MS14-025: Vulnerability in Group Policy Preferences could allow elevation of privilege: May 13, 2014

Archived: 2026-04-05 18:16:07 UTC

INTRODUCTION

Microsoft has released security bulletin MS14-025. To learn more about this security bulletin:

- Home users:

  https://www.microsoft.com/security/pc-security/updates.aspxSkip the details: Download the updates for your home computer or laptop from the Microsoft Update website now:

  https://update.microsoft.com/microsoftupdate/

- IT professionals:

  https://technet.microsoft.com/security/bulletin/MS14-025

How to obtain help and support for this security update

More Information

Known issues and more information about this security update

The following articles contain more information about this security update as it relates to individual product versions. The articles may contain known issue information. If this is the case, the known issue is listed under each article link.

- 2928120

  MS14-025: Description of the security update for Windows Remote Server Administration Tools for systems that have update 2919355 installed: May 13, 2014

- 2961899

  MS14-025: Description of the security update for Windows Remote Server Administration Tools for systems that do not have update 2919355 installed: May 13, 2014

Group Policy Preferences

Overview

Some Group Policy Preferences can store a password. This functionality is being removed because the password was stored insecurely. This article describes the user interface changes and any available workarounds.

The following Group Policy Preferences will no longer allow user names and passwords to be saved:

- Drive Maps
- Local Users and Groups
- Scheduled Tasks
- Services
- Data Sources

This will affect the behavior of any existing Group Policy Objects (GPOs) in your environment that rely on passwords that are contained in these preferences. It will also prevent creating new Group Policy Preferences by using this functionality.

For Drive Maps, Local Users and Groups, and Services, you may be able to achieve similar goals through other, more secure functionality in Windows.

For Scheduled Tasks and Data Sources, you will be unable to achieve the same goals that were available through the nonsecure functionality of Group Policy Preferences passwords.

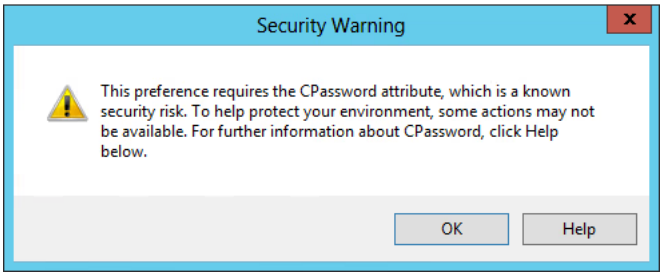
Scenarios

The following Group Policy Preferences are affected by this change. Each preference is covered briefly and then in more detail. Additionally, workarounds are provided that enable you to perform the same tasks.

Affected preference	Applies to user	Applies to computer
Local user management	Yes	Yes
Mapped drives	Yes	No
Services	No	Yes
Scheduled tasks (up-level)	Yes	Yes
Scheduled tasks (down-level)	Yes	Yes
Immediate tasks (up-level)	Yes	Yes
Immediate tasks (down-level)	Yes	Yes
Data sources	Yes	Yes

Summary of changes

- Password fields in all affected preferences are disabled. Administrators cannot create new preferences by using these password fields.
- The username field is disabled in some preferences.
- Existing preferences that contain a password cannot be updated. They can only be deleted or disabled, as appropriate for the specific preference.
- The behavior for Delete and Disable actions have not changed for the preferences.
- When an administrator opens any preference that contains the CPassword attribute, the administrator receives the following warning dialog box to inform him or her of the recent deprecation. Attempts to save changes to new or existing preferences that require the CPassword attribute will trigger the same dialog box. Only Delete and Disable actions will not trigger warning dialog boxes.



Scenario 1: Local user management

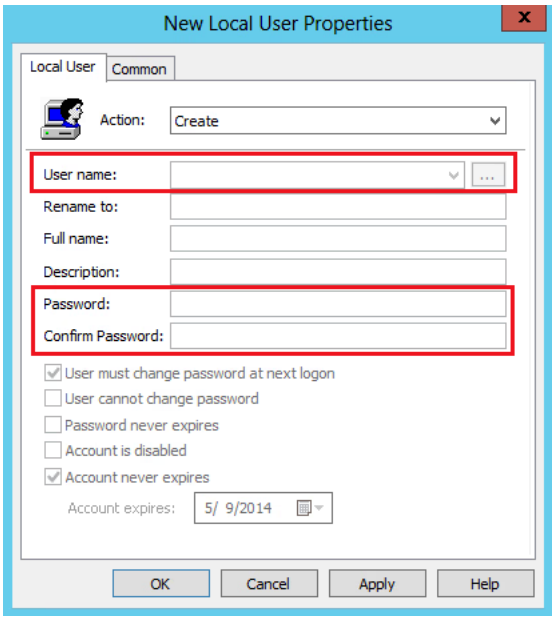
The Local User Management preference is frequently used to create local administrators who have a known password on a computer. This feature is not secure because of the way that Group Policy Preferences stores passwords. Therefore, this functionality is no longer available. The following preferences are affected:

- Computer Configuration -> Control Panel Settings -> Local Users and Groups-> New-> Local User
- User Configuration -> Control Panel Settings -> Local Users and Groups-> New-> Local User

