present, the above one-liner's won't be as useful because the commands only run locally and not globally against a larger environment. That is, unless these commands are modified to handle more than one server. Several available options present themselves in order to solve this conundrum.

```
(1) Loop
$Servers = Get-ExchangeServer
Foreach ($Server in $Servers) {
Get-PopSettings -Server $Server | fl Server, ProtocolName,x509*,logint*, logfilel*, unen*,
 SSL*,*authe*
}
(2) One liner
Get-ExchangeServer | Get-PopSettings | fl Server, ProtocolName, x509*, logint*, logfilel*,
 unen*, SSL*, *authe*
```

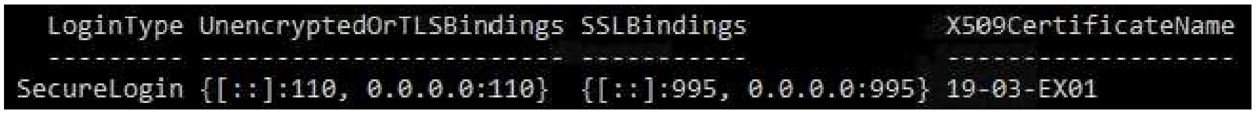
Both of these methods create the same results in the end:



```
Server                            : 19-03-EX01
ProtocolName                      : POP3
X509CertificateName               : 19-03-EX01
LoginType                         : SecureLogin
LogFileLocation                   : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Pop3
UnencryptedOrTLSBindings          : {[::]:110, 0.0.0.0:110}
SSLBindings                       : {[::]:995, 0.0.0.0:995}
AuthenticatedConnectionTimeout    : 00:30:00
PreAuthenticatedConnectionTimeout : 00:01:00

Server                            : 19-03-EX02
ProtocolName                      : POP3
X509CertificateName               : 19-03-EX02
LoginType                         : SecureLogin
LogFileLocation                   : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Pop3
UnencryptedOrTLSBindings          : {[::]:110, 0.0.0.0:110}
SSLBindings                       : {[::]:995, 0.0.0.0:995}
AuthenticatedConnectionTimeout    : 00:30:00
PreAuthenticatedConnectionTimeout : 00:01:00
```

If a POP3 connection is failing, use this command to review the LoginType and X509 certificate, as well as the Unencrypted and SSL bindings. If your POP3 application does not support a secure login, we need to change the options with 'Set-POPSettings' (see the next page for examples):

Before:

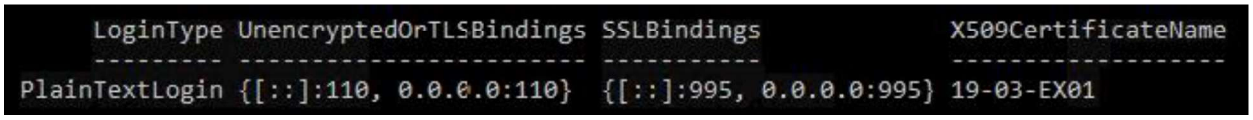


```
  LoginType UnencryptedOrTLSBindings SSLBindings             X509CertificateName
  --------- ------------------------ -----------             -------------------
SecureLogin {[::]:110, 0.0.0.0:110}  {[::]:995, 0.0.0.0:995} 19-03-EX01
```

Run this command to change the LoginType (to handle the different connection type):

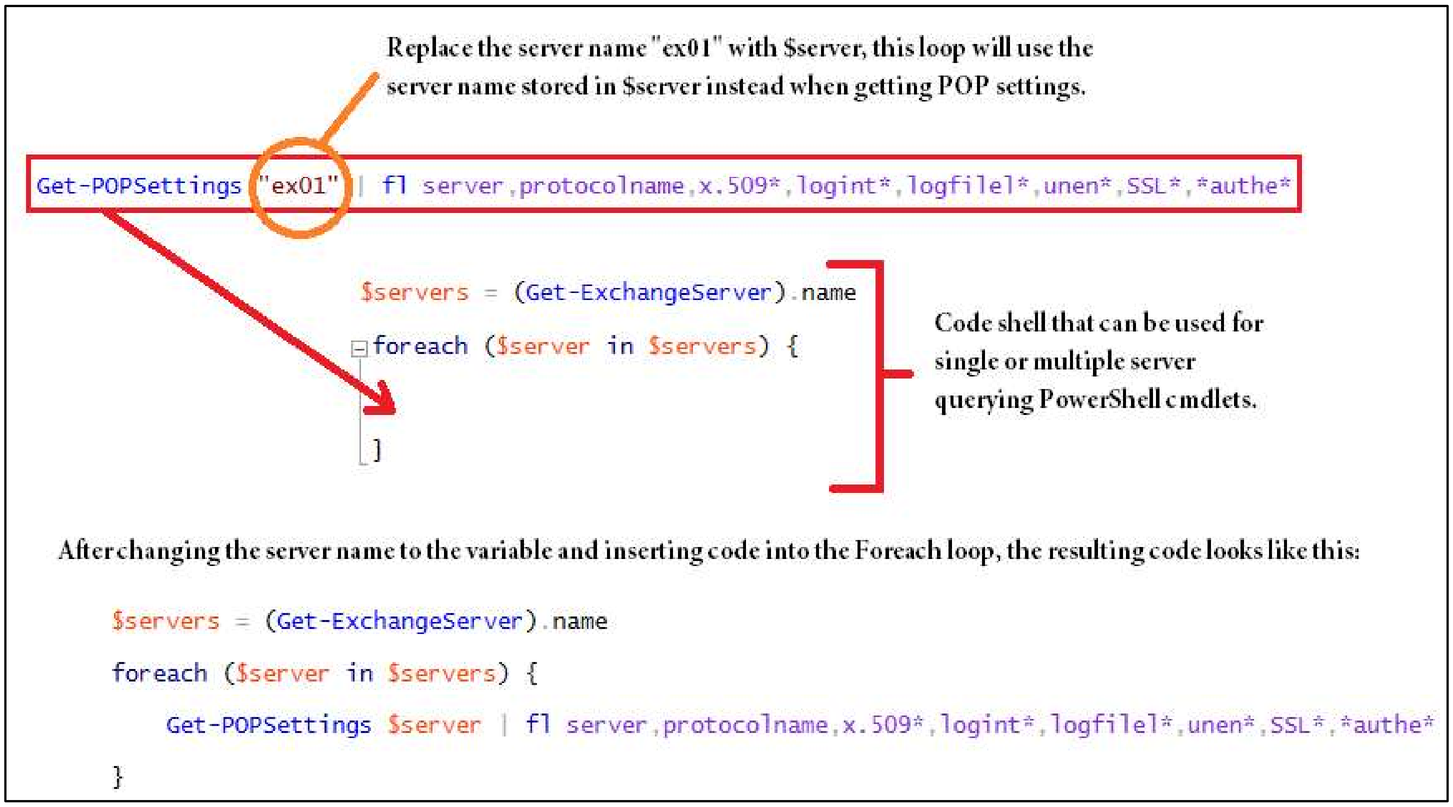
```
Set-PopSettings -LoginType PlaintextLogin
```

After:



```
     LoginType UnencryptedOrTLSBindings SSLBindings             X509CertificateName
     --------- ------------------------ -----------             -------------------
PlainTextLogin {[::]:110, 0.0.0.0:110}  {[::]:995, 0.0.0.0:995} 19-03-EX01
```

Remember to utilize Get-Help <command name> -Full in order to get information on what options are available for a particular PowerShell cmdlet. For converting single server queries to multiple server queries:


Replace the server name "ex01" with $server, this loop will use the server name stored in $server instead when getting POP settings.

```
Get-POPSettings "ex01" | fl server,protocolname,x.509*,logint*,logfilel*,unen*,SSL*,*authe*
```

```
$servers = (Get-ExchangeServer).name
foreach ($server in $servers) {

}
```

Code shell that can be used for single or multiple server querying PowerShell cmdlets.

