Log Name: Microsoft-Windows-PowerShell/Operational

Source: Microsoft-Windows-PowerShell

Date: 5/24/2020 7:00:44 PM

Event ID: 4104

Task Category: Execute a Remote Command

Level: Warning

Keywords: None

User: DESKTOP/MYNAME

Computer: DESKTOP

Description:

Creating Scriptblock text (1 of 1):

```
function ExtractPluginProperties([string]$pluginDir, $objectToWriteTo)
{
   function Unescape-Xml($s) {
      if ($s) {
         $s = $s.Replace("&lt;", "<");
         $s = $s.Replace("&gt;", ">");
         $s = $s.Replace("&quot;", '"');
         $s = $s.Replace("&apos;", "'");
         $s = $s.Replace("&#39;", "'");
         $s = $s.Replace("&amp;", "&");
      }
      return $s;
   }
```

```
# The default comparer is case insensitive and it is supported on Core CLR.
$h = new-object system.collections.hashtable

function Get-Details([string]$path, [hashtable]$h) {
    foreach ($o in (get-childitem -LiteralPath $path)) {
        if ($o.PSIsContainer) {
            Get-Details $o.PSPath $h
        } else {
            $h[$o.Name] = $o.Value
        }
    }
}

Get-Details $pluginDir $h

if ($h["AssemblyName"] -eq "Microsoft.PowerShell.Workflow.ServiceCore, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL") {

    $serviceCore = [Reflection.Assembly]::Load("Microsoft.Powershell.Workflow.ServiceCore,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL")

    if ($serviceCore -ne $null) {

        $ci = new-Object system.management.automation.cmdletinfo "New-
PSWorkflowExecutionOptions",
([Microsoft.PowerShell.Commands.NewPSWorkflowExecutionOptionCommand])
```

```
        $wf = [powershell]::Create("currentrunspace").AddCommand($ci).Invoke()


        if($wf -ne $null -and $wf.Count -ne 0) {
            $wf = $wf[0]


            foreach ($o in $wf.GetType().GetProperties()) {
                $h[$o.Name] = $o.GetValue($wf, $null)
            }
        }
    }


    if (test-path -LiteralPath $pluginDir\InitializationParameters\SessionConfigurationData) {
        $xscd = [xml](Unescape-xml (Unescape-xml (get-item -LiteralPath
$pluginDir\InitializationParameters\SessionConfigurationData).Value))


        foreach ($o in $xscd.SessionConfigurationData.Param) {
            if ($o.Name -eq "PrivateData") {
                foreach($wf in $o.PrivateData.Param) {
                    $h[$wf.Name] = $wf.Value
                }
            } else {
                $h[$o.Name] = $o.Value
            }
        }
    }
```

```
    ## Extract DISC related information
    if(test-path -LiteralPath $pluginDir\InitializationParameters\ConfigFilePath) {
        $DISCFilePath = (get-item -LiteralPath $pluginDir\InitializationParameters\ConfigFilePath).Value

        if(test-path -LiteralPath $DISCFilePath) {
            $DISCFileContent = get-content $DISCFilePath | out-string
            $DISCHash = invoke-expression $DISCFileContent

            foreach ($o in $DISCHash.Keys) {
                if ($o -ne "PowerShellVersion") {
                    $objectToWriteTo = $objectToWriteTo | add-member -membertype noteproperty -name $o
-value $DISCHash[$o] -force -passthru
                }
            }
        }
    }

    if ($h["SessionConfigurationData"]) {
        $h["SessionConfigurationData"] = Unescape-Xml (Unescape-Xml $h["SessionConfigurationData"])
    }

    foreach ($o in $h.Keys) {
        if ($o -eq 'sddl') {
            $objectToWriteTo = $objectToWriteTo | add-member -membertype noteproperty -name
'SecurityDescriptorSddl' -value $h[$o] -force -passthru
```

```
      } else {
         $objectToWriteTo = $objectToWriteTo | add-member -membertype noteproperty -name $o -value $h[$o] -force -passthru
      }
   }
}

$shellNotErrMsgFormat = $args[1]
$force = $args[2]
$args[0] | foreach {
  $shellsFound = 0;
  $filter = $_
  Get-ChildItem 'WSMan:\localhost\Plugin\' -Force:$force | ? { $_.name -like "$filter" } | foreach {
    $customPluginObject = new-object object
    $customPluginObject.pstypenames.Insert(0, 'Microsoft.PowerShell.Commands.PSSessionConfigurationCommands#PSSessionConfiguration')
    ExtractPluginProperties "$($_.PSPath)" $customPluginObject
    # this is powershell based custom shell only if its plugin dll is pwrshplugin.dll
    if (($customPluginObject.FileName) -and ($customPluginObject.FileName -match 'pwrshplugin.dll'))
    {
      $shellsFound++
      $customPluginObject
    }
   } # end of foreach

   if (!$shellsFound -and ![System.Management.Automation.WildcardPattern]::ContainsWildcardCharacters($_)))
```

```
    {
      $errMsg = $shellNotErrMsgFormat -f $_
      Write-Error $errMsg
    }
  }
```

