Asked by:

5

Points

MS Mike

Joined Jan 2011

MS Mike's threads

3 Show activity

Why does Get-ACL return FileSystemRights that are invalid in FileSystemAccesRule used with Set-ACL? (Ex: -536805376)

Windows Server > Windows PowerShell

Question

0

Sign in to vote

I have a script that is reading local group permissions on folders using Get-Acl and .Access and then granting the same permissions to a domain group through FileSystemAccessRule and Set-Acl. Everything worked great until it hit a permission for a local group (COMPUTERNAME\groupname) with the FileSystemRights value of -536805376 . Then it returned this error:

New-Object : Exception calling ".ctor" with "5" argument(s): "The value '-536805376' is not valid for this usage of the type FileSystemRights.
Parameter name: fileSystemRights"
At C:\Replace-LocalGroups.ps1:54 char:26
+ $AccessRule = New-Object <<<<
system.security.AccessControl.FileSystemAccessRule($AccessRuleIdentity, $Permission, $Inheritence, $Propogation, $AccessType)
+ CategoryInfo : InvalidOperation: (:) [New-Object], MethodInvocationException
+ FullyQualifiedErrorId :
ConstructorInvokedThrowException,Microsoft.PowerShell.Commands.NewObjectCommand

Why?

Friday, April 1, 2011 11:55 PM

Reply | Quote

MS Mike 5 Points

All replies

PS > [system.enum]::getvalues([System.Security.AccessControl.FileSystemRights]) | %{$_.value__} | meas

0
Sign in to vote

Count : 23
Average :
Sum : 4524076
Maximum :
Minimum :
Property :

winnt.h contains:

#define DELETE (0x00010000L)
#define READ_CONTROL (0x00020000L)
#define WRITE_DAC (0x00040000L)
#define WRITE_OWNER (0x00080000L)
#define SYNCHRONIZE (0x00100000L)

#define STANDARD_RIGHTS_REQUIRED (0x000F0000L)

#define STANDARD_RIGHTS_ALL (0x001F0000L)

#define SPECIFIC_RIGHTS_ALL (0x0000FFFFL)

#define ACCESS_SYSTEM_SECURITY (0x01000000L)

#define MAXIMUM_ALLOWED (0x02000000L)

#define GENERIC_READ (0x80000000L)
#define GENERIC_WRITE (0x40000000L)
#define GENERIC_EXECUTE (0x20000000L)
#define GENERIC_ALL (0x10000000L)

PS > 0x80000000 -bor 0x40000000 -bor 0x20000000 -bor 0x0001000

-536805376

268435456 - FullControl
-536805376 - Modify, Synchronize
-1610612736 - ReadAndExecute, Synchronize

$Permission = [System.Security.AccessControl.FileSystemRights]"Modify, Synchronize"

http://blogs.technet.com/b/josebda/archive/2010/11/12/how-to-handle-ntfs-folder-permissions-security-descriptors-and-acls-in-powershell.aspx

Marked as answer by Dale Qiao Moderator Monday, April 4, 2011 2:40 AM

Unmarked as answer by MS Mike Monday, April 4, 2011 6:25 PM

Saturday, April 2, 2011 7:23 AM

Reply | Quote

Kazun 85,440 Points

$Permission = [System.Security.AccessControl.FileSystemRights]"Modify, Synchronize"

http://blogs.technet.com/b/josebda/archive/2010/11/12/how-to-handle-ntfs-folder-permissions-security-descriptors-and-acls-in-powershell.aspx

0
Sign in to vote

So you are saying I need to use $Permission = [System.Security.AccessControl.FileSystemRights]"Modify, Synchronize" instead of the value return by Get-Acl?

I thought those were two different values? In the link you provided, it shows -536805376 and Modify, Synchornize as two different values:

NT AUTHORITY\Authenticated Users Allow Modify, Synchronize

NT AUTHORITY\Authenticated Users Allow -536805376

Monday, April 4, 2011 4:34 PM

Reply | Quote

MS Mike 5 Points

1

Sign in to vote

Hi, you can use those values (decimal representation of the permissions) with the AccessRuleFactory:

```
System.Security.AccessControl.AccessRule AccessRuleFactory(
 System.Security.Principal.IdentityReference identityReference,
 int accessMask,
 bool isInherited,
 System.Security.AccessControl.InheritanceFlags inheritanceFlags,
 System.Security.AccessControl.PropagationFlags propagationFlags,
 System.Security.AccessControl.AccessControlType type
)
```

fx

```
$ACL = Get-Acl .\test
$User = New-Object System.Security.Principal.NTAccount("DOMAIN\acct")
$newACL = $ACL.AccessRuleFactory($User,268435456,$false,'ContainerInherit,ObjectInherit','None','A
$ACL.SetAccessRule($newACL)
Set-Acl .\test -AclObject $ACL
```

- which will reset the domain user DOMAIN\acct's permissions and grant it Full permissions with standard inheritance (both ways).