After changing the server name to the variable and inserting code into the Foreach loop, the resulting code looks like this:

```
$servers = (Get-ExchangeServer).name
foreach ($server in $servers) {
    Get-POPSettings $server | fl server,protocolname,x.509*,logint*,logfilel*,unen*,SSL*,*authe*
}
```

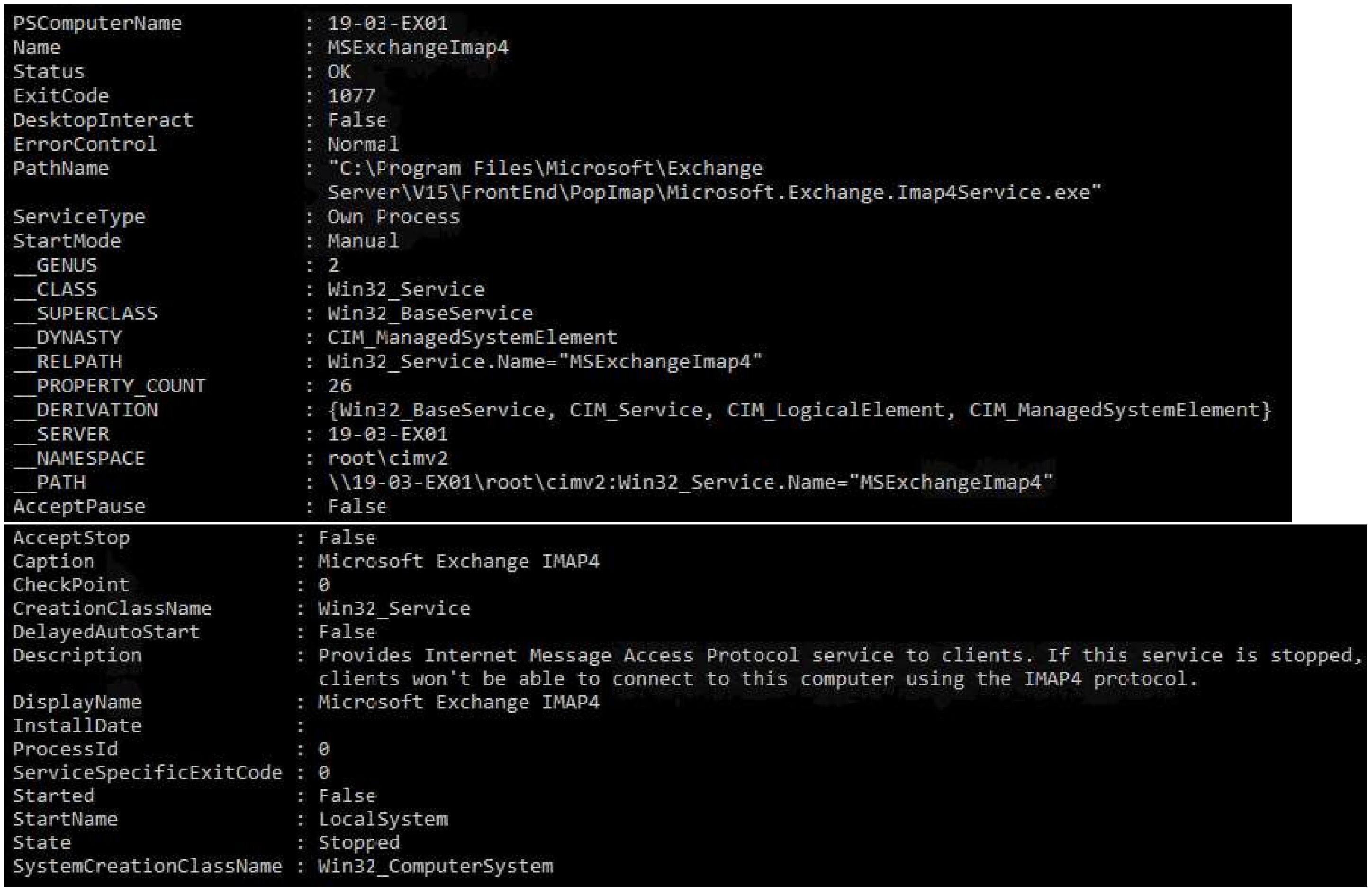
IMAP4 Setting

IMAP4 in Exchange Server 2019 contains a similar set of cmdlets and configuration data to POP3. As such, we can use very similar cmdlets to work on the IMAP4 like POP3.

```
Get-Service *IMAP*
Get-IMAPSettings
```

We know from the POP3 section that the Get-Service command is good, but it is not nearly detailed enough for providing information. Using WMI or CIM queries are a better bet as more information is provided when compared to the *-Service cmdlets, such as checking the startup mode of services (StartMode):

```
Get-WmiObject Win32_Service -Property * | Where {$_.Name -Like "*imap*"} | Select *
```

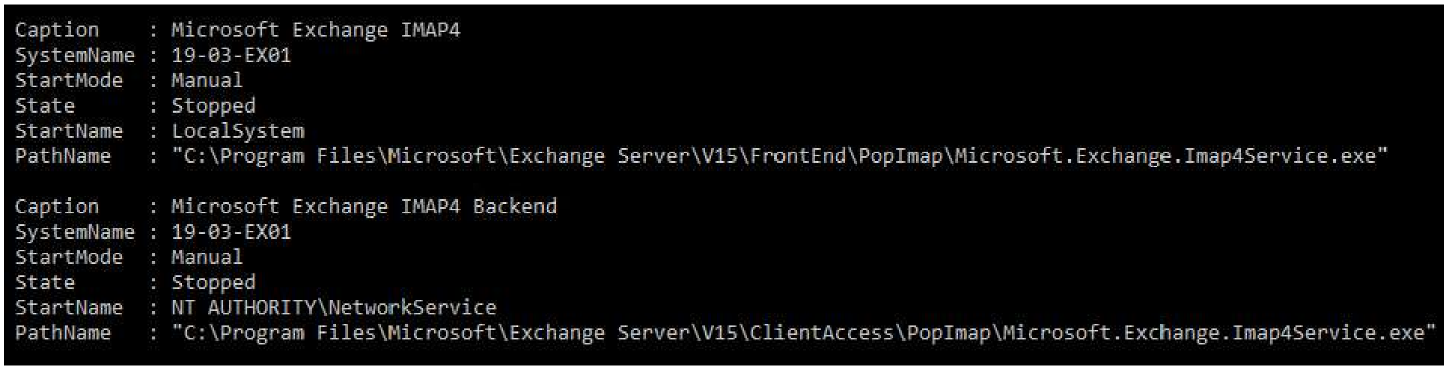


```
PSComputerName          : 19-03-EX01
Name                    : MSExchangeImap4
Status                  : OK
ExitCode                : 1077
DesktopInteract         : False
ErrorControl            : Normal
PathName                : "C:\Program Files\Microsoft\Exchange
                          Server\V15\FrontEnd\PopImap\Microsoft.Exchange.Imap4Service.exe"
ServiceType             : Own Process
StartMode               : Manual
__GENUS                 : 2
__CLASS                 : Win32_Service
__SUPERCLASS            : Win32_BaseService
__DYNASTY               : CIM_ManagedSystemElement
__RELPATH               : Win32_Service.Name="MSExchangeImap4"
__PROPERTY_COUNT        : 26
__DERIVATION            : {Win32_BaseService, CIM_Service, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                : 19-03-EX01
__NAMESPACE             : root\cimv2
__PATH                  : \\19-03-EX01\root\cimv2:Win32_Service.Name="MSExchangeImap4"
AcceptPause             : False
AcceptStop              : False
Caption                 : Microsoft Exchange IMAP4
CheckPoint              : 0
CreationClassName       : Win32_Service
DelayedAutoStart        : False
Description             : Provides Internet Message Access Protocol service to clients. If this service is stopped,
                          clients won't be able to connect to this computer using the IMAP4 protocol.
DisplayName             : Microsoft Exchange IMAP4
InstallDate             :
ProcessId               : 0
ServiceSpecificExitCode : 0
Started                 : False
StartName               : LocalSystem
State                   : Stopped
SystemCreationClassName : Win32_ComputerSystem
```

Just like POP3, we can use the Get-WmiObject command to help pick the needed details. The service account, if there is a service account, is shown as “StartName” and the path to the executable that is linked to the service on the server is listed as “PathName”.

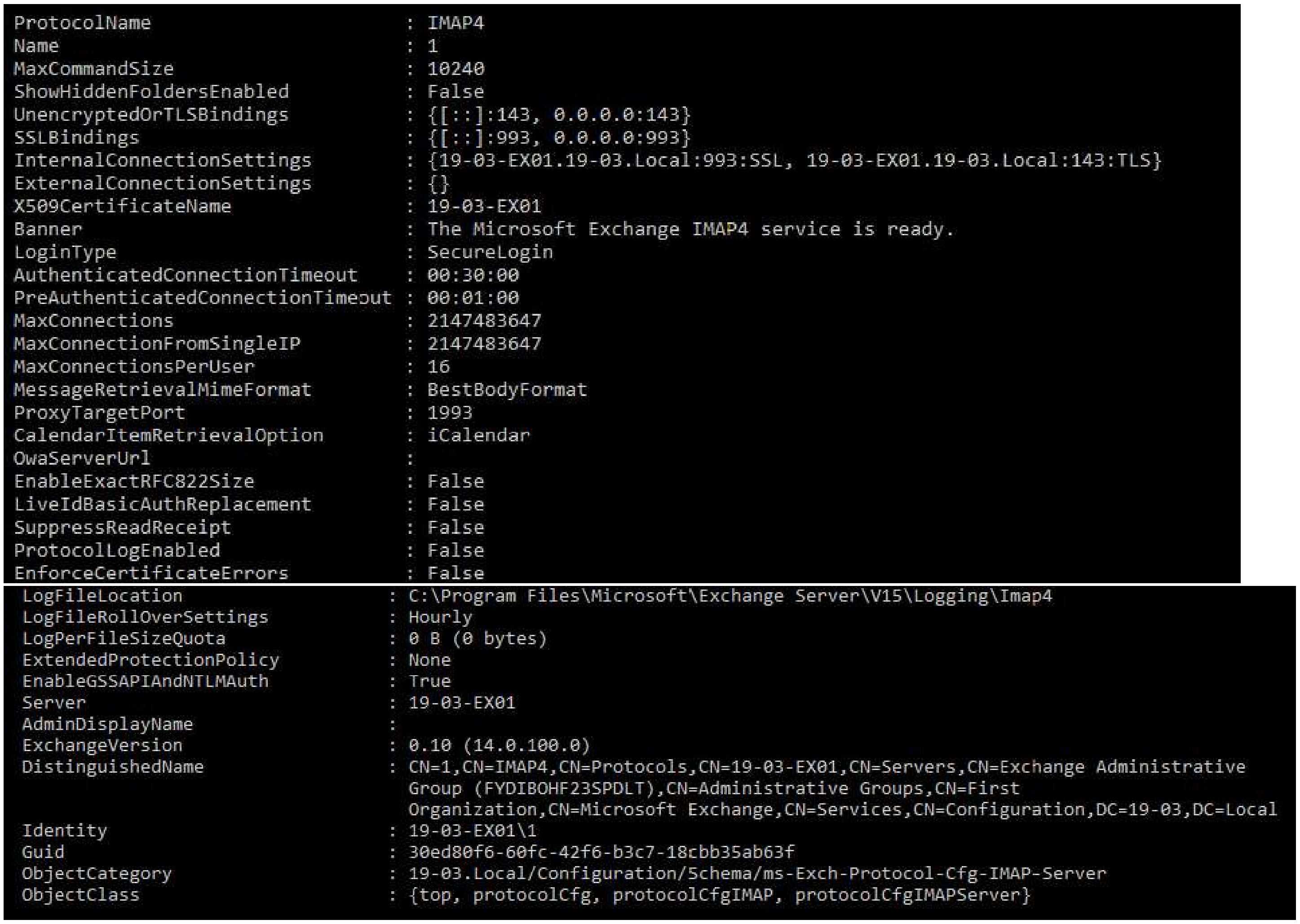
```
      Get-WmiObject Win32_Service -Property * | Where {$_.Name -Like "*imap*"} | Select Caption,
       SystemName, StartMode, State, StartName, PathName | fl
```