ScriptBlock ID: a74822b9-d262-44c0-befe-36d7dda474a6

Path:

Event Xml:

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-PowerShell" Guid="{a0c1853b-5c40-4b15-8766-3cf1c58f985a}"
/>
    <EventID>4104</EventID>
    <Version>1</Version>
    <Level>3</Level>
    <Task>2</Task>
    <Opcode>15</Opcode>
    <Keywords>0x0</Keywords>
    <TimeCreated SystemTime="2020-05-25T02:00:44.579548300Z" />
    <EventRecordID>15</EventRecordID>
    <Correlation ActivityID="{61dcfbed-3216-0000-5837-de611632d601}" />
    <Execution ProcessID="4356" ThreadID="2508" />
    <Channel>Microsoft-Windows-PowerShell/Operational</Channel>
    <Computer>DESKTOP</Computer>
```

<Security UserID="S-1-5-21-2220568526-3560049053-1827468789-1001" />
</System>
<EventData>
<Data Name="MessageNumber">1</Data>
<Data Name="MessageTotal">1</Data>
<Data Name="ScriptBlockText">
function ExtractPluginProperties([string]$pluginDir, $objectToWriteTo)
{
function Unescape-Xml($s) {
if ($s) {
$s = $s.Replace("<", "<");
$s = $s.Replace(">", ">");
$s = $s.Replace(""", '"');
$s = $s.Replace("'", "'");
$s = $s.Replace("'", "'");
$s = $s.Replace("&", "&");
}
return $s;
}

The default comparer is case insensitive and it is supported on Core CLR.
$h = new-object system.collections.hashtable

function Get-Details([string]$path, [hashtable]$h) {
foreach ($o in (get-childitem -LiteralPath $path)) {

```
        if ($o.PSIsContainer) {
            Get-Details $o.PSPath $h
        } else {
            $h[$o.Name] = $o.Value
        }
    }
}

Get-Details $pluginDir $h

if ($h["AssemblyName"] -eq "Microsoft.PowerShell.Workflow.ServiceCore, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL") {

    $serviceCore = [Reflection.Assembly]::Load("Microsoft.Powershell.Workflow.ServiceCore,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL")

    if ($serviceCore -ne $null) {

        $ci = new-Object system.management.automation.cmdletinfo "New-
PSWorkflowExecutionOptions",
([Microsoft.PowerShell.Commands.NewPSWorkflowExecutionOptionCommand])
        $wf = [powershell]::Create("currentrunspace").AddCommand($ci).Invoke()

        if($wf -ne $null -and $wf.Count -ne 0) {
            $wf = $wf[0]

            foreach ($o in $wf.GetType().GetProperties()) {
```

```
            $h[$o.Name] = $o.GetValue($wf, $null)
          }
        }
      }
    }

    if (test-path -LiteralPath $pluginDir\InitializationParameters\SessionConfigurationData) {
      $xscd = [xml](Unescape-xml (Unescape-xml (get-item -LiteralPath
$pluginDir\InitializationParameters\SessionConfigurationData).Value))

      foreach ($o in $xscd.SessionConfigurationData.Param) {
        if ($o.Name -eq "PrivateData") {
          foreach($wf in $o.PrivateData.Param) {
            $h[$wf.Name] = $wf.Value
          }
        } else {
          $h[$o.Name] = $o.Value
        }
      }
    }