The access mask is, BTW described here: http://msdn.microsoft.com/en-us/library/windows/desktop/aa374892(v=vs.85).aspx

I think you may discard the numerical values all together as they seem redundant (an artifact of the way those permissions were originally set).

Edited by Rune Mariboe Thursday, June 21, 2012 3:52 PM

Thursday, June 21, 2012 2:21 PM

Reply | Quote

Rune Mariboe Progressive A/S 165 Points

0
Sign in to vote

I am using below script to DUAL permission few folders with new domain users , but it is not working on few folders it only re permissions on root level of the folder where as on some folders it re-permissions its on root folder and all its sub folders also.
I am not sure why, can someone help what is wrong in the script. please see the error below.

Script

```
================================================================================
============================================================
$OldDomain = "Contoso"     # NetBIOS Name
$NewDomain = "Redmond"     # NetBIOS Name

# Get the current Security Descriptor
#$path =
get-content "C:\t.txt" | %{
$ACL = Get-ACL $_
$Path = $_

# Get current Access and filter down to ACEs applied here (not inherited) for users
# in the old domain
#$AccessRights =
$ACL.Access | ?{$_.IsInherited -eq $False -And $_.IdentityReference.Value -Like "$OldDomain\*"
} | %{

 # Create the new Identity string (will be NewDomain\OriginalUsername)
 $Identity = $_.IdentityReference.Value -Replace $OldDomain, $NewDomain

 # Build a new rule based on the current rule
 $NewRule = New-Object Security.AccessControl.FileSystemAccessRule(
  $Identity,
  $_.FileSystemRights,
  $_.InheritanceFlags,
  $_.PropagationFlags,
  $_.AccessControlType)

 # Add the new rule to the Access Control List
 $ACL.AddAccessRule($NewRule)


 # Apply the modified Access Control List
 Set-ACL $Path -AclObject $ACL
}
}
================================================================================
================================================================================
===
```

Error:

```
PS C:\Users\shajja\Desktop> .\DUAL.ps1
New-Object : Exception calling ".ctor" with "5" argument(s): "The value '268435456' is not valid
for this usage of the type FileSystemRights.
Parameter name: fileSystemRights"
At C:\Users\shajja\Desktop\DUAL.ps1:19 char:24
+   $NewRule = New-Object <<<<  Security.AccessControl.FileSystemAccessRule(
   + CategoryInfo          : InvalidOperation: (:) [New-Object], MethodInvocationException
   + FullyQualifiedErrorId :
ConstructorInvokedThrowException,Microsoft.PowerShell.Commands.NewObjectCommand
Exception calling "AddAccessRule" with "1" argument(s): "Value cannot be null.
Parameter name: rule"
At C:\Users\shajja\Desktop\DUAL.ps1:27 char:21
+   $ACL.AddAccessRule <<<< ($NewRule)
   + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
   + FullyQualifiedErrorId : DotNetMethodExcep
```

Sunday, July 14, 2013 5:51 AM

Reply | Quote

4313 0 Points

The information wasn't totally clear, I think, but everything was all in this thread already. Here's what

I've pieced together from a little bit of trial and error:

3

Sign in to vote

The .NET Framework FileSystemRights enum does not contain all of the possible values from the Win32 API. I ran this command to list all of the current flag values in FileSystemRights:

```
[System.Enum]::GetNames([System.Security.AccessControl.FileSystemRights]) |
% { '{0,-35}{1:X8}' -f $_, ([System.Security.AccessControl.FileSystemRights]$_).value__ }

<#

Results:

ListDirectory                       00000001
ReadData                            00000001
WriteData                           00000002
CreateFiles                         00000002
CreateDirectories                   00000004
AppendData                          00000004
ReadExtendedAttributes              00000008
WriteExtendedAttributes             00000010
Traverse                            00000020
ExecuteFile                         00000020
DeleteSubdirectoriesAndFiles        00000040
ReadAttributes                      00000080
WriteAttributes                     00000100
Write                               00000116
Delete                              00010000
ReadPermissions                     00020000
Read                                00020089
ReadAndExecute                      000200A9
Modify                              000301BF
ChangePermissions                   00040000
TakeOwnership                       00080000
Synchronize                         00100000
FullControl                         001F01FF
#>
```