This cmdlet reveals some key information about the IMAP4 services on an Exchange 2019 server:



```
Caption    : Microsoft Exchange IMAP4
SystemName : 19-03-EX01
StartMode  : Manual
State      : Stopped
StartName  : LocalSystem
PathName   : "C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\PopImap\Microsoft.Exchange.Imap4Service.exe"

Caption    : Microsoft Exchange IMAP4 Backend
SystemName : 19-03-EX01
StartMode  : Manual
State      : Stopped
StartName  : NT AUTHORITY\NetworkService
PathName   : "C:\Program Files\Microsoft\Exchange Server\V15\ClientAccess\PopImap\Microsoft.Exchange.Imap4Service.exe"
```

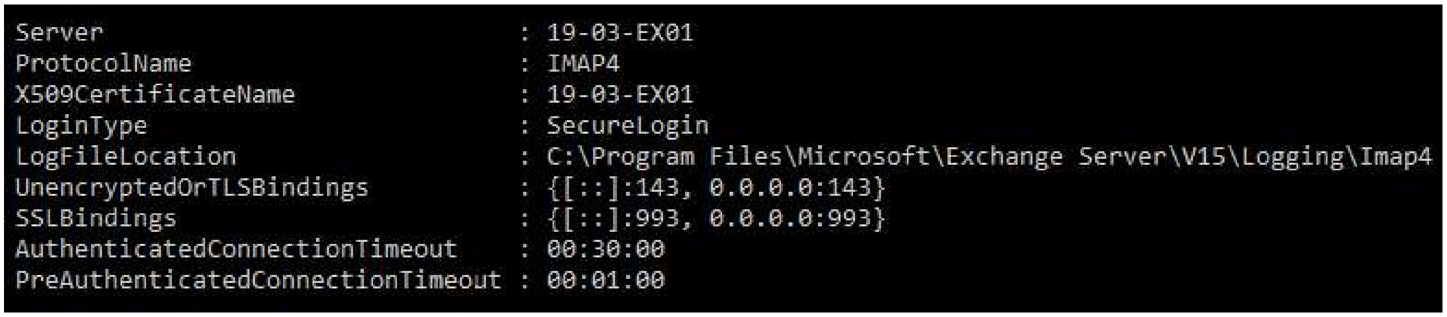
Next the ‘Get-IMAPSettings | fl’ cmdlet provides the remaining information on IMAP4 for the Exchange Server:



```
ProtocolName                       : IMAP4
Name                               : 1
MaxCommandSize                     : 10240
ShowHiddenFoldersEnabled           : False
UnencryptedOrTLSBindings           : {[::]:143, 0.0.0.0:143}
SSLBindings                        : {[::]:993, 0.0.0.0:993}
InternalConnectionSettings         : {19-03-EX01.19-03.Local:993:SSL, 19-03-EX01.19-03.Local:143:TLS}
ExternalConnectionSettings         : {}
X509CertificateName                : 19-03-EX01
Banner                             : The Microsoft Exchange IMAP4 service is ready.
LoginType                          : SecureLogin
AuthenticatedConnectionTimeout     : 00:30:00
PreAuthenticatedConnectionTimeout : 00:01:00
MaxConnections                     : 2147483647
MaxConnectionFromSingleIP          : 2147483647
MaxConnectionsPerUser              : 16
MessageRetrievalMimeFormat         : BestBodyFormat
ProxyTargetPort                    : 1993
CalendarItemRetrievalOption        : iCalendar
OwaServerUrl                       :
EnableExactRFC822Size              : False
LiveIdBasicAuthReplacement         : False
SuppressReadReceipt                : False
ProtocolLogEnabled                 : False
EnforceCertificateErrors           : False
LogFileLocation                    : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Imap4
LogFileRollOverSettings            : Hourly
LogPerFileSizeQuota                : 0 B (0 bytes)
ExtendedProtectionPolicy           : None
EnableGSSAPIAndNTLMAuth            : True
Server                             : 19-03-EX01
AdminDisplayName                   :
ExchangeVersion                    : 0.10 (14.0.100.0)
DistinguishedName                  : CN=1,CN=IMAP4,CN=Protocols,CN=19-03-EX01,CN=Servers,CN=Exchange Administrative
                                     Group (FYDIBOHF23SPDLT),CN=Administrative Groups,CN=First
                                     Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=19-03,DC=Local
Identity                           : 19-03-EX01\1
Guid                               : 30ed80f6-60fc-42f6-b3c7-18cbb35ab63f
ObjectCategory                     : 19-03.Local/Configuration/Schema/ms-Exch-Protocol-Cfg-IMAP-Server
ObjectClass                        : {top, protocolCfg, protocolCfgIMAP, protocolCfgIMAPServer}
```

Depending on what the IMAP4 server is used for or what clients are connecting, settings can be chosen for documentation purposes:

```
Get-IMAPSettings -Server $Server| fl Server,ProtocolName,x509*,Logint*,Logfilel*,
 Unen*,SSL*,*Authe*
```



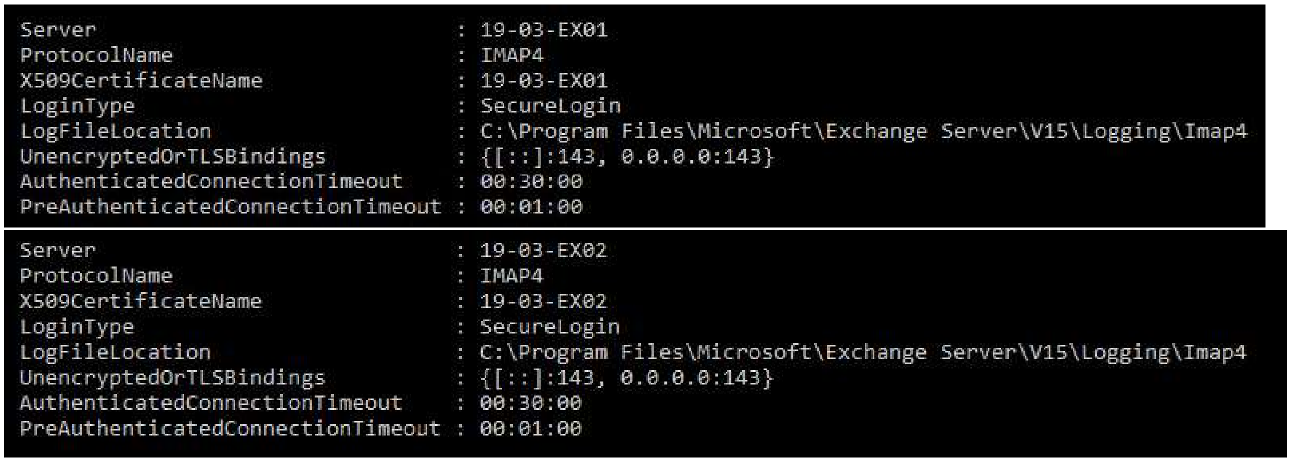
```
Server                             : 19-03-EX01
ProtocolName                       : IMAP4
X509CertificateName                : 19-03-EX01
LoginType                          : SecureLogin
LogFileLocation                    : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Imap4
UnencryptedOrTLSBindings           : {[::]:143, 0.0.0.0:143}
SSLBindings                        : {[::]:993, 0.0.0.0:993}
AuthenticatedConnectionTimeout     : 00:30:00
PreAuthenticatedConnectionTimeout : 00:01:00
```