Important changes

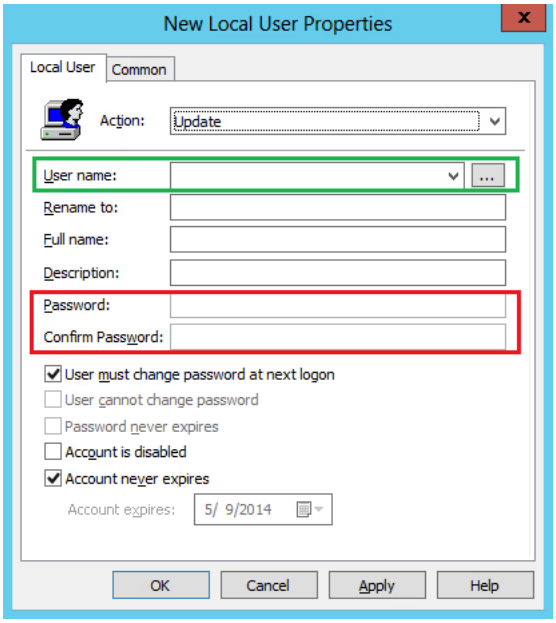
Action: Create or Replace

- The **User name**, **Password**, and **Confirm Password** fields are disabled.
- The warning dialog box appears when the administrator opens or tries to save any changes to an existing preference that contains a password.



Action: Update

- The **Password** and **Confirm Password** fields are disabled.
- The warning dialog box appears when the administrator opens or tries to save any changes to an existing preference that contains a password.



Action: Delete

- No change in behavior

Workarounds

For those who previously relied on the Group Policy Preference for setting local administrator passwords, the following script is provided as a secure alternative to CPassword. Copy and save the contents to a new Windows PowerShell file, and then run the script as indicated in its .EXAMPLE section.

Microsoft provides programming examples for illustration only, without warranty either expressed or implied. This includes, but is not limited to, the implied warranties of merchantability or fitness for a particular purpose. This article assumes that you are familiar with the programming language that is being demonstrated and with the tools that are used to create and to debug procedures. Microsoft support engineers can help explain the functionality of a particular procedure. However, they will not modify these examples to provide added functionality or construct procedures to meet your specific requirements.

function Invoke-PasswordRoll{<#.SYNOPSIS

This script can be used to set the local account passwords on remote machines to random passwords. The username/password/server combination will be saved in a CSV file.The account passwords stored in the CSV file can be encrypted using a password of the administrators choosing to ensure clear-text account passwords aren't written to disk.The encrypted passwords can be decrypted using another function in this file: ConvertTo-CleartextPassword

Function: Invoke-PasswordRollAuthor: MicrosoftVersion: 1.0

.DESCRIPTION

This script can be used to set the local account passwords on remote machines to random passwords. The username/password/server combination will be saved in a CSV file.The account passwords stored in the CSV file can be encrypted using a password of the administrators choosing to ensure clear-text account passwords aren't written to disk.The encrypted passwords can be decrypted using another function in this file: ConvertTo-CleartextPassword

.PARAMETER ComputerName

An array of computers to run the script against using PowerShell remoting.

.PARAMETER LocalAccounts

An array of local accounts whose password should be changed.

.PARAMETER TsvFileName

The file to output the username/password/server combinations to.

.PARAMETER EncryptionKey

A password to encrypt the TSV file with. Uses AES encryption. Only the passwords stored in the TSV file will be encrypted, the username and servername will be clear-text.

.PARAMETER PasswordLength

The length of the passwords which will be randomly generated for local accounts.

.PARAMETER NoEncryption

Do not encrypt the account passwords stored in the TSV file. This will result in clear-text passwords being written to disk.

.EXAMPLE

. .\Invoke-PasswordRoll.ps1 #Loads the functions in this script fileInvoke-PasswordRoll -ComputerName (Get-Content computerlist.txt) -LocalAccounts @("administrator","CustomLocalAdmin") -TsvFileName "LocalAdminCredentials.tsv" -EncryptionKey "Password1"

Connects to all the computers stored in the file "computerlist.txt". If the local account "administrator" and/or "CustomLocalAdmin" are present on the system, their password is changedto a randomly generated password of length 20 (the default). The username/password/server combinations are stored in LocalAdminCredentials.tsv, and the account passwords are AES encrypted using the password "Password1".

.EXAMPLE

. .\Invoke-PasswordRoll.ps1 #Loads the functions in this script fileInvoke-PasswordRoll -ComputerName (Get-Content computerlist.txt) -LocalAccounts @("administrator") -TsvFileName "LocalAdminCredentials.tsv" -NoEncryption -PasswordLength 40

Connects to all the computers stored in the file "computerlist.txt". If the local account "administrator" is present on the system, its password is changed to a random generatedpassword of length 40. The username/password/server combinations are stored in LocalAdminCredentials.tsv unencrypted.