    ## Extract DISC related information
    if(test-path -LiteralPath $pluginDir\InitializationParameters\ConfigFilePath) {
      $DISCFilePath = (get-item -LiteralPath $pluginDir\InitializationParameters\ConfigFilePath).Value

      if(test-path -LiteralPath $DISCFilePath) {
```

```
        $DISCFileContent = get-content $DISCFilePath | out-string

        $DISCHash = invoke-expression $DISCFileContent


        foreach ($o in $DISCHash.Keys) {

          if ($o -ne "PowerShellVersion") {

            $objectToWriteTo = $objectToWriteTo | add-member -membertype noteproperty -name $o
-value $DISCHash[$o] -force -passthru

          }

        }

      }

  }


  if ($h["SessionConfigurationData"]) {

    $h["SessionConfigurationData"] = Unescape-Xml (Unescape-Xml $h["SessionConfigurationData"])

  }


  foreach ($o in $h.Keys) {

    if ($o -eq 'sddl') {

      $objectToWriteTo = $objectToWriteTo | add-member -membertype noteproperty -name
'SecurityDescriptorSddl' -value $h[$o] -force -passthru

    } else {

      $objectToWriteTo = $objectToWriteTo | add-member -membertype noteproperty -name $o -
value $h[$o] -force -passthru

    }

  }

}
```

```
$shellNotErrMsgFormat = $args[1]

$force = $args[2]

$args[0] | foreach {

  $shellsFound = 0;

  $filter = $_

  Get-ChildItem 'WSMan:\localhost\Plugin\' -Force:$force | ? { $_.name -like "$filter" } | foreach {

    $customPluginObject = new-object object

    $customPluginObject.pstypenames.Insert(0,
'Microsoft.PowerShell.Commands.PSSessionConfigurationCommands#PSSessionConfiguration')

    ExtractPluginProperties "$($_.PSPath)" $customPluginObject

    # this is powershell based custom shell only if its plugin dll is pwrshplugin.dll

    if (($customPluginObject.FileName) -and ($customPluginObject.FileName -match 'pwrshplugin.dll'))

    {

      $shellsFound++

      $customPluginObject

    }

   } # end of foreach


   if (!$shellsFound -and
![System.Management.Automation.WildcardPattern]::ContainsWildcardCharacters($_)))

   {

     $errMsg = $shellNotErrMsgFormat -f $_

     Write-Error $errMsg

   }

  }

</Data>
```

<Data Name="ScriptBlockId">a74822b9-d262-44c0-befe-36d7dda474a6</Data>

<Data Name="Path">

</Data>

</EventData>

</Event>

Log Name: Microsoft-Windows-PowerShell/Operational

Source: Microsoft-Windows-PowerShell

Date: 5/24/2020 7:00:45 PM

Event ID: 4104

Task Category: Execute a Remote Command

Level: Warning

Keywords: None

User: DESKTOP/MYNAME

Computer: DESKTOP

Description:

Creating Scriptblock text (1 of 1):

```
function Set-PSSessionConfiguration([PSObject]$customShellObject,
    [Array]$initParametersMap,
    [bool]$force,
    [string]$sddl,
    [bool]$isSddlSpecified,
    [bool]$shouldShowUI,
    [string]$resourceUri,
    [string]$pluginNotFoundErrorMsg,
```

```
    [string]$pluginNotPowerShellMsg,
    [System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]$accessMode
)
{
  $wsmanPluginDir = 'WSMan:\localhost\Plugin'
  $pluginName = $customShellObject.Name;
  $pluginDir = Join-Path "$wsmanPluginDir" "$pluginName"
  if ((!$pluginName) -or !(test-path "$pluginDir"))
  {
    Write-Error $pluginNotFoundErrorMsg
    return
  }

  # check if the plugin is a PowerShell plugin
  $pluginFileNamePath = Join-Path "$pluginDir" 'FileName'
  if (!(test-path "$pluginFileNamePath"))
  {
    Write-Error $pluginNotPowerShellMsg
    return
  }

  $pluginFileName = get-item -literalpath "$pluginFileNamePath"
  if ((!$pluginFileName) -or ($pluginFileName.Value -notmatch 'pwrshplugin.dll'))
  {
    Write-Error $pluginNotPowerShellMsg
```

```
    return
  }

  # set Initialization Parameters
  $initParametersPath = Join-Path "$pluginDir" 'InitializationParameters'
  foreach($initParameterName in $initParametersMap)
  {
      if ($customShellObject | get-member $initParameterName)
      {
          $parampath = Join-Path "$initParametersPath" $initParameterName

          if (test-path $parampath)
          {
            remove-item -path "$parampath"
          }