Just like we did with POP3, if there are multiple Exchange 2019 servers present, the above one-liner's won't be as useful because the commands only run locally and not globally against a larger environment. That is, unless these commands are modified to handle more than one server. Several available options present themselves in order to solve this conundrum.

```
(1) Loop
$Servers = Get-ExchangeServer
Foreach ($Server in $Servers) {
Get-IMAPSettings -Server $Server | fl Server, ProtocolName, x509*, Logint*, Logfilel*,
 Unen*, SSL*, *Authe*
}
```

```
      (2) One liner
      Get-ExchangeServer | Get-IMAPSettings | fl Server, ProtocolName, X509*, Logint*, Logfilel*,
       Unen*,SL*, *Authe*
```

Both of these methods create the same results in the end:



```
Server                              : 19-03-EX01
ProtocolName                        : IMAP4
X509CertificateName                 : 19-03-EX01
LoginType                           : SecureLogin
LogFileLocation                     : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Imap4
UnencryptedOrTLSBindings            : {[::]:143, 0.0.0.0:143}
AuthenticatedConnectionTimeout      : 00:30:00
PreAuthenticatedConnectionTimeout   : 00:01:00

Server                              : 19-03-EX02
ProtocolName                        : IMAP4
X509CertificateName                 : 19-03-EX02
LoginType                           : SecureLogin
LogFileLocation                     : C:\Program Files\Microsoft\Exchange Server\V15\Logging\Imap4
UnencryptedOrTLSBindings            : {[::]:143, 0.0.0.0:143}
AuthenticatedConnectionTimeout      : 00:30:00
PreAuthenticatedConnectionTimeout   : 00:01:00
```

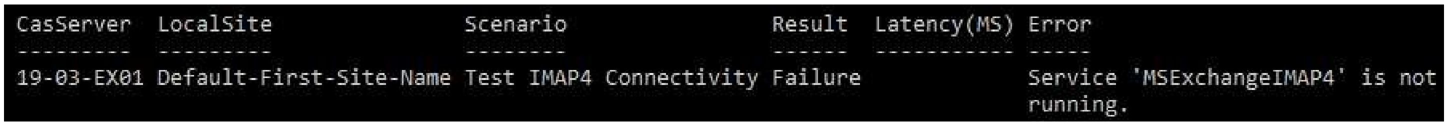
If an IMAP4 connection is failing, use this command to review the LoginType and X509 certificate, as well as the Unencrypted and SSL bindings. If your IMAP4 application does not require a secure login, the options need to change.

Testing POP3 and IMAP Connections

Exchange Server 2019 includes a series of test-xxx cmdlets that enable an Exchange admin to test various parts of Exchange to make sure they are functional. Utilizing these cmdlets would allow the construction of a server health script. After prepping the environment for testing, the Test-IMAPConnectivity cmdlet will now generate results. The results would quickly show what mailboxes have IMAP and which have POP3 enabled. On Exchange 2019 servers, POP3 and IMAP4 are disabled and the test cmdlets reveal this configuration:

IMAP4

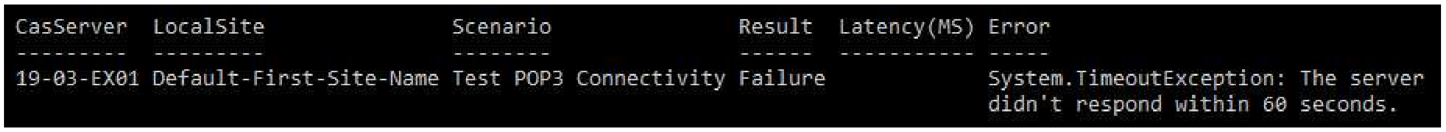
```
      Test-ImapConnectivity -ClientAccessServer:19-03-EX01 |Ft -Auto
```



```
CasServer  LocalSite               Scenario                 Result  Latency(MS) Error
---------  ---------               --------                 ------  ----------- -----
19-03-EX01 Default-First-Site-Name Test IMAP4 Connectivity Failure             Service 'MSExchangeIMAP4' is not
                                                                                running.
```

POP3

```
      Test-POPConnectivity -ClientAccessServer:19-03-EX01 |Ft -Auto
```



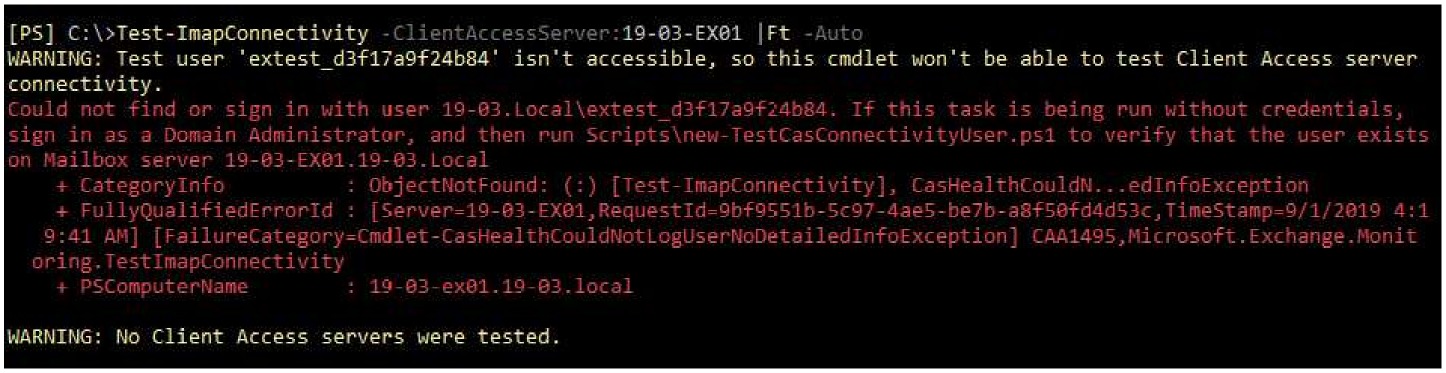
```
CasServer  LocalSite               Scenario                Result  Latency(MS) Error
---------  ---------               --------                ------  ----------- -----
19-03-EX01 Default-First-Site-Name Test POP3 Connectivity Failure             System.TimeoutException: The server
                                                                               didn't respond within 60 seconds.
```

In a troubleshooting scenario, this cmdlet could verify that a service is up and connections are good. If one user reports an issue, the command can be tailored to that one user. To do so, specifying credentials will enable the test cmdlet to connect to the POP3 or IMAP4 services with those credentials, effectively impersonating the user (place this at the end of the IMAP4 and POP3 one-liners above):

```
-MailboxCredential:(Get-Credential 19-03\administrator)
```

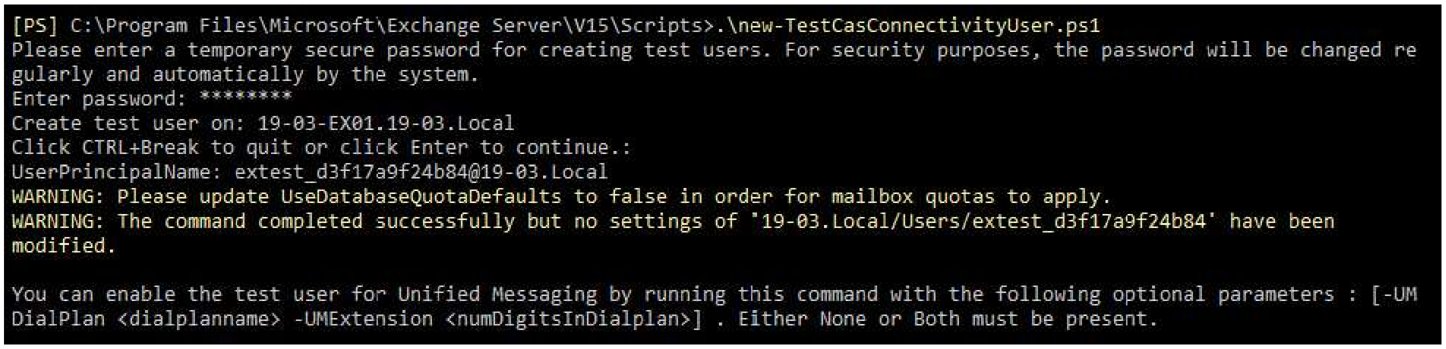
If this command succeeds and the user still cannot connect, comparing the settings on the application being used to the service settings is key. Specifically looking at the LoginType. Some applications will not handle 'SecureOnly' (the encrypted version of POP3 or IMAP4) well at all.