Notice that the highest byte is always 00 in this enum. Next I ran this script to search for some examples of "weird" ACEs that didn't contain legal .NET FileSystemRights values:

```
Get-ChildItem c:\ -Recurse -ErrorAction SilentlyContinue | ForEach-Object {
   try {
      $weirdACEs = $_ |
      Get-Acl |
      Select-Object -ExpandProperty Access |
      Where-Object { $_.IsInherited -eq $false -and $_.FileSystemRights.ToString() -match '^-?[0-9]+
   } catch { }

   foreach ($ace in $weirdACEs) {
      '{0,-50}{1,-10:X8}{2}' -f $_.FullName, $ace.FileSystemRights.value__, $ace.IdentityReference.ToS
   }
} | Select-Object -First 20

<#
Results:

C:\ad77b46eece580b52387ae3fe3a6                  10000000  CREATOR OWNER
C:\ad77b46eece580b52387ae3fe3a6                  10000000  NT AUTHORITY\SYSTEM
C:\ad77b46eece580b52387ae3fe3a6                  10000000  BUILTIN\Administrators
C:\ad77b46eece580b52387ae3fe3a6                  A0000000  BUILTIN\Users
C:\ad77b46eece580b52387ae3fe3a6                  10000000  NT SERVICE\TrustedInstaller
C:\Program Files                                 10000000  CREATOR OWNER
C:\Program Files                                 10000000  NT AUTHORITY\SYSTEM
C:\Program Files                                 10000000  BUILTIN\Administrators
C:\Program Files                                 A0000000  BUILTIN\Users
C:\Program Files                                 10000000  NT SERVICE\TrustedInstaller
C:\Program Files (x86)                           10000000  CREATOR OWNER
C:\Program Files (x86)                           10000000  NT AUTHORITY\SYSTEM
C:\Program Files (x86)                           10000000  BUILTIN\Administrators
C:\Program Files (x86)                           A0000000  BUILTIN\Users
C:\Program Files (x86)                           10000000  NT SERVICE\TrustedInstaller
C:\Users                                         A0000000  Everyone
C:\Users                                         A0000000  BUILTIN\Users
C:\Windows                                       10000000  CREATOR OWNER
```

```
C:\Windows                      10000000  NT AUTHORITY\SYSTEM
C:\Windows                      10000000  BUILTIN\Administrators
#>
```

Looking at the first reply in this thread, you'll see that winnt.h defines these additional values for this mask which are not legal in a .NET FileSystemRights value, and some of these values are what is being found in the "weird" ACEs (GENERIC_READ and GENERIC_READ|GENERIC_EXECUTE , in the case of my test output):

```
#define ACCESS_SYSTEM_SECURITY  (0x01000000L)

#define MAXIMUM_ALLOWED   (0x02000000L)

#define GENERIC_READ   (0x80000000L)
#define GENERIC_WRITE   (0x40000000L)
#define GENERIC_EXECUTE   (0x20000000L)
#define GENERIC_ALL   (0x10000000L)
```

As Rune Mariboe mentioned in the 6/21/2012 post, there is an AccessRuleFactory method on the FileSystemSecurity class. That method seems to accept the Win32 API values that can't be casted to a FileSystemRights value in .NET. I used this code as a test:

```
$acl = Get-Acl -Path 'C:\Program Files'
$weirdACE = $acl.Access | ? { $_.FileSystemRights.ToString() -match '^-?[0-9]+$' } | Select-Object -F

$newAce = $acl.AccessRuleFactory(
    $weirdACE.IdentityReference,
    $weirdACE.FileSystemRights,
    $weirdACE.IsInherited,
    $weirdACE.InheritanceFlags,
    $weirdACE.PropagationFlags,
    $weirdACE.AccessControlType
)

$weirdAce | fl * -Force
$newAce | fl * -Force

<#
Results:

FileSystemRights  : 268435456
AccessControlType : Allow
IdentityReference : CREATOR OWNER
IsInherited       : False
InheritanceFlags  : ContainerInherit, ObjectInherit
PropagationFlags  : InheritOnly


FileSystemRights  : 268435456
AccessControlType : Allow
IdentityReference : CREATOR OWNER
IsInherited       : False
InheritanceFlags  : ContainerInherit, ObjectInherit
PropagationFlags  : InheritOnly

#>
```

So there you have it. In your script:

```
#change this line:

$NewRule = New-Object Security.AccessControl.FileSystemAccessRule(

# to this:

$NewRule = $ACL.AccessRuleFactory(
```

Edited by David Wyatt Monday, July 15, 2013 3:22 AM edit

Sunday, July 14, 2013 2:13 PM

Reply | Quote

David Wyatt Palantir Technologies 37,659 Points

0

Sign in to vote

Hi David thanks for the details information !!