.NOTESRequirements: -PowerShellv2 or above must be installed-PowerShell remoting must be enabled on all systems the script will be run against

Script behavior:-If a local account is present on the system, but not specified in the LocalAccounts parameter, the script will write a warning to the screen to alert you to the presence of this local account. The script will continue running when this

happens.-If a local account is specified in the LocalAccounts parameter, but the account does not exist on the computer, nothing will happen (an account will NOT be created).-The function ConvertTo-CleartextPassword, contained in this file, can be used to decrypt passwords that are stored encrypted in the TSV file.-If a server specified in ComputerName cannot be connected to, PowerShell will output an error message.-Microsoft advises companies to regularly roll all local and domain account passwords.

#> [CmdletBinding(DefaultParameterSetName="Encryption")] Param([Parameter(Mandatory=$true)] [String[]] $ComputerName,

[Parameter(Mandatory=$true)] [String[]] $LocalAccounts,

[Parameter(Mandatory=$true)] [String] $TsvFileName,

[Parameter(ParameterSetName="Encryption", Mandatory=$true)] [String] $EncryptionKey,

[Parameter()] [ValidateRange(20,120)] [Int] $PasswordLength = 20,

[Parameter(ParameterSetName="NoEncryption", Mandatory=$true)] [Switch] $NoEncryption)

#Load any needed .net classes Add-Type -AssemblyName "System.Web" -ErrorAction Stop

#This is the scriptblock that will be executed on every computer specified in ComputerName $RemoteRollScript = { Param([Parameter(Mandatory=$true, Position=1)] [String[]] $Passwords,

[Parameter(Mandatory=$true, Position=2)] [String[]] $LocalAccounts,

#This is here so I can record what the server name that the script connected to was, sometimes the DNS records get messed up, it can be nice to have this. [Parameter(Mandatory=$true, Position=3)] [String] $TargettedServerName)

$LocalUsers = Get-WmiObject Win32_UserAccount -Filter "LocalAccount=true" | Foreach {$_.Name}

#Check if the computer has any local user accounts whose passwords are not going to be rolled by this script foreach ($User in $LocalUsers) { if ($LocalAccounts -inotcontains $User) { Write-Warning "Server: '$($TargettedServerName)' has a local account '$($User)' whos password is NOT being changed by this script" } }

#For every local account specified that exists on this server, change the password $PasswordIndex = 0 foreach ($LocalAdmin in $LocalAccounts) { $Password = $Passwords[$PasswordIndex]

if ($LocalUsers -icontains $LocalAdmin) { try { $objUser = [ADSI]"WinNT://localhost/$($LocalAdmin), user" $objUser.psbase.Invoke("SetPassword", $Password)

$Properties = @{ TargettedServerName = $TargettedServerName Username = $LocalAdmin Password = $Password RealServerName = $env:computername }

$ReturnData = New-Object PSObject -Property $Properties Write-Output $ReturnData } catch { Write-Error "Error changing password for user:$($LocalAdmin) on server:$($TargettedServerName)" } }

$PasswordIndex++ } }

#Generate the password on the client running this script, not on the remote machine. System.Web.Security isn't available in the .NET Client profile. Making this call # on the client running the script ensures only 1 computer needs the full .NET runtime installed (as opposed to every system having the password rolled). function Create-RandomPassword { Param([Parameter(Mandatory=$true)] [ValidateRange(20,120)] [Int] $PasswordLength)

$Password = [System.Web.Security.Membership]::GeneratePassword($PasswordLength, $PasswordLength / 4)

#This should never fail, but I'm putting a sanity check here anyways if ($Password.Length -ne $PasswordLength) { throw new Exception("Password returned by GeneratePassword is not the same length as required. Required length: $($PasswordLength). Generated length: $($Password.Length)") }

return $Password }

#Main functionality - Generate a password and remote in to machines to change the password of local accounts specified if ($PsCmdlet.ParameterSetName -ieq "Encryption") { try { $Sha256 = new-object System.Security.Cryptography.SHA256CryptoServiceProvider $SecureStringKey = $Sha256.ComputeHash([System.Text.UnicodeEncoding]::Unicode.GetBytes($EncryptionKey)) } catch { Write-Error "Error creating TSV encryption key" -ErrorAction Stop } }

foreach ($Computer in $ComputerName) { #Need to generate 1 password for each account that could be changed $Passwords = @() for ($i = 0; $i -lt $LocalAccounts.Length; $i++) { $Passwords += Create-RandomPassword -PasswordLength $PasswordLength }

Write-Output "Connecting to server '$($Computer)' to roll specified local admin passwords" $Result = Invoke-Command -ScriptBlock $RemoteRollScript -ArgumentList @($Passwords, $LocalAccounts, $Computer) -ComputerName $Computer #If encryption is being used, encrypt the password with the user supplied key prior to writing to disk if ($Result -ne $null) { if ($PsCmdlet.ParameterSetName -ieq "NoEncryption") { $Result | Select-Object Username,Password,TargettedServerName,RealServerName | Export-Csv -Append -Path $TsvFileName -NoTypeInformation } else { #Filters out $null entries returned $Result = $Result | Select-Object Username,Password,TargettedServerName,RealServerName

foreach ($Record in $Result) { $PasswordSecureString = ConvertTo-SecureString -AsPlainText -Force -String ($Record.Password) $Record | Add-Member -MemberType NoteProperty -Name EncryptedPassword -Value (ConvertFrom-SecureString -Key $SecureStringKey -SecureString $PasswordSecureString) $Record.PSObject.Properties.Remove("Password") $Record | Select-Object Username,EncryptedPassword,TargettedServerName,RealServerName | Export-Csv -Append -Path $TsvFileName -NoTypeInformation } } } }}

function ConvertTo-CleartextPassword{<#.SYNOPSISThis function can be used to decrypt passwords that were stored encrypted by the function Invoke-PasswordRoll.