Note: If while running test cmdlets, this error occurs:



```
[PS] C:\>Test-ImapConnectivity -ClientAccessServer:19-03-EX01 |Ft -Auto
WARNING: Test user 'extest_d3f17a9f24b84' isn't accessible, so this cmdlet won't be able to test Client Access server
connectivity.
Could not find or sign in with user 19-03.Local\extest_d3f17a9f24b84. If this task is being run without credentials,
sign in as a Domain Administrator, and then run Scripts\new-TestCasConnectivityUser.ps1 to verify that the user exists
on Mailbox server 19-03-EX01.19-03.Local
    + CategoryInfo          : ObjectNotFound: (:) [Test-ImapConnectivity], CasHealthCouldN...edInfoException
    + FullyQualifiedErrorId : [Server=19-03-EX01,RequestId=9bf9551b-5c97-4ae5-be7b-a8f50fd4d53c,TimeStamp=9/1/2019 4:1
   9:41 AM] [FailureCategory=Cmdlet-CasHealthCouldNotLogUserNoDetailedInfoException] CAA1495,Microsoft.Exchange.Monit
  oring.TestImapConnectivity
    + PSComputerName        : 19-03-ex01.19-03.local

WARNING: No Client Access servers were tested.
```

This means we need to create a test account, as the error message states, using the new-TestCasConnectivityUser.ps1 script:

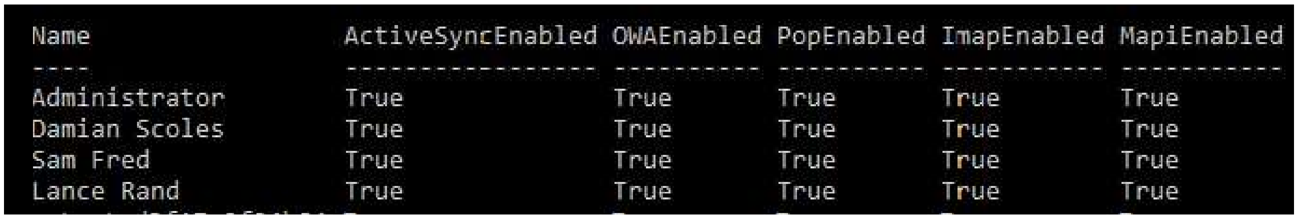


```
[PS] C:\Program Files\Microsoft\Exchange Server\V15\Scripts>.\new-TestCasConnectivityUser.ps1
Please enter a temporary secure password for creating test users. For security purposes, the password will be changed re
gularly and automatically by the system.
Enter password: ********
Create test user on: 19-03-EX01.19-03.Local
Click CTRL+Break to quit or click Enter to continue.:
UserPrincipalName: extest_d3f17a9f24b84@19-03.Local
WARNING: Please update UseDatabaseQuotaDefaults to false in order for mailbox quotas to apply.
WARNING: The command completed successfully but no settings of '19-03.Local/Users/extest_d3f17a9f24b84' have been
modified.

You can enable the test user for Unified Messaging by running this command with the following optional parameters : [-UM
DialPlan <dialplanname> -UMExtension <numDigitsInDialplan>] . Either None or Both must be present.
```

IMAP4 and POP3 – User Settings

One of the often forgotten PowerShell cmdlets is Get-CASMailbox. Seems like an odd cmdlet, but it turns out to be very convenient. The Get-CASMailbox can be quite useful in providing information on what protocols a user with a mailbox can use to connect to an Exchange 2019 server - but on a mailbox level and not a server level. In the above screenshot, all mailboxes are enabled for EAS (ActiveSync), OWA, POP, IMAP and MAPI. If one were to run a simple one-liner like this:

```
Get-Mailbox | Get-CASMailbox
```



```
Name                 ActiveSyncEnabled OWAEnabled PopEnabled ImapEnabled MapiEnabled
----                 ----------------- ---------- ---------- ----------- -----------
Administrator        True              True       True       True        True
Damian Scoles        True              True       True       True        True
Sam Fred             True              True       True       True        True
Lance Rand           True              True       True       True        True
```

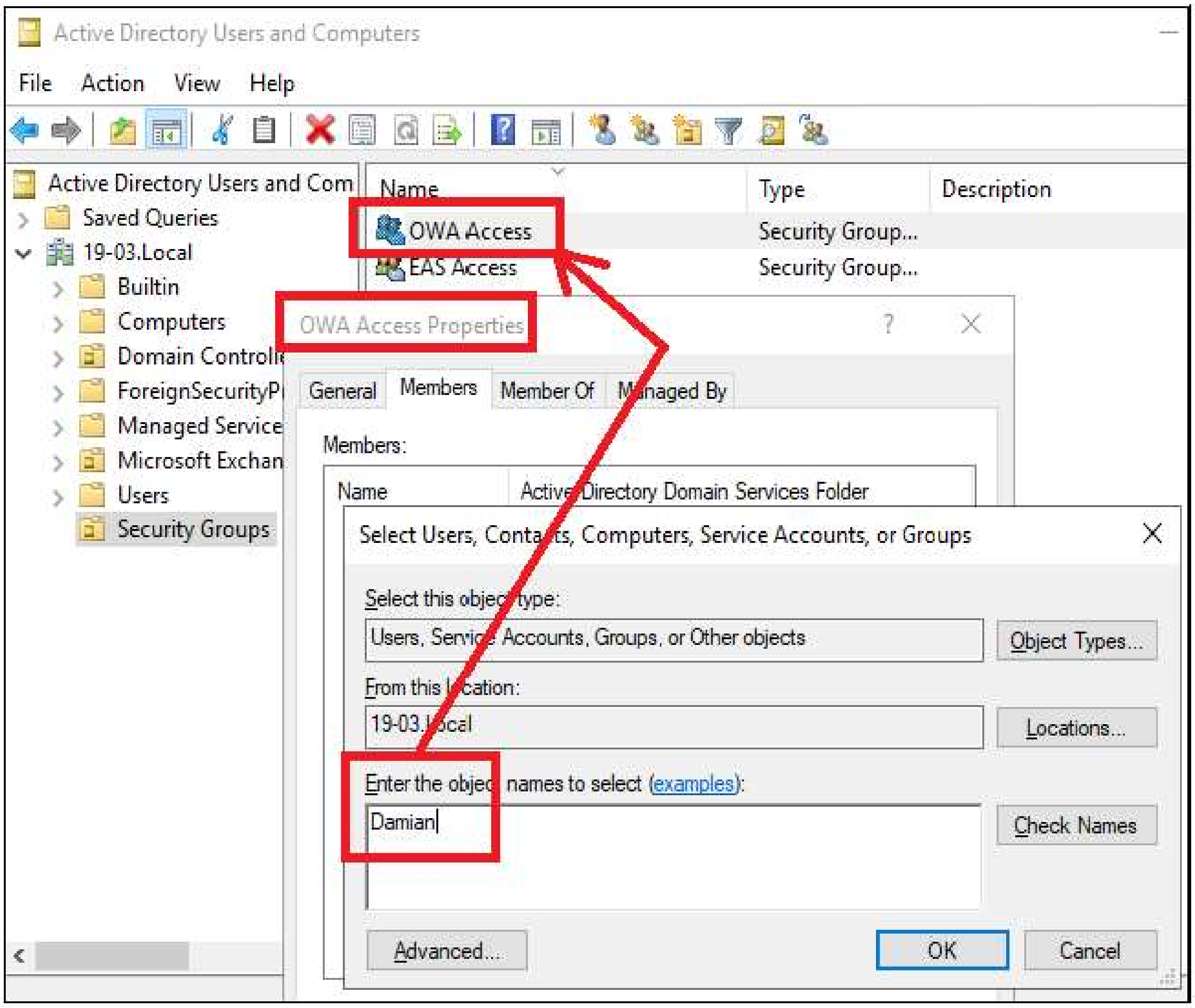
Wait. What? Get-CASMailbox? CAS did stand for Client Access Server, a role that existed in Exchange Server 2013? However, this command is used for some special operations on mailboxes in Exchange Server. In some sense it is a holdover purely based on its name. The command is useful for managing mailboxes in Exchange 2019. Get-CASMailbox can also reveal other properties of a mailbox like:

```
OWAMailboxPolicy                          OWAforDevicesEnabled      ECPEnabled
MapiHttpEnabled                           UniversalOutlookEnabled   EwsEnabled
EwsAllowOutlook                           EwsAllowMacOutlook        EwsAllowEntourage
EwsApplicationAccessPolicyEwsAllowList    EwsBlockList
ShowGalAsDefaultView                      … and more …
```

How can these properties be utilized? Let's take a scenario where a company has put in place a policy that restricts the use of certain protocols with their new Exchange 2019 servers. IT wants to restrict POP3 and IMAP access. There is a small subset of users that require the use of these protocols and IT wants to limit IMAP and POP protocols. Lastly, we need to accomplish these goals with PowerShell, and it needs to be scalable.

Where To Start

Each protocol will have two groups of users – those that have access and those that do not. The first step would be to group users that should have access as this group will be the easiest to build and maintain rather than trying to maintain a group of those to block access to a resource. In Active Directory, create a group called "POP3 Access" and another call "IMAP Access". We then add users who will have access to these resources like so: (using Active Directory Users and Computers).



Select this object type:
Users, Service Accounts, Groups, or Other objects

From this location:
19-03.local

Enter the object names to select (examples):
Damian

Now that the groups are populated with users a script will be needed to enforce these access conditions. What is needed in order to build the script?

