NTFSSecurity Tutorial 1 - Getting, adding and removing permissions

12/05/2014 • 7 minutes to read

In this article

Summary

Managing file and folder permissions in Windows PowerShell is not that easy, and there are numerous articles and blog posts describing how it works by using the .NET classes. This is far from being comfortable, and there is one major and one minor restriction:

- Path length
- Generic rights

This post introduces the NTFSSecurity module, which provides a bunch of cmdlets for managing permissions on NTFS drives. It does not use the Windows PowerShell way to access the file system, and it works around the MAX_PATH, which is 260 characters. (For more information, see Naming Files, Paths, and Namespaces). This is achieved thanks to AlphaFS.

This post examines displaying permissions and granting users permission.

Installation

You can download the module from the Script Center Repository: File System Security PowerShell Module. Please unblock the file before extracting it.

For more information about installing Windows PowerShell modules, see Hey, Scripting Guy! How Can I Install Windows PowerShell Modules on Multiple Users' Computers?

Some backgrounds

Windows stores the permissions in the discretionary access control list (DACL), which is part of the Security Descriptor. The Security Descriptor also includes the system access control list (SACL), where the auditing is configured, and member information. This post is about permissions and it does not discuss the SACL or member information.
The DACL contains access control entries (ACEs) that define the permissions someone has on the object. Each ACE contains the following values:

- Account: Who is granted or denied access. Windows does not store the user's SamAccountName, but rather, the SID.
- Rights: The permissions granted or denied.
- Type: Grant or deny access.
- IsInherited: True if the ACE is inherited from a parent object.
- InheritanceFlags and PropagationFlags: These bits control the inheritance. The NTFSSecurity module converts the bits into something more readable that is discussed later in this post.

By default, a security descriptor on the file system inherits permissions from the parent object. Users who have full access on drive C also have full access to all subfolders if the inheritance is not disabled.

Managing permissions

Reading the permissions of a single item

The first and easiest task is to retrieve the DACL from a specific file. The cmdlet that the NTFSSecurity module provides for retrieving existing permissions is Get-NTFSAccess. You can pipe a file or folder to that cmdlet or work with the **Path** parameter:

Get-Item D:\Data | Get-NTFSAccess

Get-NTFSAccess -Path D:\Data

The output might look like this:

```
PS D:\> Get-Access -Path D:\Data


    Path: D:\Data (Inheritance enabled)

Account                           Access Rights        Applies to               Type                IsInherited          InheritedFrom
-------                           -------------        ----------               ----                -----------          -------------
RAANDREE1\randr_000               FullControl          ThisFolderSubfoldersAn... Allow              False
S-1-5-21-3893903987-2340964142-2... FullControl        ThisFolderSubfoldersAn... Allow              False
BUILTIN\Administrators            FullControl          ThisFolderSubfoldersAn... Allow              True                 D:\
NT AUTHORITY\Authenticated Users  Modify, Synchronize  ThisFolderSubfoldersAn... Allow              True                 D:\
NT AUTHORITY\SYSTEM               FullControl          ThisFolderSubfoldersAn... Allow              True                 D:\
BUILTIN\Users                     ReadAndExecute, Sy... ThisFolderSubfoldersAn... Allow              True                 D:\
```

The output is grouped by the file or folder, which is important when getting the

permissions of more than one object. Next to the path is information about if the file or folder inherits the permissions from the parent object. My example shows that four of the displayed ACEs have been inherited from drive D.

Some more details about the columns:

- Account: The account that has been granted or denied access to the item. As mentioned, Windows does not store the user's name, but rather, the SID. If the SID can be translated into the name, NTFSSecurity shows it; otherwise, the SID is displayed.
- AccessRights: These are the actual permissions that the account has been granted or denied. The list behind this field also supports generic rights.
- Applies to: The .NET Framework stores the inheritance information in two-bit fields: InheritanceFlags and PropagationFlags. These fields are quite difficult to interpret, so NTFSSecurity converts them into something that is known from the Windows Explorer:
  - ThisFolderOnly
  - ThisFolderSubfoldersAndFiles
  - ThisFolderAndSubfolders
  - ThisFolderAndFiles
  - SubfoldersAndFilesOnly
  - SubfoldersOnly
  - FilesOnly
- Type: Either Allow or Deny
- Inherited: If the ACE is inherited from the parent, this is True. The first two ACEs have been defined explicitly in the folder.
- InhertedFrom: This column only contains information if IsInherited is True, and it indicates where the ACE is inherited from.

Reading the permissions of a multiple item

All NTFSSecurity cmdlets support pipelining. If you need to get the permissions from multiple items, you do not need to run a **ForEach** loop. You can simply pipe the files and folders to **Get-NTFSAccess**.

```
dir C:\Data | Get-NTFSAccess
```

Get-NTFSAccess provides ways to filter the ACEs. A common scenario is to get the ACEs of a specific account or only those that have not been inherited.
If you want to display only permissions that have been added explicitly and hide all the inherited permissions, use the **ExcludeInherited** switch:

```
dir | Get-NTFSAccess –ExcludeInherited
```

If you want to display only the permissions assigned to a certain user, use the **Account** parameter:

```
dir | Get-NTFSAccess -Account raandree9\randr_000
```

Note: *This displays the permissions as defined in the ACL. This is not the effective permissions. Effective permissions will be discussed in an upcoming post.*

Granting access

Granting access to a file or folder is also quite easy to do by using the Add-NTFSAccess cmdlet. Add-NTFSAccess provides the following parameters:

- Account: This can be a user account name (SamAccountName) or a SID. The user account name has to contain the domain (domain\username). Built-in SIDs are also supported, such as Everyone, NT AUTHORITY\SYSTEM, or BUILTIN\Administrators. For more information, see Well-known security identifiers in Windows operating systems.
- AccessRights: This parameter takes one or more of file system rights, for example, FullControl, Modify, or Read. If you want to assign multiple rights, provide them in a comma-separated list.