          # 0 is an accepted value for MaximumReceivedDataSizePerCommandMB and
MaximumReceivedObjectSizeMB
          if (($customShellObject.$initParameterName) -or ($customShellObject.$initParameterName -eq
0))
          {
            new-item -path "$initParametersPath" -paramname $initParameterName  -paramValue
"$($customShellObject.$initParameterName)" -Force
          }
      }
  }
```

```
# sddl processing
if ($isSddlSpecified)
{
   $resourcesPath = Join-Path "$pluginDir" 'Resources'
   Get-ChildItem -literalpath "$resourcesPath" | % {
      $securityPath = Join-Path "$($_.pspath)" 'Security'
      if ((@(Get-ChildItem -literalpath "$securityPath")).count -gt 0)
      {
         Get-ChildItem -literalpath "$securityPath" | % {
            $securityIDPath = "$($_.pspath)"
            remove-item -path "$securityIDPath" -recurse -force
         } #end of securityPath

         if ($sddl)
         {
            new-item -path "$securityPath" -Sddl $sddl -force
         }
      }
      else
      {
         if ($sddl)
         {
            new-item -path "$securityPath" -Sddl $sddl -force
         }
      }
```

```
    } # end of resources
    return
  } #end of sddl processing
  elseif ($shouldShowUI)
  {
     $null = winrm configsddl $resourceUri
  }

  # If accessmode is 'Disabled', we don't bother to check the sddl
  if
([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Disabled.Equals($a
ccessMode))
  {
     return
  }

  # Construct SID for network users
  [system.security.principal.wellknownsidtype]$evst = "NetworkSid"
  $networkSID = new-object system.security.principal.securityidentifier $evst,$null

  $resPath = Join-Path "$pluginDir" 'Resources'
  Get-ChildItem -literalpath "$resPath" | % {
     $securityPath = Join-Path "$($_.pspath)" 'Security'
     if ((@(Get-ChildItem -literalpath "$securityPath")).count -gt 0)
     {
        Get-ChildItem -literalpath "$securityPath" | % {
```

```
        $sddlPath = Join-Path "$($_.pspath)" 'Sddl'

        $curSDDL = (get-item -path $sddlPath).value

        $sd = new-object system.security.accesscontrol.commonsecuritydescriptor
$false,$false,$curSDDL

        $newSDDL = $null


        $disableNetworkExists = $false

        $securityIdentifierToPurge = $null

        $sd.DiscretionaryAcl | % {

           if (($_.acequalifier -eq "accessdenied") -and ($_.securityidentifier -match $networkSID) -and
($_.AccessMask -eq 268435456))

           {

              $disableNetworkExists = $true

              $securityIdentifierToPurge = $_.securityidentifier

           }

        }


        if
([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Local.Equals($acces
sMode) -and !$disableNetworkExists)

        {

           $sd.DiscretionaryAcl.AddAccess("deny", $networkSID, 268435456, "None", "None")

           $newSDDL = $sd.GetSddlForm("all")

        }

        if
([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Remote.Equals($ac
cessMode) -and $disableNetworkExists)

        {
```

```
        # Remove the specific ACE
        $sd.discretionaryacl.RemoveAccessSpecific('Deny', $securityIdentifierToPurge, 268435456,
'none', 'none')

        # if there is no discretionaryacl..add Builtin Administrators and Remote Management Users
        # to the DACL group as this is the default WSMan behavior
        if ($sd.discretionaryacl.count -eq 0)
        {
           # Built-in administrators
           [system.security.principal.wellknownsidtype]$bast = "BuiltinAdministratorsSid"
           $basid = new-object system.security.principal.securityidentifier $bast,$null
           $sd.DiscretionaryAcl.AddAccess('Allow',$basid, 268435456, 'none', 'none')

           # Remote Management Users, Win8+ only
           if ([System.Environment]::OSVersion.Version -ge "6.2.0.0")
           {
              $rmSidId = new-object system.security.principal.securityidentifier "S-1-5-32-580"
              $sd.DiscretionaryAcl.AddAccess('Allow', $rmSidId, 268435456, 'none', 'none')
           }

           # Interactive Users
           $iuSidId = new-object system.security.principal.securityidentifier "S-1-5-4"
           $sd.DiscretionaryAcl.AddAccess('Allow', $iuSidId, 268435456, 'none', 'none')
        }

