# Ensure the Active Directory module is available for AD queries

Import-Module ActiveDirectory -ErrorAction Stop

# Define the output file path

$outputFile = "C:\Users\Public\non\_admin\_users.csv"

# Helper function to check if a user is in an administrative group

function Is-AdminUser {

param (

[string]$UserName,

[string[]]$AdminGroups

)

foreach ($group in $AdminGroups) {

if (Get-ADGroupMember -Identity $group -Recursive | Where-Object { $\_.SamAccountName -eq $UserName }) {

return $true

}

}

return $false

}

# Retrieve Active Directory Users

Write-Host "Retrieving Active Directory users..."

try {

$adminGroups = @("Administrators", "Domain Admins", "Enterprise Admins") # Add other groups if necessary

$adUsers = Get-ADUser -Filter \* -Property SamAccountName, Name, UserPrincipalName, Enabled |

Where-Object {

-not (Is-AdminUser -UserName $\_.SamAccountName -AdminGroups $adminGroups)

} |

Select-Object SamAccountName, Name, UserPrincipalName, Enabled

Write-Host "Successfully retrieved non-administrative Active Directory users."

} catch {

Write-Warning "Failed to retrieve Active Directory users. Make sure the AD module is installed and you have the required permissions."

$adUsers = @()

}

# Retrieve Local Machine Users

Write-Host "Retrieving local machine users..."

try {

# Exclude users in the local Administrators group

$adminGroupUsers = (Get-LocalGroupMember -Group "Administrators").Name

$localUsers = Get-LocalUser |

Where-Object {

$\_.Enabled -eq $true -and

$\_.Name -notin $adminGroupUsers

} |

Select-Object Name, Enabled

Write-Host "Successfully retrieved non-administrative local machine users."

} catch {

Write-Warning "Failed to retrieve local machine users. Make sure the script is running with administrative privileges."

$localUsers = @()

}

# Combine AD and local users into a single array

$allUsers = @()

if ($adUsers) {

$allUsers += $adUsers | ForEach-Object {

[PSCustomObject]@{

Source = "Active Directory"

Username = $\_.SamAccountName

Name = $\_.Name

UserPrincipal = $\_.UserPrincipalName

Enabled = $\_.Enabled

}

}

}

if ($localUsers) {

$allUsers += $localUsers | ForEach-Object {

[PSCustomObject]@{

Source = "Local Machine"

Username = $\_.Name

Name = $\_.Name

UserPrincipal = "N/A"

Enabled = $\_.Enabled

}

}

}

# Export the combined user list to CSV

if ($allUsers.Count -gt 0) {

$allUsers | Export-Csv -Path $outputFile -NoTypeInformation -Force

Write-Host "All non-administrative users have been exported to $outputFile"

} else {

Write-Warning "No non-administrative users found to export."

}

# Input CSV file with the list of users

$inputFile = "C:\Users\Public\non\_admin\_users.csv"

# Output file to store user passwords securely

$passwordFile = "C:\Users\Public\user\_passwords.txt"

# Generate a random secure password

function Generate-RandomPassword {

param ([int]$Length = 12)

$allowedChars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*()-\_=+'

-join ((1..$Length) | ForEach-Object { $allowedChars | Get-Random })

}

# Import the list of users from the CSV

$users = Import-Csv -Path $inputFile

# Array to store user-password mapping

$passwordStore = @()

# Loop through each user and reset their password

foreach ($user in $users) {

try {

# Generate a random password

$newPassword = Generate-RandomPassword

# Check the source of the user and reset the password

if ($user.Source -eq "Active Directory") {

# Reset Active Directory user password

Set-ADAccountPassword -Identity $user.Username -Reset -NewPassword (ConvertTo-SecureString $newPassword -AsPlainText -Force)

Set-ADUser -Identity $user.Username -PasswordNeverExpires $false -ChangePasswordAtLogon $true

Write-Host "Password reset for AD user: $($user.Username)"

} elseif ($user.Source -eq "Local Machine") {

# Reset local user password

$localUser = Get-LocalUser -Name $user.Username

if ($localUser) {

$localUser | Set-LocalUser -Password (ConvertTo-SecureString $newPassword -AsPlainText -Force)

Write-Host "Password reset for local user: $($user.Username)"

}

}

# Store the username and new password in the array

$passwordStore += [PSCustomObject]@{

Username = $user.Username

Password = $newPassword

}

} catch {

Write-Warning "Failed to reset password for user: $($user.Username) - $\_"

}

}

# Export the username-password mapping to a secure file

$passwordStore | Out-File -FilePath $passwordFile -Encoding UTF8

Write-Host "Passwords stored in $passwordFile"