1. Need to store this user list in a variable to determine access or use the group name in the script as the determining factor for access in the script.
2. What access should be granted or denied depending on members?
3. What commands we can use to query this information in Exchange?
4. What commands we can use to add or remove access to the resource?
5. How often do we would want to reinforce this action?

PowerShell Cmdlet Determination

Storing the names of users who need access or using the group name?

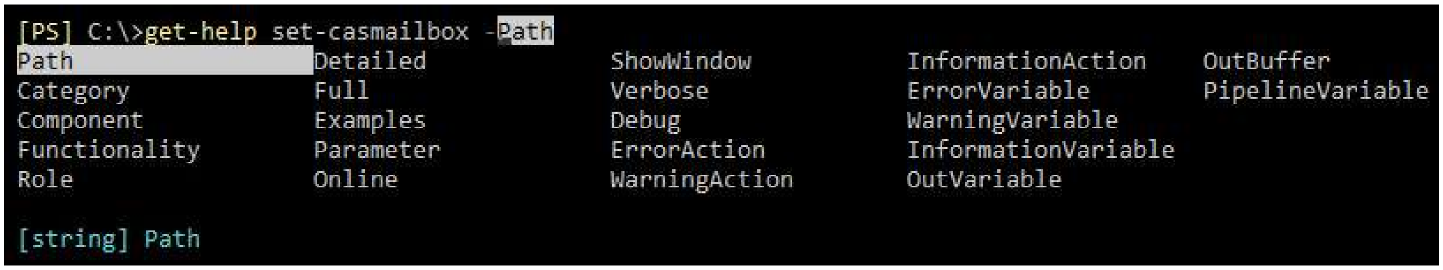
Using a group name is far easier than storing the list of names. The reason to do this is if PowerShell were to cycle through each mailbox to determine access, it would be easier to see if they are a member of a group (single comparison) versus comparing a list of uses (multiple comparisons). The single comparison method is much faster. Storing the group name in a variable is step one in the script build:

```
$POP3AccessGroup = “POP3 Access”
```

What access should be granted or denied depending on group membership?
In this case, the requirement was to allow certain user’s access to POP3 and block all other users from using POP3. Going back to the Get-CASMailbox cmdlet, ‘OWA Enabled’ is an option that can be configured on a mailbox. Get CASMailbox has a set-command for changing settings – Set-CASMailbox. Not knowing what options can be configured, we can try two options. One is to run this:

```
Get-Help Set-CASMailbox -Parameter *
```

Or we can type in the below syntax and then hit the Ctrl and Spacebar at the same time to reveal all parameters:



```
[PS] C:\>get-help set-casmailbox -Path
Path          Detailed      ShowWindow     InformationAction    OutBuffer
Category      Full          Verbose        ErrorVariable        PipelineVariable
Component     Examples      Debug          WarningVariable
Functionality Parameter     ErrorAction    InformationVariable
Role          Online        WarningAction  OutVariable

[string] Path
```

Looking closely at the options we see IMAPEnabled and POPEnabled. Both have two settings - $True and $False. We can set these values for each mailbox on the Exchange 2019 Servers.

With the base command figured out, the next step is to use these cmdlets to set the access for all mailboxes or for some mailboxes based off of group membership. How can group membership be verified?

What commands we can use to query this information in Exchange?
First, get a list of groups a user is in using the ‘memberof ’ property:

```
$MemberOf = (Get-ADUser -Identity (Get-Mailbox Administrator).Alias -Properties
 Memberof).MemberOf
```

What commands can we use to add or remove access to the resource?
As discussed above, one PowerShell cmdlet and two parameters will allow for the changing of access to IMAP or POP:

```
Set-CasMailbox –IMAPEnabled $True
Set-CasMailbox –POPEnabled $True
```

$False would be used to disable access to those not in the AD Group.

Putting it together

Then, using the names of the groups stored in $MemberOf, check for any groups that match our POP3 Access groups and then set POP3access:

```
If ($MemberOf -Like "*POP3 Access*") { Set-CasMailbox $Alias –POPEnabled $True}
```

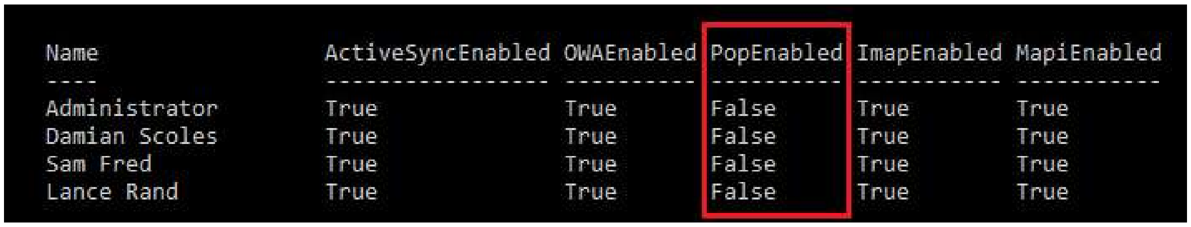
Putting these pieces together, we can create a script like this: (note we are excluding the Discovery mailbox as well)

```
$Mailboxes = Get-Mailbox | where {$_.Name -NotLike "DiscoverySearchMailbox*"}
Foreach ($Mailbox in $Mailboxes) {
# Get all groups a user is a 'member of'
$MemberOf = (Get-ADUser -Identity (Get-Mailbox $Mailbox).Alias -Properties
 MemberOf).MemberOf
# If the user is in the POP Access group enable access and if not ('Else'), disable access
If ($MemberOf -Like "*POP3 Access*") {
Set-CasMailbox $Mailbox –POPEnabled $True
} Else {
Set-CasMailbox $Mailbox –POPEnabled $False
}
}
```

Note: That we are excluding the Discovery mailbox as well and the POP3 Access lines can be compressed into one line where the $True or $False value from $MemberOf could be applied to the PopEnabled value for the mailbox:

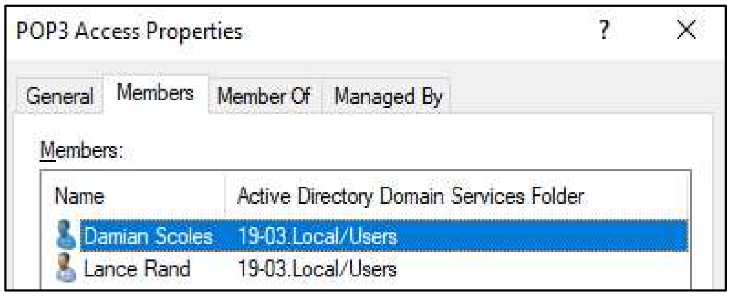
```
Set-CasMailbox $mailbox –POPEnabled ($MemberOf -like "*POP3 Access*")
```

Prior to running the above script, validate who has the setting enabled or disabled: (note what mailboxes are disabled)



```
Name                ActiveSyncEnabled OWAEnabled PopEnabled ImapEnabled MapiEnabled
----                ----------------- ---------- ---------- ----------- -----------
Administrator       True              True       False      True        True
Damian Scoles       True              True       False      True        True
Sam Fred            True              True       False      True        True
Lance Rand          True              True       False      True        True
```

The POP Access group has these members:

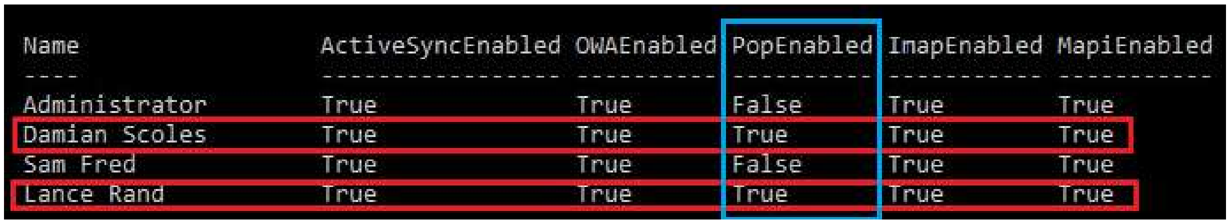


POP3 Access Properties

Members:

Name	Active Directory Domain Services Folder
Damian Scoles	19-03.Local/Users
Lance Rand	19-03.Local/Users

After the script is run, ONLY Damian and Lance would get POP3 access, all others remain disabled ($False).