        $newSDDL = $sd.GetSddlForm("all")
```

```
            }


            if ($newSDDL)
            {
                set-item -WarningAction SilentlyContinue -path $sddlPath -value $newSDDL -force
            }
          }
        }
        else
        {
          if ((([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Local.Equals($accessMode)))
          {
            new-item -path "$securityPath" -Sddl "O:NSG:BAD:P(D;;GA;;;NU)(A;;GA;;;BA)(A;;GA;;;RM)(A;;GA;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)" -force
          }
        }
    }
}

Set-PSSessionConfiguration $args[0] $args[1] $args[2] $args[3] $args[4] $args[5] $args[6] $args[7] $args[8] $args[9]
```

ScriptBlock ID: d8011222-8c34-4d4b-a11f-73b7baa98eb2

Path:

Event Xml:

<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">

<System>

<Provider Name="Microsoft-Windows-PowerShell" Guid="{a0c1853b-5c40-4b15-8766-3cf1c58f985a}" />

<EventID>4104</EventID>

<Version>1</Version>

<Level>3</Level>

<Task>2</Task>

<Opcode>15</Opcode>

<Keywords>0x0</Keywords>

<TimeCreated SystemTime="2020-05-25T02:00:45.589310400Z" />

<EventRecordID>16</EventRecordID>

<Correlation ActivityID="{61dcfbed-3216-0002-bd44-de611632d601}" />

<Execution ProcessID="4356" ThreadID="2508" />

<Channel>Microsoft-Windows-PowerShell/Operational</Channel>

<Computer>DESKTOP</Computer>

<Security UserID="S-1-5-21-2220568526-3560049053-1827468789-1001" />

</System>

<EventData>

<Data Name="MessageNumber">1</Data>

<Data Name="MessageTotal">1</Data>

<Data Name="ScriptBlockText">

function Set-PSSessionConfiguration([PSObject]$customShellObject,

[Array]$initParametersMap,

```
    [bool]$force,
    [string]$sddl,
    [bool]$isSddlSpecified,
    [bool]$shouldShowUI,
    [string]$resourceUri,
    [string]$pluginNotFoundErrorMsg,
    [string]$pluginNotPowerShellMsg,
    [System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]$accessMode
)
{
  $wsmanPluginDir = 'WSMan:\localhost\Plugin'
  $pluginName = $customShellObject.Name;
  $pluginDir = Join-Path "$wsmanPluginDir" "$pluginName"
  if ((!$pluginName) -or !(test-path "$pluginDir"))
  {
    Write-Error $pluginNotFoundErrorMsg
    return
  }

  # check if the plugin is a PowerShell plugin
  $pluginFileNamePath = Join-Path "$pluginDir" 'FileName'
  if (!(test-path "$pluginFileNamePath"))
  {
    Write-Error $pluginNotPowerShellMsg
    return
```

```
  }

  $pluginFileName = get-item -literalpath "$pluginFileNamePath"
  if ((!$pluginFileName) -or ($pluginFileName.Value -notmatch 'pwrshplugin.dll'))
  {
    Write-Error $pluginNotPowerShellMsg
    return
  }

  # set Initialization Parameters
  $initParametersPath = Join-Path "$pluginDir" 'InitializationParameters'
  foreach($initParameterName in $initParametersMap)
  {
      if ($customShellObject | get-member $initParameterName)
      {
          $parampath = Join-Path "$initParametersPath" $initParameterName

          if (test-path $parampath)
          {
            remove-item -path "$parampath"
          }

          # 0 is an accepted value for MaximumReceivedDataSizePerCommandMB and
MaximumReceivedObjectSizeMB
          if (($customShellObject.$initParameterName) -or ($customShellObject.$initParameterName -eq
0))
```

```
        {
            new-item -path "$initParametersPath" -paramname $initParameterName  -paramValue
"$($customShellObject.$initParameterName)" -Force
        }
      }
  }

  # sddl processing
  if ($isSddlSpecified)
  {
     $resourcesPath = Join-Path "$pluginDir" 'Resources'
     Get-ChildItem -literalpath "$resourcesPath" | % {
        $securityPath = Join-Path "$($_.pspath)" 'Security'
        if ((@(Get-ChildItem -literalpath "$securityPath")).count -gt 0)
        {
           Get-ChildItem -literalpath "$securityPath" | % {
              $securityIDPath = "$($_.pspath)"
              remove-item -path "$securityIDPath" -recurse -force
           } #end of securityPath

           if ($sddl)
           {
              new-item -path "$securityPath" -Sddl $sddl -force
           }
        }
        else
```