Function: ConvertTo-CleartextPasswordAuthor: MicrosoftVersion: 1.0

.DESCRIPTIONThis function can be used to decrypt passwords that were stored encrypted by the function Invoke-PasswordRoll.

.PARAMETER EncryptedPassword

The encrypted password that was stored in a TSV file.

.PARAMETER EncryptionKey

The password used to do the encryption.

.EXAMPLE

. .\Invoke-PasswordRoll.ps1 #Loads the functions in this script fileConvertTo-CleartextPassword -EncryptionKey "Password1" -EncryptedPassword 76492d1116743f0423413b16050a5345MgB8AGcAZgBaAHUAaQBwADAAQgB2AGgAcABNADMASwBaAFoAQQBzADEAeABjAEEAPQA9AHw

Decrypts the encrypted password which was stored in the TSV file.

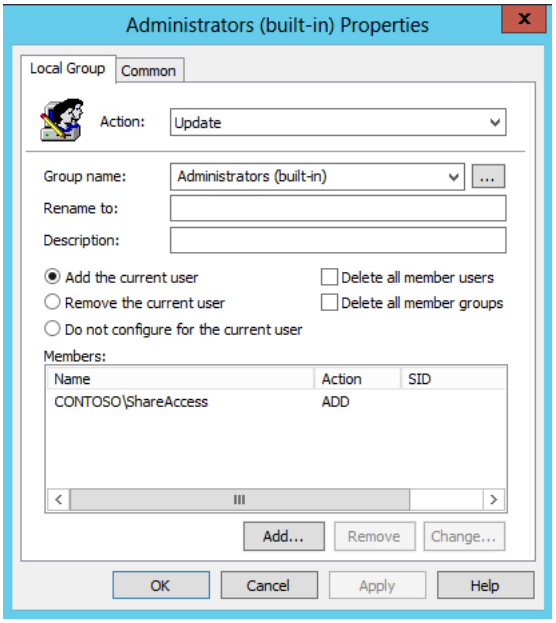
#> Param([Parameter(Mandatory=$true)] [String] $EncryptedPassword,

[Parameter(Mandatory=$true)] [String] $EncryptionKey)

$Sha256 = new-object System.Security.Cryptography.SHA256CryptoServiceProvider $SecureStringKey = $Sha256.ComputeHash([System.Text.UnicodeEncoding]::Unicode.GetBytes($EncryptionKey))

[SecureString]$SecureStringPassword = ConvertTo-SecureString -String $EncryptedPassword -Key $SecureStringKey Write-Output ([System.Runtime.InteropServices.Marshal]::PtrToStringAuto([System.Runtime.InteropServices.Marshal]::SecureStringToCoTaskMemUnicode($Secure

Administrators can add local administrator accounts to computers by creating an Active Directory group and adding it to the local Administrators group through Group Policy Preferences -> Local Group. This action does not cache credentials. The dialog box resembles the following. This workaround does require a connection to Active Directory Domain Services when the user is logged on by using these credentials.



Scenario 2: Mapped drives

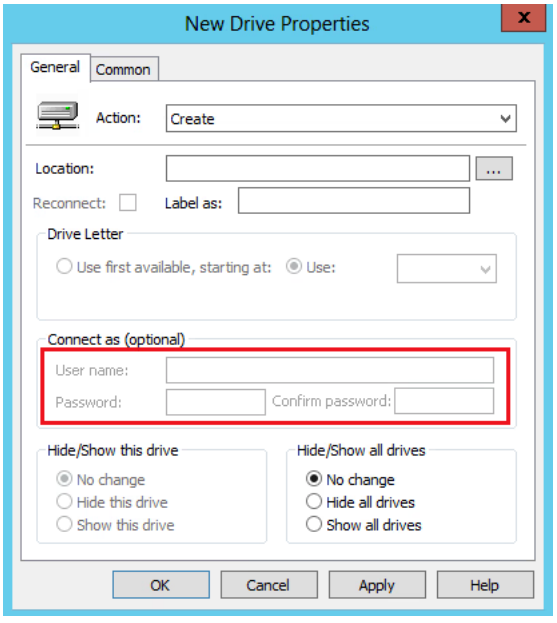
Administrators use drive maps to allocate network locations to users. The password protection feature is used to make sure of authorized access to the drive. The following preferences are affected:

- User Configuration -> Windows Settings -> Drive Maps -> New -> Mapped Drive

Important changes

Action: Create, Update, or Replace

- The **User name**, **Password**, and **Confirm password** fields are disabled.



Action: Delete

- No change in behavior

Workarounds

Instead of using the password method for authentication, you can use Windows Explorer to manage share permissions and allocate rights to users. You can use Active Directory objects to control permissions to the folder.