Note: *Use Tab expansion or the ISE to get a list of all available values.*

- AccessType: Allow or deny
- AppliesTo: This parameter sets the scope of the ACE. The options are the same as Windows Explorer provides. By default (when not defined), the scope is **ThisFolderSubfoldersAndFiles**.

Note: *Use Tab expansion or the ISE to get a list of all available values.*

- PassThru: By default, the cmdlet does not return any data. If the **PassThru** switch is used, the cmdlet displays the ACL after adding the ACE.

The next commands give the well-known group, Authenticated Users, read access to the folder C:\Data. The built-in administrators and the local group, Editors, are getting full control:

```
Add-NTFSAccess -Path C:\Data `

-Account 'NT AUTHORITY\Authenticated Users' `

-AccessRights Read

Add-NTFSAccess -Path C:\Data `
```

-Account 'BUILTIN\Administrators', 'raandree9\Editors' `

-AccessRights FullControl

Note: *The modifying cmdlets of the NTFSSecurity Module do not return any data by default. If you want to get back the modified ACL, use the PassThru switch.*

Removing Access

Removing access is similar to adding permissions. The command **Remove-NTFSAccess** takes the same parameters as **Add-NTFSAccess**.
To remove a user from the ACL, provide the path, the account name, and the permissions you want to remove, for example:

Remove-NTFSAccess D:\Data -Account RAANDREE0\randr_000 -AccessRights Read -PassThru

If the user has different permissions than those you want to remove, nothing happens. There needs to be an exact match.

Note: *You cannot remove inherited permissions. Get-NTFSAccess informs about the source of the inherited permissions where the respective ACE can be changed or removed.*

Remove-NTFSAccess accepts pipeline input. If you want to remove all permissions for a certain user account, you can read the permissions first and then pipe the results to **Remove-NTFSAccess**. This operation can also run reclusively:

Get-ChildItem -Path d:\ -Recurse |

Get-NTFSAccess -Account raandree0\randr_000 -ExcludeInherited |

Remove-NTFSAccess

Note: *The cmdlets in the NTFSSecurity module do not provide a way to process files and folders reclusively. You have to use* ***Get-ChildItem*** *or* ***Get-ChildItem2*** *with the* ***Recurse*** *switch. (The* ***Get-ChildItem2*** *cmdlet is part of the NTFSSecurity module, and it will be discussed in a future post).*

NTFS Inheritance

After you set permissions on a parent folder, new files and subfolders that are created in the folder inherit these permissions. If you do not want them to inherit permissions, set ApplyTo to “ThisFolderOnly” when you set special permissions for the parent folder. In cases where you want to prevent certain files or subfolders from inheriting permissions, disable (or block) the inheritance.

There are two types of permissions:

- Explicit permissions: Set by default when the object is created by user action.
- Inherited permissions: Propagated to an object from a parent object. Inherited permissions ease the task of managing permissions and ensure consistency of permissions among all objects within a given container.

To add an ACE that does not affect any child elements, use the following command:

Add-NTFSAccess .\Data -Account raandree1\install -AccessRights Modify -AppliesTo ThisFolderOnly

If the **AppliesTo** parameter is not used, the ACE applies to "ThisFolderSubfoldersAndFiles," like when using the Windows Explorer to add permissions. All child elements will inherit the ACE created by the following command:

Add-Access -Path .\Data -Account BUILTIN\Administrators -AccessRights FullControl

To verify which child items have inherited the ACE, you can get and pipe all child elements recursively to **Get-NTFSAccess**. With the following command, Windows PowerShell reads only the inherited ACEs that are assigned to the built-in administrators group that are inherited from D:\Data:

dir -Recurse | Get-NTFSAccess -Account BUILTIN\Administrators -ExcludeExplicit | Where-Object InheritedFrom -eq 'D:\Data'

```
PS D:\> dir -Recurse | Get-NTFSAccess -Account BUILTIN\Administrators -ExcludeExplicit | Where-Object InheritedFrom -eq 'D:\Data'


    Path: D:\Data\Test1 (Inheritance enabled)

Account                          Access Rights           Applies to              Type                 IsInherited            InheritedFrom
-------                          -------------           ----------              ----                 -----------            -------------
BUILTIN\Administrators (S-1-5-32... FullControl           ThisFolderSubfoldersAn... Allow              True                   D:\Data


    Path: D:\Data\Test.txt (Inheritance enabled)

Account                          Access Rights           Applies to              Type                 IsInherited            InheritedFrom
-------                          -------------           ----------              ----                 -----------            -------------
BUILTIN\Administrators (S-1-5-32... FullControl           ThisFolderOnly          Allow                True                   D:\Data
```

The next post will explore how to report, enable, and disable inheritance in folders (the NTFSSecurity module provides the same feature as the Windows Explorer). I will also discuss taking ownership of files without losing the ACL.