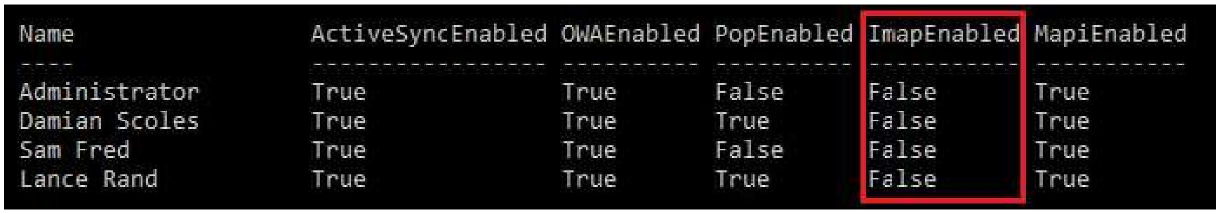


```
Name                 ActiveSyncEnabled OWAEnabled PopEnabled ImapEnabled MapiEnabled
----                 ----------------- ---------- ---------- ----------- -----------
Administrator        True              True       False      True        True
Damian Scoles        True              True       True       True        True
Sam Fred             True              True       False      True        True
Lance Rand           True              True       True       True        True
```

As expected, the results produced are what IT had given as a task. Now, to do the same for IMAP, the script has a couple of lines changes:

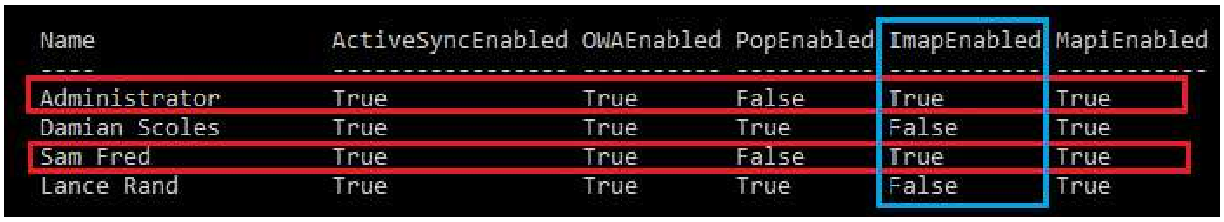
```
       $Mailboxes = Get-Mailbox | Where {$_.Name -NotLike "DiscoverySearchMailbox*"}
       Foreach ($Mailbox in $Mailboxes) {
       # Get all groups a user is a 'member of'
       $MemberOf = (Get-ADUser -Identity (Get-Mailbox $Mailbox).Alias -Properties
        MemberOf).MemberOf
       # If the user is in the IMAP Access group enable access and if not ('Else'), disable access
       If ($MemberOf -like "*IMAP Access*") {
       Set-CasMailbox $Mailbox –IMAPEnabled $True
       } Else {
       Set-CasMailbox $Mailbox –IMAPEnabled $False
       }
       }
```

Before the script:



```
Name                 ActiveSyncEnabled OWAEnabled PopEnabled ImapEnabled MapiEnabled
----                 ----------------- ---------- ---------- ----------- -----------
Administrator        True              True       False      False       True
Damian Scoles        True              True       True       False       True
Sam Fred             True              True       False      False       True
Lance Rand           True              True       True       False       True
```

After the script:



```
Name                 ActiveSyncEnabled OWAEnabled PopEnabled ImapEnabled MapiEnabled
----                 ----------------- ---------- ---------- ----------- -----------
Administrator        True              True       False      True        True
Damian Scoles        True              True       True       False       True
Sam Fred             True              True       False      True        True
Lance Rand           True              True       True       False       True
```

How often we would want to reinforce this action?

Once the script is created, we would then need to decide how and when to enforce these settings. Running the script weekly and daily would be ideal from a restrictive access view. The script can be deployed as a cleanup tool or enforcement tool using the Windows Scheduler, setting up jobs to handle these changes on a weekly or daily basis.

Client Access Rules

Client Access Rules are something new to Exchange Servers. These rules control access to Exchange Services by various client types. Client Access Rules are like Transport Rules for SMTP mail flow. We can create conditions and actions to affect access and authentication with Exchange 2019. Online documentation for this cmdlet can be found here:

https://docs.microsoft.com/en-us/powershell/module/exchange/client-access/new-clientaccessrule

PowerShell Cmdlets

Configuring Client Access Rules is only possible in Exchange PowerShell. There is no option or viewable configuration in the Exhange Admin Center for Exchange 2019. Let's see what PowerShell cmdlets are available for us:

```
Get-Command *ClientAccessRule
```

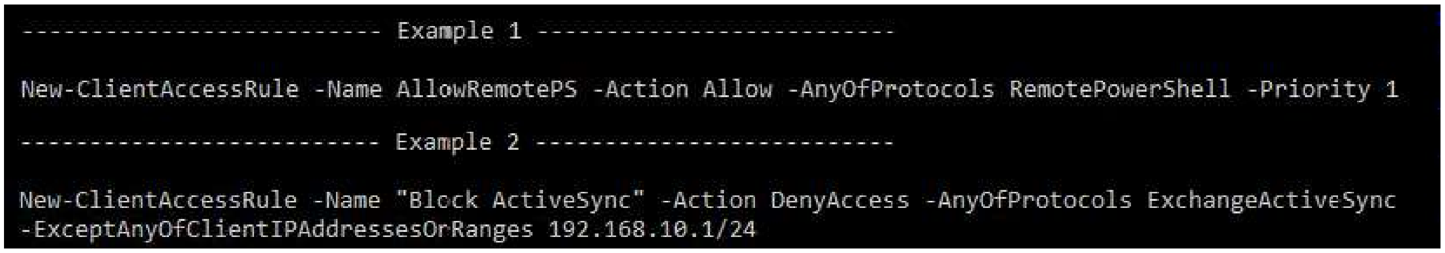
Which reveals these cmdlets for working with Client Access Rules

```
Get-ClientAccessRule            New-ClientAccessRule
Remove-ClientAccessRule         Set-ClientAccessRule
Test-ClientAccessRule
```

If we run Get-ClientAccessRule, the first cmdlet we are listed above, in a Greenfield environment we should find no Client Access Rules configured. This is the default and expected configuration for Exchange 2019. Now, what if we wanted to create new Rules? What can we do and what is their expected usage?

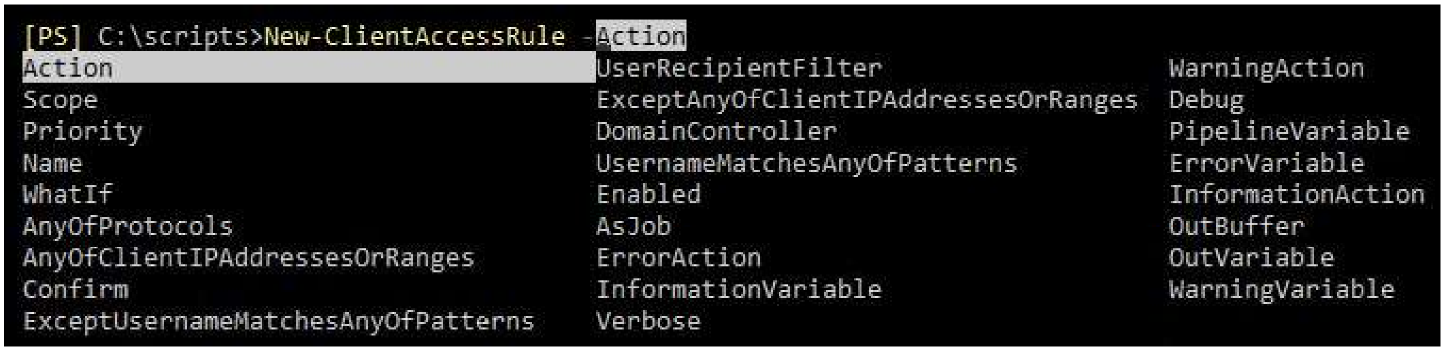
New-ClientAccessRule
Let's review what we have available in Examples and Parameters for this cmdlet:

Examples:



```
-------------------------- Example 1 --------------------------

New-ClientAccessRule -Name AllowRemotePS -Action Allow -AnyOfProtocols RemotePowerShell -Priority 1

-------------------------- Example 2 --------------------------

New-ClientAccessRule -Name "Block ActiveSync" -Action DenyAccess -AnyOfProtocols ExchangeActiveSync
-ExceptAnyOfClientIPAddressesOrRanges 192.168.10.1/24
```

Parameters Available



```
[PS] C:\scripts>New-ClientAccessRule -Action
Action                                 UserRecipientFilter                    WarningAction
Scope                                  ExceptAnyOfClientIPAddressesOrRanges   Debug
Priority                               DomainController                       PipelineVariable
Name                                   UsernameMatchesAnyOfPatterns           ErrorVariable
WhatIf                                 Enabled                                InformationAction
AnyOfProtocols                         AsJob                                  OutBuffer
AnyOfClientIPAddressesOrRanges         ErrorAction                            OutVariable
Confirm                                InformationVariable                    WarningVariable
ExceptUsernameMatchesAnyOfPatterns     Verbose
```

Note: Be careful when creating rules that block PowerShell access. If you create a rule that block PowerShell access to where you cannot connect, you will have to open a support case with Microsoft. The first example use case for the Client Access

Rule cmdlet, shows how to avoid this by allowing PowerShell access with a Priority of 1. Subsequent rules will be overridden by this rule.