Scenario 3: Services

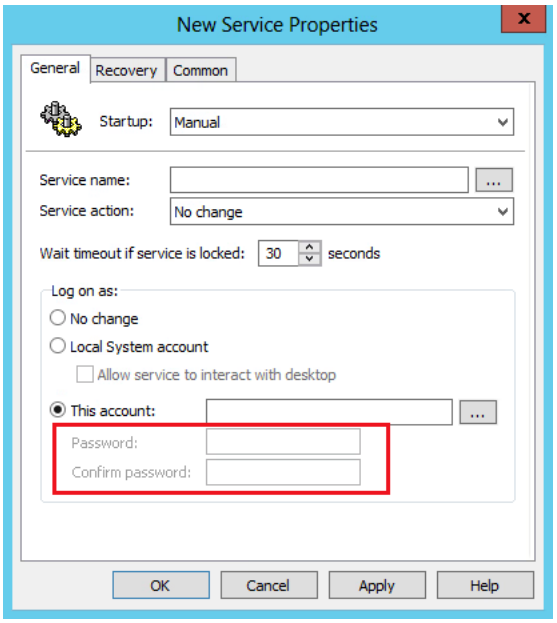
You can use the Services preference to change service properties in such a way that they run in a context other than their original security context. The following preferences are affected:

- Computer Configuration -> Control Panel Settings -> Services -> New -> Service

Important changes

Startup: No Change, Automatic, or Manual

- The **Password** and **Confirm password** fields are disabled.
- The administrator can use only built-in accounts.

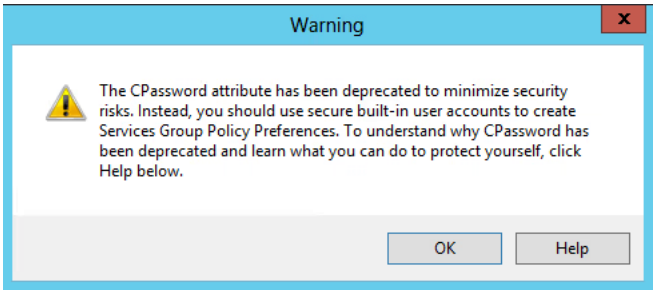


Startup: Disable

- No change in behavior

New dialog box

- Administrators who try to use non-built-in users for This account" receive the following warning:



Workarounds

Services can still run as a local system account. Service permissions can be altered as documented in the following article in the Microsoft Knowledge Base:

256345 How to Configure Group Policy settings to set security for system services

Note If the service that you want to configure is not present, you must configure the settings on a computer that has the service running.

Scenario 4: Scheduled and immediate tasks (up-level)

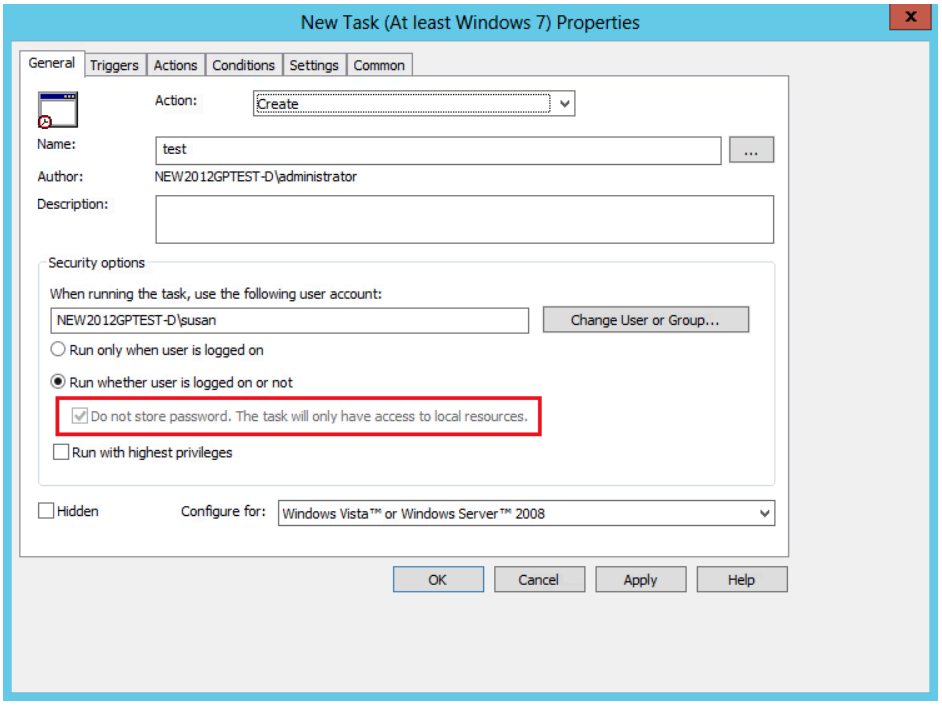
These are used to run scheduled tasks in a specific security context. The ability to store credentials for scheduled tasks to run as an arbitrary user when that user is not logged on is no longer available. The following preferences are affected. (Be aware that on some platforms, "At least Windows 7" is replaced with "Windows Vista and later.")

