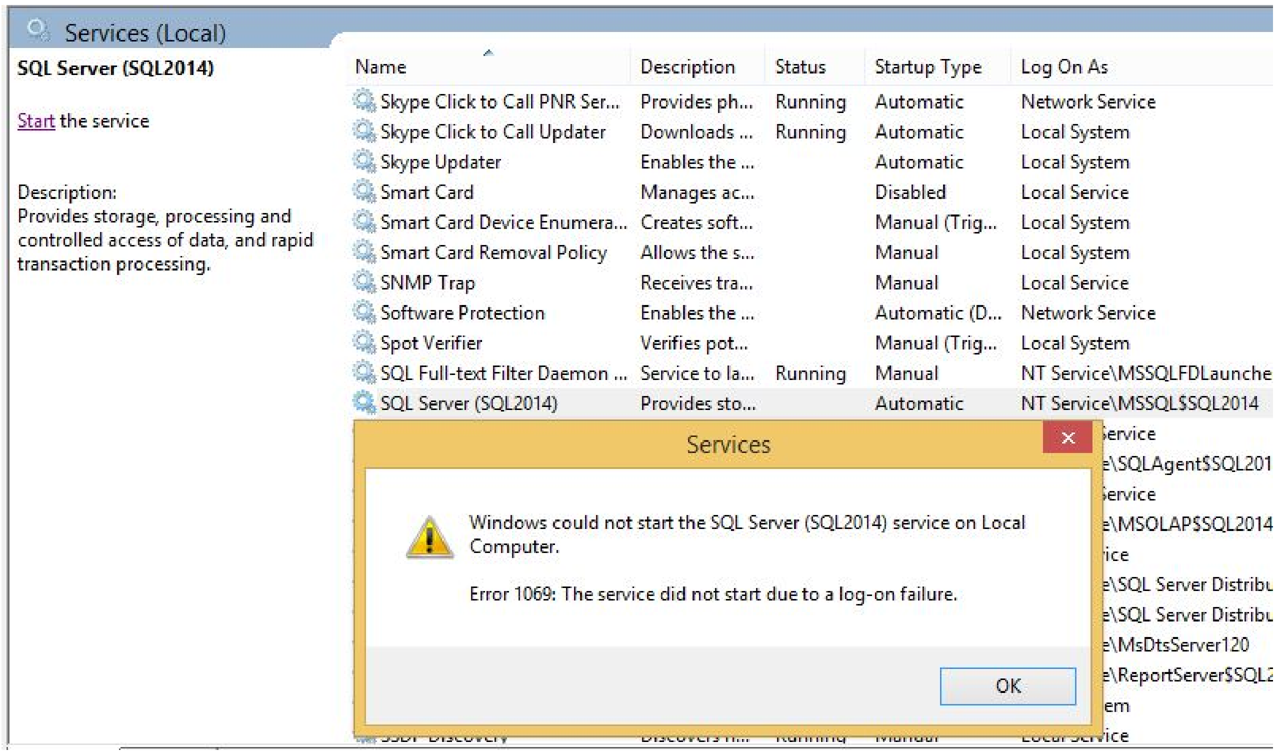
Rebuilding the "Log on as a service" list after it has been overwritten by Group Policy

codykonior.com/2015/11/16/rebuilding-the-log-on-as-a-service-list-after-it-has-been-overwritten-by-group-policy

November 16, 2015

Updated 2017-04-26: Removed "gpupdate /force" from the end of the sample script. This can overwrite the changes you just made with the group policy you were trying to avoid in the first place!

Recently I came across this kind of error starting an SQL Server service after a server had unexpectedly lost power and rebooted.



Services (Local)

SQL Server (SQL2014)

Start the service

Description:
Provides storage, processing and controlled access of data, and rapid transaction processing.

Name	Description	Status	Startup Type	Log On As
Skype Click to Call PNR Ser...	Provides ph...	Running	Automatic	Network Service
Skype Click to Call Updater	Downloads ...	Running	Automatic	Local System
Skype Updater	Enables the ...		Automatic	Local System
Smart Card	Manages ac...		Disabled	Local Service
Smart Card Device Enumera...	Creates soft...		Manual (Trig...	Local System
Smart Card Removal Policy	Allows the s...		Manual	Local System
SNMP Trap	Receives tra...		Manual	Local Service
Software Protection	Enables the ...		Automatic (D...	Network Service
Spot Verifier	Verifies pot...		Manual (Trig...	Local System
SQL Full-text Filter Daemon ...	Service to la...	Running	Manual	NT Service\MSSQLFDLaunche
SQL Server (SQL2014)	Provides sto...		Automatic	NT Service\MSSQL$SQL2014

Services

Windows could not start the SQL Server (SQL2014) service on Local Computer.

