```
<#
  .SYNOPSIS
    This files disables mailboxes for people who are out of security
    compliance training requirements. AD is not affected.

  .DESCRIPTION
    This file connects to a SOLE database to get a list of people who are
    not in compliance with security awareness training. If they are not in
    compliance with training, their email access (OWA, MAPI, ActiveSync)
    will be disabled. This file does not affect AD.

    The database is maintained by the SOLE group, and is populated at 7:07 am every day.
    This file runs as a scheduled task at 7:15 am every day.

  .PARAMETER MaximumDisables
    This is a safety parameter to make sure too many accounts aren't disabled.

  .PARAMETER DisableAccounts
    Specifies whether accounts that are actually disabled

  .PARAMETER SQLServer
    The ip address of the sql server

  .PARAMETER DBName
    DB name to query

  .PARAMETER DBTableName
    Table name to query

  .NOTES
    Author: Jeff Brusoe
    Last Updated by: Jeff Brusoe
    Last Updated: September 1, 2021

    Version Updates:
      - March 25, 2019
        - Cleaned up file output
        - Modified to work with GitHub directory
        - Added new common code parameters
        - Removed old commmon code to initialize environment
        - Added new common code to initialize environment
        - Used SQL Module for SQL function calls

      - October 14, 2019
        - Changed DB calls to use parameters

      - March 25, 2020
        - Switched to using Invoke-SqlCmd call instead of .NET methods

      - September 1, 2020
        - Moved to updated HSC modules
        - Changed to new Get-HSCSPOExclusionList function with
          the SP PNP cmdlets.

      - December 22, 2020
        - Made changes to conform with HSC PowerShell best practices and style

      - September 1, 2021
        - Moved SQL connection string code to Get-HSCSQLConnectionString
        - Added parameter validation
        - Minor code cleanup
#>

[CmdletBinding()]
param (
  [ValidateNotNullOrEmpty()]
  [string]$DBName = "SecurityAwareness",

  [ValidateNotNullOrEmpty()]
  [string]$DBTableName = "DisabledUser",

  [ValidateRange(0,50)]
  [int]$MaximumDisables = 20,

  [switch]$DisableAccounts
)

try {
  Write-Output "Configuring environment"

  Set-HSCEnvironment -ErrorAction Stop
  Connect-HSCExchangeOnlineV1 -ErrorAction Stop

  $DoNotDisable = Get-HSCSPOExclusionList -ErrorAction Stop

  Write-Output "`n`n----------------------------------------------------"
  Write-Output "Processing will not be done on these accounts:"
  $DoNotDisable
  Write-Output "----------------------------------------------------`n`n"

  $Count = 0 #Keeps track of current user for visual output
  $DisableCount = 0 #Keeps track of new accounts being disabled.
}
catch {
  Write-Warning "There was an error configuring the environment. Program is exiting."
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

try {
  Write-Output "Decrypting SQL Password & generating connection string"

    $SQLPassword = Get-HSCSQLPassword -SOLEDB -Verbose -ErrorAction Stop

  $GetHSCSQLConnectionStringParams = @{
    Password = $SQLPassword
    Database = $DBName
    ErrorAction = "Stop"

  }

    $SQLConnectionString = Get-HSCSQLConnectionString @GetHSCSQLConnectionStringParams
}
catch {
  Write-Warning "Unable to decrypt SQL Passowrd"
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

```

```powershell
try {
  Write-Output "Querying users from SOLE DB to be disabled"

  $Query = "select * from $DBTableName"
  Write-Output "Query: $Query"

  $InvokeSQLCmdParams = @{
    Query = $Query
    ConnectionString = $SQLConnectionString
    ErrorAction = "Stop"
  }

  $UsersToDisable = Invoke-SQLCmd @InvokeSQLCmdParams
}
catch {
  Write-Warning "There was an error reading the SOLE database. Program is exiting."
  Invoke-HSCExitCommand -ErrorCount $Error.Count
}

$UsersToDisableCount = ($UsersToDisable | Measure-Object).Count
Write-Output "Total Number of Users to Disable: $UsersToDisableCount"

foreach ($UserToDisable in $UsersToDisable)
{
  Write-Output "User number: $Count"
  Write-Output "Disable Count: $DisableCount"
  Write-Output $("Error Count: " + $Error.Count)

  $SQLUserName = $UserToDisable.Username.Trim()
  Write-Output "User to be disabled: $SQLUserName"

  if ($DisableCount -lt $MaximumDisables)
  {
        if ($DoNotDisable -notcontains $SQLUserName)
        {
      #This if statement ensures that nobody in the exclusion list will be disabled.
      Write-Output "User is not in exclusion list"

      if($DisableAccounts)
      {
        try
        {
          $MbxExist = $false

          $MbxExist = [bool](Get-Mailbox $SQLUserName -ErrorAction SilentlyContinue)

          if ($MbxExist)
          {
            $Mbx = Get-Mailbox $SQLUserName -ErrorAction Stop

            $CasMailboxInfoFound = $true
            try {
              $CasMbx = $Mbx | Get-CasMailbox -ErrorAction Stop
              Write-Verbose "Successfully pulled CasMailbox information"
            }
            catch
            {
              #I'm not sure why certain accounts need this instead of the previous
              #code, but there are a small number that do.
              try {
                $CasMbx = Get-CasMailbox -Identity $Mbx.PrimarySMTPAddress -ErrorAction Stop
                Write-Output "Successfully pulled CasMailbox information with PrimarySMTPAddress"
              }
              catch {
                Write-Warning "Unable to pull CasMailbox information for this user"
                $CasMailboxInfoFound = $false
              }
            }

            if (!$CasMbx.OWAEnabled -AND
              !$CasMbx.MAPIEnabled -AND
              !$CasMbx.ActiveSyncEnabled -AND
              $CasMailboxInfoFound) {
              Write-Output "Mailbox is already disabled"
            }
            else
            {
              if (($Mbx | Measure-Object).Count -eq 1)
              {
                $SetCasMailboxParams =@{
                  OWAEnabled = $false
                  MAPIEnabled = $false
                  ActiveSyncEnabled = $false
                  ErrorAction = "Stop"
                }

                try {
                  $Mbx | Set-CasMailbox @SetCasMailboxParams
                  Write-Output "User has been disabled: $SQLUserName"
                }
                catch {
                  Write-Warning "Unable to disable user"
                }

                $DisableCount++
              }
              else {
                Write-Warning "Unable to find one unique mailbox for this user."
                Write-Warning "User should be disabled, but isn't due to this issue."
              }
            }
          }
          else {
            Write-Output "No mailbox for this user: $user"
          }
        }
        catch {
          Write-Warning "An error happened trying to disable mailbox"
        }
      }
      else {
        Write-Warning "User should be disabled"
        Write-Warning $("DisableAccounts: $DisableAccounts")
      }
        }
```

```
    else {
      Write-Output "The user is in the exclusion list and will not be disabled."
    }
  }
    else {
    Write-Warning "The maximum number of disables has been reached."
    Write-Warning "No further accounts will be disabled."
    }

  $Count++

  Write-Output "****************************"
}

Write-Output "`nProcessing has been completed."
Write-Output "Accounts Processed: $Count"
Write-Output "New Accounts Disabled: $DisableCount"

Invoke-HSCExitCommand -ErrorCount $Error.Count
```