- Computer Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Scheduled Task (At least Windows 7)
- Computer Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Immediate Task (At least Windows 7)
- User Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Scheduled Task (At least Windows 7)
- User Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Immediate Task (At least Windows 7)

Important changes

Action: Create, Update, or Replace

- When you select the **Run whether user is logged on or not** option, a dialog box no longer prompts the administrator for credentials.
- The **Do not store password** check box is disabled. By default, the box is also checked.



Action: Delete

No change in behavior

Workarounds

For the "Scheduled Task (at least Windows 7)" and "Immediate Task (at least Windows 7)" tasks, administrators can use specific user accounts when the given user is logged on. Or, they can only have access to local resources as that user. These tasks still can run in the context of the local service.

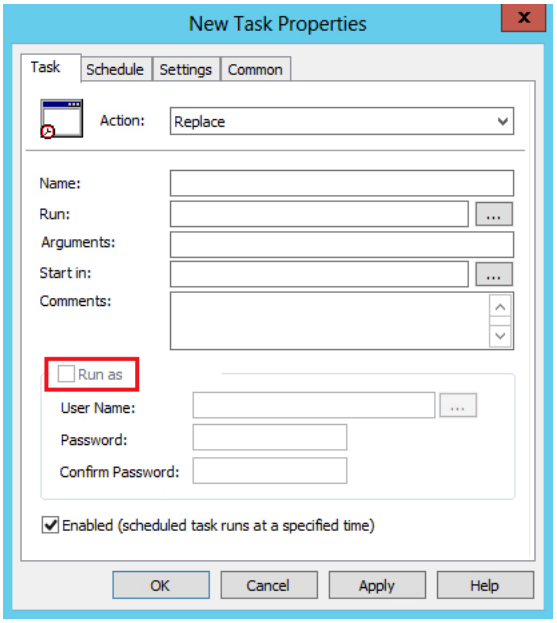
Scenario 5: Scheduled and immediate tasks (down-level) This is the down-level version of preferences used to run Scheduled Tasks in a specific security context. The ability to store credentials for scheduled tasks to run as an arbitrary user when that user is not logged on is no longer available. The following preferences are affected:

- Computer Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Scheduled Task
- Computer Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Immediate Task (Windows XP)
- User Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Scheduled Task
- User Configuration -> Control Panel Settings -> Scheduled Tasks -> New -> Immediate Task (Windows XP)

Important changes

Action: Create, Update, or Replace

- The **Run as** check box is disabled. Therefore, the **User Name**, **Password**, and **Confirm Password** fields are all disabled.



Action: Delete

No change in behavior

Workarounds

For the "Scheduled Task" and "Immediate Task (Windows XP)" items, scheduled tasks run by using the permissions that are currently available to the local service.

Scenario 6: Data Sources

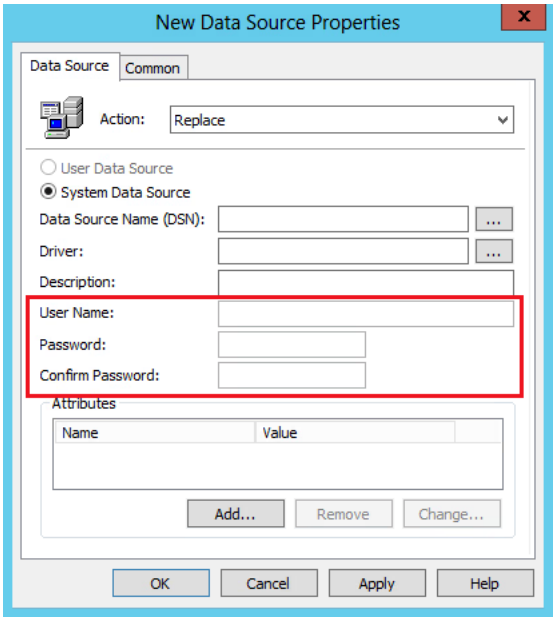
The Data Sources preference is used to associate a data source with a computer or user. This feature no longer stores credentials to enable access to data sources that are protected by a password. The following preferences are affected:

- Computer Configuration -> Control Panel Settings -> Data Sources
- User Configuration -> Control Panel Settings -> Data Sources

Important Changes

Action: Create, Update, or Replace

- The **User Name**, **Password**, and **Confirm Password** fields are disabled:



Action: Delete

- No change in behavior

Workarounds

No workarounds are available. This preference no longer stores credentials to allow access to data sources that are protected by a password.

Deprecation of CPassword

Removing CPassword

The Windows PowerShell script that is included in this Microsoft Knowledge Base article detects whether a domain contains any Group Policy Preferences that might use CPassword. If CPassword XML is detected in a given preference, it is displayed in this list.

Detecting CPassword preferences

This script must be run from a local directory on the domain controller that you want to clean. Copy and save the contents to a new Windows PowerShell file, determine your system drive, and then run the script as indicated in the following usage.

<#.SYNOPSISGroup Policy objects in your domain can have preferences that store passwords for different tasks, such as the following: 1. Data Sources 2. Drive Maps 3. Local Users 4. Scheduled Tasks (both XP and up-level) 5. ServicesThese passwords are stored in SYSVOL as part of GP preferences and are not secure because of weak encryption (32-byte AES). Therefore, we recommend that you not deploy such preferences in your domain environment and remove any such existing preferences. This script is to help administrator find GP Preferences in their domain's SYSVOL that contains passwords.