I made the changes to the script as suggested by you but i got some errors when i ran it, i have copied the error below any help in this would be highly appreciated..

```
$OldDomain = "Contoso"     # NetBIOS Name
$NewDomain = "redmond"     # NetBIOS Name

# Get the current Security Descriptor
#$path =
get-content "C:\t.txt" | %{
$ACL = Get-ACL $_
$Path = $_

# Get current Access and filter down to ACEs applied here (not inherited) for users
# in the old domain
#$AccessRights =
$ACL.Access | ?{$_.IsInherited -eq $False -And $_.IdentityReference.Value -Like "$OldDomain\*" } | %{

  # Create the new Identity string (will be NewDomain\OriginalUsername)
  $Identity = $_.IdentityReference.Value -Replace $OldDomain, $NewDomain

  # Build a new rule based on the current rule
  $NewRule = $ACL.AccessRuleFactory(
    $Identity,
    $_.FileSystemRights,
    $_.InheritanceFlags,
    $_.PropagationFlags,
    $_.AccessControlType)

  # Add the new rule to the Access Control List
  $ACL.AddAccessRule($NewRule)


  # Apply the modified Access Control List
  Set-ACL $Path -AclObject $ACL
}
}
```

Error:

```
================================================================
========

PS C:\Users\shajja\Desktop> .\DUAL.ps1
Cannot find an overload for "AccessRuleFactory" and the argument count: "5".
At C:\Users\shajja\Desktop\DUAL.ps1:19 char:36
+   $NewRule = $ACL.AccessRuleFactory <<<< (
    + CategoryInfo          : NotSpecified: (:) [], MethodException
    + FullyQualifiedErrorId : MethodCountCouldNotFindBest

Exception calling "AddAccessRule" with "1" argument(s): "Value cannot be null.
Parameter name: rule"
At C:\Users\shajja\Desktop\DUAL.ps1:27 char:21
+   $ACL.AddAccessRule <<<< ($NewRule)
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
================================================================
Cannot find an overload for "AccessRuleFactory" and the argument count: "5".
At C:\Users\shajja\Desktop\DUAL.ps1:19 char:36
+   $NewRule = $ACL.AccessRuleFactory <<<< (
    + CategoryInfo          : NotSpecified: (:) [], MethodException
    + FullyQualifiedErrorId : MethodCountCouldNotFindBest

Exception calling "AddAccessRule" with "1" argument(s): "Value cannot be null.
```

Parameter name: rule"
At C:\Users\shajja\Desktop\DUAL.ps1:27 char:21
+ $ACL.AddAccessRule <<<< ($NewRule)
+ CategoryInfo : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : DotNetMethodException

Sunday, July 14, 2013 2:40 PM

Reply | Quote

4313 0 Points

Oh, right. The AccessRuleFactory method has an extra IsInherited argument. You also might need to cast the Identity variable to an NTAccount type, since the AccessRuleFactory doesn't have an overload that accepts a string (and I'm not sure if it will cast automatically to the IdentityReference base class. Probably not.)

1

Sign in to vote

Try this:

```
# Create the new Identity string (will be NewDomain\OriginalUsername)
$Identity = [System.Security.Principal.NTAccount]($_.IdentityReference.Value -Replace $OldDo

# Build a new rule based on the current rule
$NewRule = $ACL.AccessRuleFactory(
  $Identity,
  $_.FileSystemRights,
  $_.IsInherited,
  $_.InheritanceFlags,
  $_.PropagationFlags,
  $_.AccessControlType)
```

Edited by David Wyatt Monday, July 15, 2013 3:23 AM edit

Sunday, July 14, 2013 4:31 PM

Reply | Quote

David Wyatt Palantir Technologies 37,659 Points

I searched around on this topic and found a few scattered posts regarding the .NET Framework and the Win32 API's GENERIC_* flags, but mostly regarding generating a report (and giving a useful string representation of the ACE instead of a number). Rune Mariboe's posts in this thread and one other repost of the same issue on these forums were the only ones I found that mentioned the AccessRuleFactory method.

0

Sign in to vote

This seems like useful (and somewhat obscure) enough information that I posted a TechNet wiki entry about it: http://social.technet.microsoft.com/wiki/contents/articles/18501.copying-aces-in-the-net-framework-powershell.aspx

Sunday, July 14, 2013 5:51 PM

Reply | Quote

David Wyatt Palantir Technologies 37,659 Points

Thank you VERY MUCH !!!! It worked like a charm ...You made my day.

I have changed the script as you suggested and issue got fixed :)

0

Sign in to vote

Monday, July 15, 2013 2:14 AM

Reply | Quote

4313 0 Points