Some basics on Client Access Rules and what they can affect:

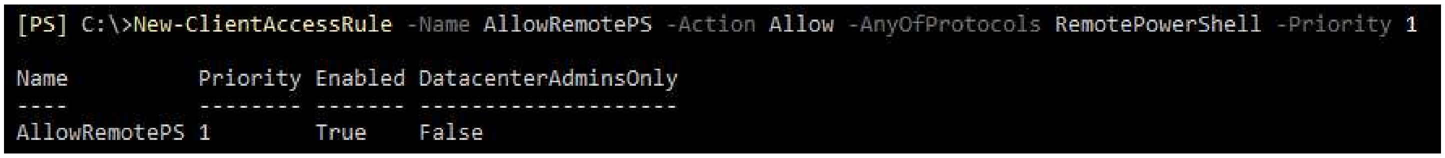
```
Exchange Protocols
ExchangeActiveSync         ExchangeAdminCenter
ExchangeWebServices        IMAP4
OfflineAddressBook         OutlookAnywhere
OutlookWebApp              POP3
PowerShellWebServices      RemotePowerShell
REST
```

```
Authentication Types
AdfsAuthentication                     BasicAuthentication
CertificateBasedAuthentication         NonBasicAuthentication
OAuthAuthentication
```

Client Access Rules can be applied against all allowable protocols and authentication types and can also include exceptions to these protocols or authentication. In addition to protocols and authentication types, we can also build rules based on IP Ranges and Username matches.

Before we begin, we should apply the first example Client Access Rule to our environment, to protect ourselves from any mistakes or typos that may lock us out of PowerShell:

```
New-ClientAccessRule -Name AllowRemotePS -Action Allow -AnyOfProtocols RemotePowerShell
 -Priority 1
```



```
[PS] C:\>New-ClientAccessRule -Name AllowRemotePS -Action Allow -AnyOfProtocols RemotePowerShell -Priority 1

Name          Priority Enabled DatacenterAdminsOnly
----          -------- ------- --------------------
AllowRemotePS 1        True    False
```

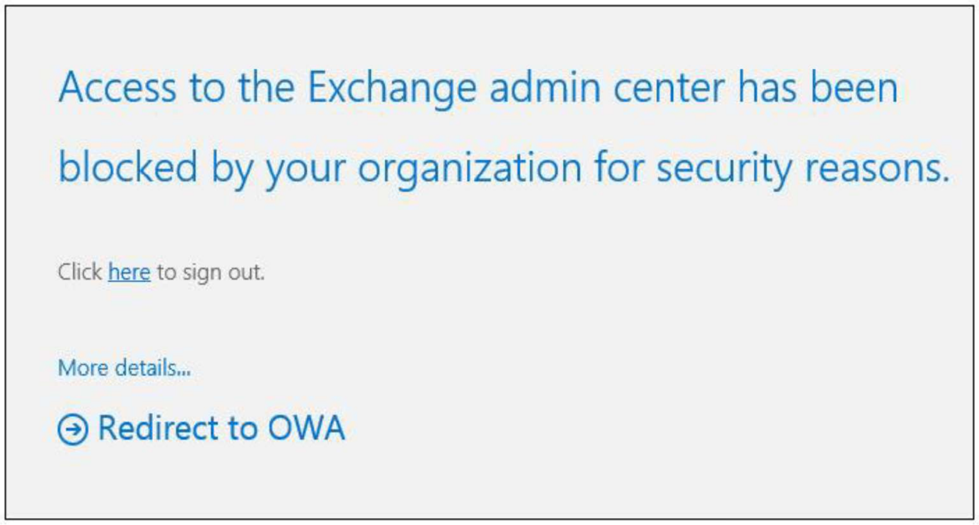
Now, let's explore some sample scenarios and what we can do with Client Access Rules.

Example 1

For this example, we need to restrict the use of the Exchange Admin Center (EAC) to only internal connections. The IT Department has their own subnet for IT computers - 10.0.0.x/24. Further we need to restrict the EAC to only users in IT Department as well, for further security. For the last option, we can use the 'UserRecipientFilter' parameter.

```
New-ClientAccessRule -Name 'EAC Restriction' -Action Deny -AnyProtocols ExchangeAdminCenter
 -ExceptAnyOfClientIPAddressesOrRanges 10.0.0.0/24 -UserRecipientFilter {Department -ne
 'IT'} -Priority 2
```

Notice that we have the IT IP Range Excluded (ExceptAnyOfClientIPAddressesOrRanges) and excluded the IT Department from the Deny Access (UserRecipientFilter). Lastly the Rule priority was set to 2, being preceded by the Remote PS rule.
If we check two users to see what their experience is, we can validate whether or not we have the rules correct. A blocked user will see this:



Access to the Exchange admin center has been blocked by your organization for security reasons.

Click here to sign out.

More details...

Redirect to OWA

Note: When the first Client Access Rules are created, it can take time for them to go into effect. From Microsoft: "To improve overall performance, Client Access Rules use a cache, which means changes to rules don't immediately take effect. The first rule that you create in your organization can take up to 24 hours to take effect. After that, modifying, adding, or removing rules can take up to one hour to take effect."

Source: https://docs.microsoft.com/en-us/exchange/clients/client-access-rules/client-access-rules?view=exchserver-2019

Example 2
For this scenario we want to restrict all protocols to internal company subnets which comes as a directive from the security department. The only exception to this rule will be for a select group of personnel that have CustomAttribute8 set to 'MobileAllowed'.

In order to perform these tasks, we will need two separate Client Access Rules. Rules that will be needed:

```
New-ClientAccessRule -Name 'Corp EWAS Exception' -Action Allow -AnyOfProtocols
 ExchangeActiveSync -UsernameMatchesAnyOfPatterns {CustomAttribute8 -eq 'MobileAccess'}
 -Priority 2
```

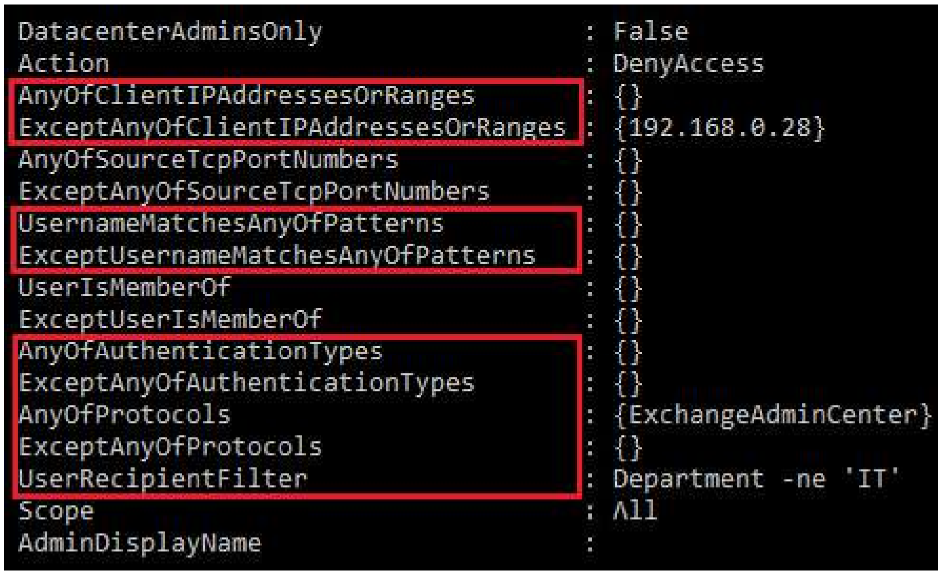
And

```
New-ClientAccessRule -Name 'Corp Deny CAR' -Action Deny -AnyOfClientIPAddressesOrRanges
 10.0.0.0/16 -AnyOfProtocols ExchangeActiveSync, ExchangeAdminCenter, ExchangeWebServices,
 IMAP4, OfflineAddressBook, OutlookAnywhere, OutlookWebApp, POP3, PowerShellWebServices,
 RemotePowerShell, REST -UsernameMatchesAnyOfPatterns {CustomAttribute8 -ne 'MobileAccess'}
 -Priority 3
```

The Client Access Rules are in order of smallest match first, which in this case is the CustomAttribute. If a user account does not match that attribute, they will then process the next rule which will then restrict them to internal use of Exchange 2019.

Client Access Rule Verification

Now that we've create some of the rules above, let's review the properties of the Client Access Rules and what we can see with PowerShell. If we review the 'EAC Restriction' rule, created in Example 1, we can see the configurable properties:



```
DatacenterAdminsOnly                    : False
Action                                  : DenyAccess
AnyOfClientIPAddressesOrRanges          : {}
ExceptAnyOfClientIPAddressesOrRanges    : {192.168.0.28}
AnyOfSourceTcpPortNumbers               : {}
ExceptAnyOfSourceTcpPortNumbers         : {}
UsernameMatchesAnyOfPatterns            : {}
ExceptUsernameMatchesAnyOfPatterns      : {}
UserIsMemberOf                          : {}
ExceptUserIsMemberOf                    : {}
AnyOfAuthenticationTypes                : {}
ExceptAnyOfAuthenticationTypes          : {}
AnyOfProtocols                          : {ExchangeAdminCenter}
ExceptAnyOfProtocols                    : {}
UserRecipientFilter                     : Department -ne 'IT'
Scope                                   : All
AdminDisplayName                        :
```

It also looks like there are other configurable values, like these:

```
      AnyOfSourceTcpPortNumbers           ExceptAnyOfSourceTcpPortNumbers
      UserIsMemberOf                      ExceptUserIsMemberOf.
```

However, there are not parameters on the New-ClientAccessRule for this. These parameters might be there for future changes to the Client Access Rule cmdlets or may be reserved for Microsoft internal use. Either way, we cannot gain any insight into these values at this time.