.DESCRIPTIONThis script should be run on a DC or a client computer that is installed with RSAT to print all the preferences that contain password with information such as GPO, Preference Name, GPEdit path under which this preference is defined.After you have a list of affected preferences, these preferences can be removed by using the editor in the Group Policy Management Console.

.SYNTAXGet-SettingsWithCPassword.ps1 [-Path <String>] .EXAMPLEGet-SettingsWithCPassword.ps1 -Path %WinDir%\SYSVOL\domainGet-SettingsWithCPassword.ps1 -Path <GPO Backup Folder Path>

.NOTESIf Group Policy PS module is not found the output will contain GPO GUIDs instead of GPO names. You can either run this script on a domain controller or rerun the script on the client after you have installed RSAT and enabled the Group Policy module.Or, you can use GPO GUIDs to obtain GPO names by using the Get-GPO cmdlet.

.LINKhttp://go.microsoft.com/fwlink/?LinkID=390507

#>#----------# Input parameters#----------param([string]$Path = $(throw "-Path is required.") # Directory path where GPPs are located.)#----------$isGPModuleAvailable = $false$impactedPrefs = { "Groups.xml",

"ScheduledTasks.xml","Services.xml", "DataSources.xml", "Drives.xml" }#----------# import Group olicy module if available#----------if (-not (Get-Module -name "GroupPolicy")){ if (Get-Module -ListAvailable | Where-Object { $_.Name -ieq "GroupPolicy" }) { $isGPModuleAvailable = $true Import-Module "GroupPolicy" } else { Write-Warning "Unable to import Group Policy module for PowerShell. Therefore, GPO guids will be reported. Run this script on DC to obtain the GPO names, or use the Get-GPO cmdlet (on DC) to obtain the GPO name from GPO guid." }}else{ $isGPModuleAvailable = $true}Function Enum-SettingsWithCpassword ([string]$sysvolLocation){ # GPMC tree paths $commonPath = " -> Preferences -> Control Panel Settings -> " $driveMapPath = " -> Preferences -> Windows Settings -> "

Recursively obtain all the xml files within the SYVOL location $impactedXmls = Get-ChildItem $sysvolLocation -Recurse -Filter "*.xml" | Where-Object { $impactedPrefs -cmatch $_.Name }

Each xml file contains multiple preferences. Iterate through each preference to check whether it # contains cpassword attribute and display it. foreach ($file in $impactedXmls) { $fileFullPath = $file.FullName

Set GPP category. If file is located under Machine folder in SYSVOL # the setting is defined under computer configuration otherwise the # setting is a to user configuration if ($fileFullPath.Contains("Machine")) { $category = "Computer Configuration" } elseif ($fileFullPath.Contains("User")) { $category = "User Configuration" } else { $category = "Unknown" } # Obtain file content as XML try { [xml]$xmlFile = get-content $fileFullPath -ErrorAction Continue } catch [Exception]{ Write-Host $_.Exception.Message } if ($xmlFile -eq $null) { continue } switch ($file.BaseName) { Groups { $gppWithCpassword = $xmlFile.SelectNodes("Groups/User") | where-Object { [String]::IsNullOrEmpty($_.Properties.cpassword) -eq $false } $preferenceType = "Local Users" } ScheduledTasks { $gppWithCpassword = $xmlFile.SelectNodes("ScheduledTasks/*") | where-Object { [String]::IsNullOrEmpty($_.Properties.cpassword) -eq $false } $preferenceType = "Scheduled Tasks" } DataSources { $gppWithCpassword = $xmlFile.SelectNodes("DataSources/DataSource") | where-Object { [String]::IsNullOrEmpty($_.Properties.cpassword) -eq $false } $preferenceType = "Data sources" } Drives { $gppWithCpassword = $xmlFile.SelectNodes("Drives/Drive") | where-Object { [String]::IsNullOrEmpty($_.Properties.cpassword) -eq $false } $preferenceType = "Drive Maps" } Services { $gppWithCpassword = $xmlFile.SelectNodes("NTServices/NTService") | where-Object { [String]::IsNullOrEmpty($_.Properties.cpassword) -eq $false } $preferenceType = "Services" } default { # clear gppWithCpassword and preferenceType for next item. try { Clear-Variable -Name gppWithCpassword -ErrorAction SilentlyContinue Clear-Variable -Name preferenceType -ErrorAction SilentlyContinue } catch [Exception]{} } } if ($gppWithCpassword -ne $null) { # Build GPO name from GUID extracted from filePath $guidRegex = [regex]"\{(.*)\}" $match = $guidRegex.match($fileFullPath) if ($match.Success) { $gpoGuid = $match.groups[1].value $gpoName = $gpoGuid } else { $gpoName = "Unknown" } if($isGPModuleAvailable -eq $true) { try { $gpoInfo = Get-GPO -Guid $gpoGuid -ErrorAction Continue $gpoName = $gpoInfo.DisplayName } catch [Exception] { Write-Host $_.Exception.Message } } # display prefrences that contain cpassword foreach ($gpp in $gppWithCpassword) { if ($preferenceType -eq "Drive Maps") { $prefLocation = $category + $driveMapPath + $preferenceType } else { $prefLocation = $category + $commonPath + $preferenceType } $obj = New-Object -typeName PSObject $obj | Add-Member –membertype NoteProperty –name GPOName –value ($gpoName) –passthru | Add-Member -MemberType NoteProperty -name Preference -value ($gpp.Name) -passthru | Add-Member -MemberType NoteProperty -name Path -value ($prefLocation) Write-Output $obj } } # end if $gppWithCpassword } # end foreach $file} # end functions Enum-PoliciesWithCpassword#----------# Check whether Path is valid. Enumerate all settings that contain cpassword. #----------if (Test-Path $Path){ Enum-SettingsWithCpassword $Path}else{ Write-Warning "No such directory: $Path"}