Error 1069: The service did not start due to a log-on failure.

OK

Let's investigate through PowerShell to get some quick error messages to start with.

```
Start-Service 'MSSQL$SQL2014' -Verbose

VERBOSE: Performing the operation "Start-Service" on target "SQL Server (SQL2014)
(MSSQL$SQL2014)".
Start-Service : Service 'SQL Server (SQL2014) (MSSQL$SQL2014)' cannot be started
due to the following error: Cannot open MSSQL$SQL2014 service on computer '.'.
At line:1 char:1
+ Start-Service  'MSSQL$SQL2014' -Verbose
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : OpenError:
(System.ServiceProcess.ServiceController:ServiceController) [Start-Service],
ServiceCommandException
    + FullyQualifiedErrorId :
CouldNotStartService,Microsoft.PowerShell.Commands.StartServiceCommand
```

Our next step is the error log.

```
Get-EventLog -LogName System -Newest 5

   Index Time          EntryType   Source                 InstanceID Message
   ----- ----          ---------   ------                 ---------- -------
   16621 Nov 16 17:28  Error       Service Control M...   3221232472 The SQL
Server (SQL2014) service failed to start due to the following error: ...
   16620 Nov 16 17:28  Error       Service Control M...   3221232513 The
MSSQL$SQL2014 service was unable to log on as NT Service\MSSQL$SQL2014 with the
currently configured password due to the foll...
   16619 Nov 16 17:24  Information WMPNetworkSvc          1074607998 The
description for Event ID '1074607998' in Source 'WMPNetworkSvc' cannot be found.
The local computer may not have the necessa...
   16618 Nov 16 17:24  Information WMPNetworkSvc          1074607996 The
description for Event ID '1074607996' in Source 'WMPNetworkSvc' cannot be found.
The local computer may not have the necessa...
   16617 Nov 16 17:24  Information WinRM                      468900 The
description for Event ID '468900' in Source 'WinRM' cannot be found.  The local
computer may not have the necessary registry ...
```

It indicates the service account password has expired. Let's just verify the service account name.

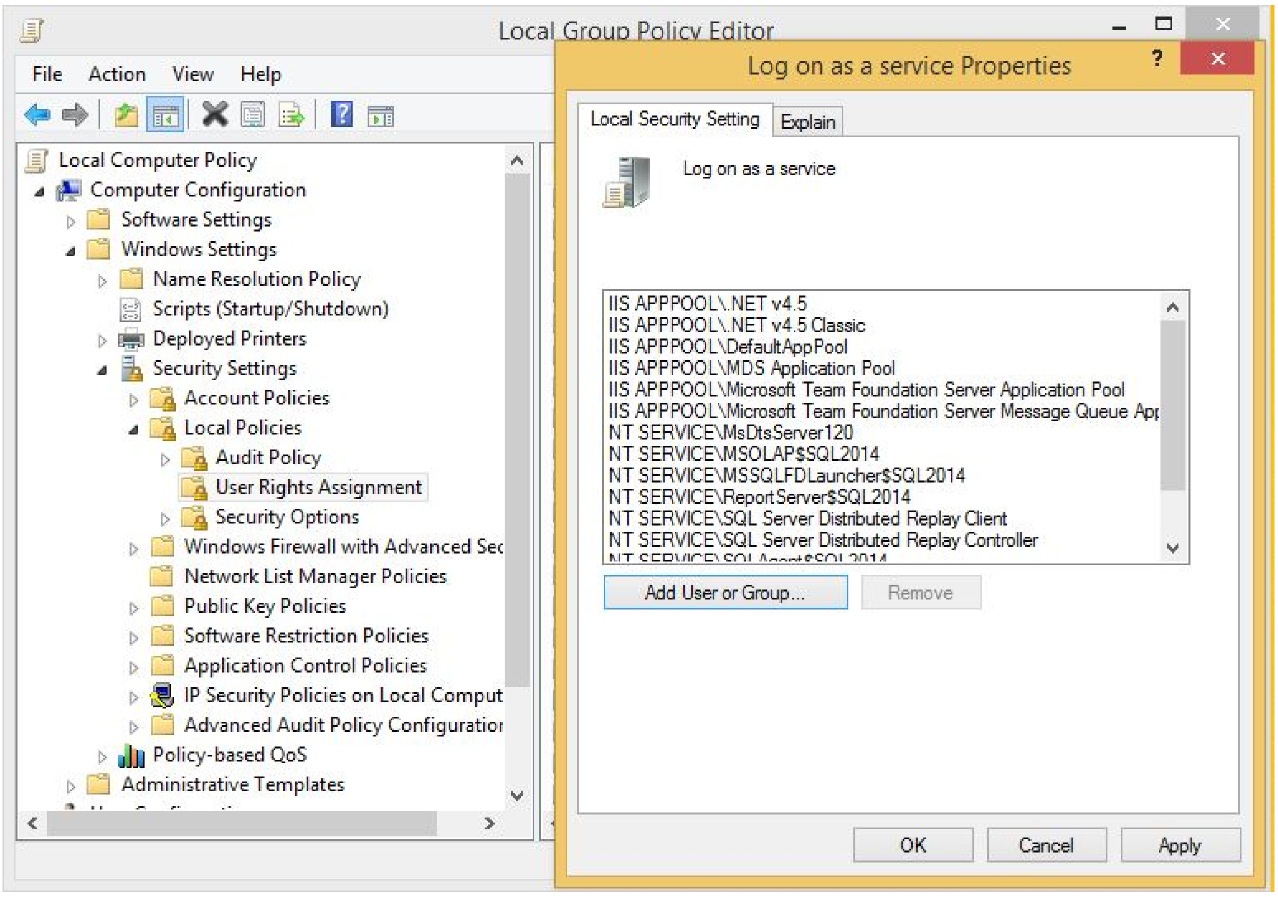
```
Get-CimInstance -ClassName Win32_Service  | Where { $_.Name -eq 'MSSQL$SQL2014' }
| Select StartName

StartName
---------
NT Service\MSSQL$SQL2014
```