```
        {
            if ($sddl)
            {
                new-item -path "$securityPath" -Sddl $sddl -force
            }
        }
    } # end of resources
    return
} #end of sddl processing
elseif ($shouldShowUI)
{
    $null = winrm configsddl $resourceUri
}

# If accessmode is 'Disabled', we don't bother to check the sddl
if
([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Disabled.Equals($a
ccessMode))
{
    return
}

# Construct SID for network users
[system.security.principal.wellknownsidtype]$evst = "NetworkSid"
$networkSID = new-object system.security.principal.securityidentifier $evst,$null
```

```
  $resPath = Join-Path "$pluginDir" 'Resources'
  Get-ChildItem -literalpath "$resPath" | % {
     $securityPath = Join-Path "$($_.pspath)" 'Security'
     if ((@(Get-ChildItem -literalpath "$securityPath")).count -gt 0)
     {
        Get-ChildItem -literalpath "$securityPath" | % {
           $sddlPath = Join-Path "$($_.pspath)" 'Sddl'
           $curSDDL = (get-item -path $sddlPath).value
           $sd = new-object system.security.accesscontrol.commonsecuritydescriptor
$false,$false,$curSDDL
           $newSDDL = $null

           $disableNetworkExists = $false
           $securityIdentifierToPurge = $null
           $sd.DiscretionaryAcl | % {
              if (($_.acequalifier -eq "accessdenied") -and ($_.securityidentifier -match $networkSID) -and
($_.AccessMask -eq 268435456))
              {
                 $disableNetworkExists = $true
                 $securityIdentifierToPurge = $_.securityidentifier
              }
           }

           if
([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Local.Equals($acce
ssMode) -and !$disableNetworkExists)
           {
```

```
            $sd.DiscretionaryAcl.AddAccess("deny", $networkSID, 268435456, "None", "None")
            $newSDDL = $sd.GetSddlForm("all")
        }
        if
([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Remote.Equals($ac
cessMode) -and $disableNetworkExists)
        {
            # Remove the specific ACE
            $sd.discretionaryacl.RemoveAccessSpecific('Deny', $securityIdentifierToPurge, 268435456,
'none', 'none')

            # if there is no discretionaryacl..add Builtin Administrators and Remote Management Users
            # to the DACL group as this is the default WSMan behavior
            if ($sd.discretionaryacl.count -eq 0)
            {
                # Built-in administrators
                [system.security.principal.wellknownsidtype]$bast = "BuiltinAdministratorsSid"
                $basid = new-object system.security.principal.securityidentifier $bast,$null
                $sd.DiscretionaryAcl.AddAccess('Allow',$basid, 268435456, 'none', 'none')

                # Remote Management Users, Win8+ only
                if ([System.Environment]::OSVersion.Version -ge "6.2.0.0")
                {
                    $rmSidId = new-object system.security.principal.securityidentifier "S-1-5-32-580"
                    $sd.DiscretionaryAcl.AddAccess('Allow', $rmSidId, 268435456, 'none', 'none')
                }
```

```
            # Interactive Users
            $iuSidId = new-object system.security.principal.securityidentifier "S-1-5-4"
            $sd.DiscretionaryAcl.AddAccess('Allow', $iuSidId, 268435456, 'none', 'none')
          }

          $newSDDL = $sd.GetSddlForm("all")
        }

        if ($newSDDL)
        {
          set-item -WarningAction SilentlyContinue -path $sddlPath -value $newSDDL -force
        }
      }
    }
    else
    {
      if (([System.Management.Automation.Runspaces.PSSessionConfigurationAccessMode]::Local.Equals($accessMode)))
      {
        new-item -path "$securityPath" -Sddl "O:NSG:BAD:P(D;;GA;;;NU)(A;;GA;;;BA)(A;;GA;;;RM)(A;;GA;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)" -force
      }
    }
  }
}
```

Set-PSSessionConfiguration $args[0] $args[1] $args[2] $args[3] $args[4] $args[5] $args[6] $args[7] $args[8] $args[9]

</Data>

<Data Name="ScriptBlockId">d8011222-8c34-4d4b-a11f-73b7baa98eb2</Data>

<Data Name="Path">

</Data>

</EventData>

</Event>