Example usage (assumes that the system drive is C)

.\Get-SettingsWithCPassword.ps1 –path “C:\Windows\SYSVOL\domain” | Format-List Note Be aware that you can also target any backup GPO for the **path** instead of the domain.

The detection script generates a list that resembles the following:

```
GPOName    : User mapped drive
Preference : G:
Path       : User Configuration -> Preferences -> Windows Settings -> Drive Maps

GPOName    : Local User and Task
Preference : Administrator (built-in)
Path       : User Configuration -> Preferences -> Control Panel Settings -> Local Users

GPOName    : Local User and Task
Preference : cmd
Path       : User Configuration -> Preferences -> Control Panel Settings -> Scheduled Tasks

GPOName    : Computer service
Preference : computer service
Path       : Computer Configuration -> Preferences -> Control Panel Settings -> Services
```

For longer lists, consider saving the output to a file:

.\Get-SettingsWithCPassword.ps1 –path “C:\Windows\SYSVOL\domain” | ConvertTo-Html > gpps.html

Removing CPassword preferences

In order to remove the preferences that contain CPassword data, we suggest that you use Group Policy Management Console (GPMC) on the domain controller or from a client that has Remote Server Administration Tools installed. You can remove any preference in five steps on these consoles. To do this, follow these steps:

1. In GPMC, open the preference that contains CPassword data.
2. Change the action to **Delete** or **Disable**, as applicable to the preference.
3. Click **OK** to save your changes.
4. Wait for one or two Group Policy refresh cycles to allow changes to propagate to clients.
5. After changes are applied on all clients, delete the preference.
6. Repeat steps 1 through 5 as needed to clean your whole environment. When the detection script returns zero results, you are finished.

File name	SHA1 hash	SHA256 hash
Windows6.0-KB2928120-ia64.msu	B2A74305CB56191774BFCF9FCDEAA983B26DC9A6	DCE8C0F9CEB97DBF1F7B9BAF76458B3770EF01C0EDC581621B
Windows6.0-KB2928120-x64.msu	386457497682A2FB80BC93346D85A9C1BC38FBF7	1AF67EB12614F37F4AC327E7B5767AFA085FE676F6E81F0CED9
Windows6.0-KB2928120-x86.msu	42FF283781CEC9CE34EBF459CA1EFE011D5132C3	016D7E9DBBC5E487E397BE0147B590CFBBB5E83795B997894870
Windows6.1-KB2928120-ia64.msu	5C2196832EC94B99AAF9B074D3938525B7219690	9958FA58134F55487521243AD9740BEE0AC210AC290D45C8322E
Windows6.1-KB2928120-x64.msu	EA5332F4E289DC799611EAB8E3EE2E86B7880A4B	417A2BA34F8FD367556812197E2395ED40D8B394F9224CDCBE8A
Windows6.1-KB2928120-x86.msu	7B7B6EE24CD8BE1AB3479F9E1CF9C98982C8BAB1	603206D44815EF2DC262016ED13D6569BE13D06E2C6029FB2262
Windows8-RT-KB2928120-x64.msu	E18FC05B4CCA0E195E62FF0AE534BA39511A8593	FCAED97BF1D61F60802D397350380FADED71AED64435D3E9EA
Windows8-RT-KB2928120-x86.msu	A5DFB34F3B9EAD9FA78C67DFC7ACACFA2FBEAC0B	7F00A72D8A15EB2CA70F7146A8014E39A71CFF5E39596F379AC

File name	SHA1 hash	SHA256 hash
Windows8.1-KB2928120-x64.msu	A07FF14EED24F3241D508C50E869540915134BB4	6641B1A9C95A7E4F0D5A247B9F488887AC94550B7F1D7B1198D
Windows8.1-KB2928120-x86.msu	DE84667EC79CBA2006892452660EB99580D27306	468EE4FA3A22DDE61D85FD3A9D0583F504105DF2F8256539051I
Windows8.1-KB2961899-x64.msu	10BAE807DB158978BCD5D8A7862BC6B3EF20038B	EC26618E23D9278FC1F02CA1F13BB289E1C6C4E0C8DA5D22E1I
Windows8.1-KB2961899-x86.msu	230C64447CC6E4AB3AD7B4D4655B8D8CEFBFBE98	E3FAD567AB6CA616E42873D3623A777185BE061232B952938A8