So far we've determined:

- We can't start the service.
- The password is wrong.
- But we're using a virtual service account and these don't have passwords as we know them.

If you tried to reset the password (through SQL Server Configuration Manager or SMO objects) you'd get an invalid password prompt. So what's going on?



```
IIS APPPOOL\.NET v4.5
IIS APPPOOL\.NET v4.5 Classic
IIS APPPOOL\DefaultAppPool
IIS APPPOOL\MDS Application Pool
IIS APPPOOL\Microsoft Team Foundation Server Application Pool
IIS APPPOOL\Microsoft Team Foundation Server Message Queue App
NT SERVICE\MsDtsServer120
NT SERVICE\MSOLAP$SQL2014
NT SERVICE\MSSQLFDLauncher$SQL2014
NT SERVICE\ReportServer$SQL2014
NT SERVICE\SQL Server Distributed Replay Client
NT SERVICE\SQL Server Distributed Replay Controller
```

In this contrived example I've removed a few required accounts from the "Log on as a service" list. In real life, the entire list can be easily overwritten by pushing out a group policy through Active Directory, and once it's done it's done for good as it isn't restored automatically even when the policy is later removed.

Some accounts aren't affected by this; namely services running under LocalSystem, Local Service, and Network Service. This is because these automatically have rights which can't be revoked in this way (this was described in the Group Policy help screen for this section).

What I wanted to do was work out how to rebuild the list after this has happened. I investigated just a few servers and came up with at least four kinds of accounts which should be added back in:

- NT SERVICE\ALL SERVICES which is built into Windows.
- Accounts which services are configured to run under (aside from the exclusions listed above).
- Some special SQL Server local group accounts. I may not have caught them all.
- Accounts which represent IIS AppPool accounts.

And I wrote a PowerShell script to calculate it out. It needs to be run as Administrator to pull in any IIS AppPool information if it exists:

```
$ErrorActionPreference = "Stop"
Set-StrictMode -Version Latest

function Find-LogonAsService {
    [CmdletBinding()]
    param (
    )

    # Defaults from Windows
    $ignoreAccounts = @("LocalSystem", "NT Authority\LocalService", "NT
Authority\Local Service", "NT AUTHORITY\NetworkService", "NT AUTHORITY\Network
Service")
    $accounts = @("NT SERVICE\ALL SERVICES")

    # Accounts that enabled services should run under
    $accounts += Get-WmiObject -Class Win32_Service | Where { $_.StartMode -ne
"Disabled" } | Select-Object -ExpandProperty StartName

    # Special groups created for SQL Server
    $accounts += Get-WmiObject -Class Win32_Account -Namespace "root\cimv2" -
Filter "LocalAccount=True" | Where { ($_.SIDType -ne 1 -or !$_.Disabled) -and
$_.Name -like "SQLServer*User$*" } | Select-Object -ExpandProperty Name

    # IIS AppPool entities
    try {
        Import-Module WebAdministration
        Get-ChildItem IIS:\AppPools | % {
            $accounts += "IIS APPPOOL\$($_.Name)"
        }
    } catch {
        Write-Warning "** No IIS, or PowerShell not running as Administrator: $_"
    }

    # LocalSystem can be ignored, as it's really NT Authority\SYSTEM, which will
    # be covered by other accounts (like ALL SERVICES).sq,ser
    $accounts | Sort -Unique | Where { $ignoreAccounts -notcontains $_ }
}

function Grant-LogOnAsService {
    [CmdletBinding(SupportsShouldProcess = $true)]
    param (
        [Parameter(Mandatory = $true, ValueFromPipeline = $true)]
        $User
    )

    begin {
        $secedit = "C:\Windows\System32\secedit.exe"
        $gpupdate = "C:\Windows\System32\gpupdate.exe"
        $seceditdb = "$($env:TEMP)\secedit.sdb"

        $oldSids = ""
        $newSids = ""
        $secfileInput = [System.IO.Path]::GetTempFileName()
        $secfileOutput = [System.IO.Path]::GetTempFileName()

        # Get list of currently used SIDs
        &$secedit /export /cfg $secfileInput | Write-Debug

        # Find the line with existing SIDs if it exists
        if (((Get-Content $secfileInput) -join [Environment]::NewLine) -match
"SeServiceLogonRight = (.*)") {
            $oldSids = $Matches[1]
        }
    }

    process {
        # Try to convert each account name to *SID, otherwise just use the account
name
        try {
            $userAccount = New-Object System.Security.Principal.NTAccount($User)
            $userTranslated =
"*$($userAccount.Translate([System.Security.Principal.SecurityIdentifier]))"
        } catch {
            $userTranslated = $User
        }

        # Only add it to the list if neither SID nor name exist already
        if (!$oldSids.Contains($userTranslated) -and !$oldSids.Contains($User)) {
            $PSCmdlet.ShouldProcess($User) | Out-Null

            if ($newSids) {
                $newSids += ",$userTranslated"
            } else {
                $newSids += $userTranslated
            }
        }
    }

    end {
        # Only update if new SIDs are needed
        if ($newSids) {
            # Concatenate existing SIDs; if there's only one SID it has a newline
            if ($oldSids) {
                $allSids = $oldSids.Trim() + "," + $newSids
            } else {
                $allSids = $newSids
            }

            # Replace the section with the concatenated SID list, or add a new one
            $secFileContent = Get-Content $secfileInput | %{
                if ($oldSids -and $_ -match "SeServiceLogonRight = (.*)") {
                    "SeServiceLogonRight = $allSids"
                } else {
                    $_

                    if ($_ -eq "[Privilege Rights]" -and !$oldSids) {
                        "SeServiceLogonRight = $allSids"
                    }
                }
            }

            Set-Content -Path $secFileOutput -Value $secFileContent -WhatIf:$false
            # If we're really doing it, make the change
            if (!$WhatIfPreference) {
                &$secedit /import /db $seceditdb /cfg $secfileOutput
                &$secedit /configure /db $seceditdb | Write-Debug
                Remove-Item $seceditdb
            }
        } else {
            Write-Verbose "No change"
        }

        Remove-Item $secfileInput -WhatIf:$false
        Remove-Item $secfileOutput -WhatIf:$false
    }
}

Find-LogonAsService
"--"

Find-LogonAsService | Grant-LogOnAsService -WhatIf
"--"

Find-LogonAsService | Grant-LogOnAsService -Verbose
```

```
IIS APPPOOL\.NET v4.5
IIS APPPOOL\.NET v4.5 Classic
IIS APPPOOL\DefaultAppPool
IIS APPPOOL\MDS Application Pool
IIS APPPOOL\Microsoft Team Foundation Server Application Pool
IIS APPPOOL\Microsoft Team Foundation Server Message Queue Application Pool
NT SERVICE\ALL SERVICES
NT Service\MsDtsServer120
NT Service\MSOLAP$SQL2014
NT Service\MSSQL$SQL2014
NT Service\MSSQLFDLauncher$SQL2014
NT Service\ReportServer$SQL2014
NT Service\SQL Server Distributed Replay Client
NT Service\SQL Server Distributed Replay Controller
NT Service\SQLAgent$SQL2014
SQLServer2005SQLBrowserUser$VM-WIN81
SQLServerMSASUser$VM-WIN81$SQL2014
--
What if: Performing the operation "Grant-LogOnAsService" on target "NT SERVICE\ALL
SERVICES".
What if: Performing the operation "Grant-LogOnAsService" on target "NT
Service\MSSQL$SQL2014".
--
VERBOSE: Performing the operation "Grant-LogOnAsService" on target "NT SERVICE\ALL
SERVICES".
VERBOSE: Performing the operation "Grant-LogOnAsService" on target "NT
Service\MSSQL$SQL2014".
```

As expected it has enumerated all of the expected accounts, and re-added the two I had removed for the demonstration.

The Grant-LogOnAsService function is really nice and originally came from Ned Bellavance, but it would error out in cases where there were no accounts with rights (because the corresponding line would not exist in the secedit output). I have fixed that in this version.