How to Manage File System ACLs with PowerShell

Ryan Brooks
Cybersecurity Expert, Netwrix Product Evangelist

Many organizations with a Microsoft Windows environment rely on NTFS as the main file system for their storage devices that contain sensitive data. It is the easiest way for users to work with files. In order to implement a least-privilege model, which is a best practice for system security, IT security specialists and system administrators configure NTFS access control lists (ACLs) by adding access control entries (ACEs) on NTFS file servers.

NTFS Permissions Types for Files and Folders

There are both basic and advanced NTFS permissions. You can set each of the permissions to "Allow" or "Deny". Here are the **basic permissions**:

- **Full Control:** Users can modify, add, move and delete files and directories, as well as their associated properties. In addition, users can change permissions settings for all files and subdirectories.
- **Modify:** Users can view and modify files and file properties, including deleting and adding files to a directory or file properties to a file.
- **Read & Execute:** Users can run executable files, including script
- **Read:** Users can view files, file properties and directories.
- **Write:** Users can write to a file and add files to directories.

Here is the list of **advanced permissions:**

- **Traverse Folder/Execute File:** Users can navigate through folders to reach other files or folders, even if they have no permissions for these files or folders. Users can also run executable files. The Traverse Folder permission takes effect only when the group or user doesn't have the "Bypass Traverse Checking" right in the Group Policy snap-in.
- **List Folder/Read Data:** Users can view a list of files and subfolders within the folder as well as the content of the files.
- **Read Attributes:** Users can view the attributes of a file or folder, such as whether it is read-only or hidden.
- **Write Attributes:** Users can change the attributes of a file or folder.
- **Read Extended Attributes:** Users can view the extended attributes of a file or folder, such as permissions and creation and modification times.
- **Write Extended Attributes:** Users can change the extended attributes of a file or folder.
- **Create Files/Write Data:** The "Create Files" permission allows users to create files within the folder. (This permission applies to folders only.) The "Write Data" permission allows users to make changes to the file and overwrite existing content. (This permission applies to files only.)
- **Create Folders/Append Data:** The "Create Folders" permission allows users to create folders within a folder. (This permission applies to folders only.) The "Append Data" permission allows users to make changes to the end of the file, but they can't change, delete or overwrite existing data. (This permission applies to files only.)

- **Delete:** Users can delete the file or folder. (If users don't have the "Delete" permission on a file or folder, they can still delete it if they have the "Delete Subfolders And Files" permission on the parent folder.)
- **Read Permissions:** Users can read the permissions of a file or folder, such as "Full Control", "Read", and "Write".
- **Change Permissions:** Users can change the permissions of a file or folder.
- **Take Ownership:** Users can take ownership of the file or folder. The owner of a file or folder can always change permissions on it, regardless of any existing permissions that protect the file or folder.
- **Synchronize:** Users can use the object for synchronization. This enables a thread to wait until the object is in the signaled state. This right is not presented in ACL Editor. You can read more about it here.

ou can find all these user permissions by running the following PowerShell script:

```
[system.enum]::getnames([System.Security.
AccessControl.FileSystemRights])
```

NTFS permissions can be either explicit or inherited. Explicit permissions are permissions that are configured individually, while inherited permissions are inherited from the parent folder. The hierarchy for permissions is as follows:

- Explicit Deny
- Explicit Allow
- Inherited Deny
- Inherited Allow

Now that we know NTFS permissions are, let's explore how to manage them.

Get ACL for Files and Folders

The first PowerShell cmdlet used to manage file and folder permissions is "get-acl"; it lists all object permissions. For example, let's get the list of all permissions for the folder with the object path "\\fs1\shared\sales":

```
get-acl \\fs1\shared\sales | fl
```

```
PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl

Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\J.Brown Allow  FullControl
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
Audit  :
```

If you want to get a full NTFS permissions report via PowerShell, you can follow this easy how-to about exporting NTFS permissions to CSV.

Copy File and Folder Permissions

To copy permissions, a user must own both the source and target folders. The following command will copy the permissions from the "Accounting" folder to the "Sales" folder:

```
get-acl \\fs1\shared\accounting | Set-Acl
\\fs1\shared\sales
```

```
PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl

Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\J.Brown Allow  FullControl
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
Audit  :
Sddl   : O:BAG:S-1-5-21-210521867-2639090965-1213260628-513D:AI(A;OICI;FA;;;S-1-5-21-6
         928-1670731417-1160)(A;OICIID;FA;;;SY)(A;OICIID;FA;;;S-1-5-21-611411812-38042



PS C:\Users\j.carter> get-acl \\fs1\shared\accounting | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl

Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\T.Simpson Deny  FullControl
         ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

As we can see from the output of the "get-acl" commands before and after the permissions copy, the "Sales" shared folder permissions have been changed.

Set File and Folder Permissions

The PowerShell "set-acl" cmdlet is used to change the security descriptor of a specified item, such as a file, folder or a registry key; in other words, it is used to modify file or folder permissions. The following script sets the "FullControl" permission to "Allow" for the user "ENTERPRISE\T. Simpson" to the folder "Sales":

```
$acl = Get-Acl \\fs1\shared\sales

$AccessRule = New-Object System.Security.
AccessControl.FileSystemAccessRule("ENTER-
PRISE\T.Simpson","FullControl","Allow")

$acl.SetAccessRule($AccessRule)

$acl | Set-Acl \\fs1\shared\sales$acl | Set-
Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$AccessRule = New-Object System.Security.AccessControl.FileSystemAccessRule("ENTERPRISE\T.Simpson","FullControl","Allow")
$acl.SetAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\T.Simpson Deny  FullControl
         ENTERPRISE\T.Simpson Allow  FullControl
         ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
```

If you want to set other permissions to users or security groups, choose them from the table below:

Access Right	Access Right's Name in PowerShell
Full Control	FullControl
Traverse Folder / Execute File	ExecuteFile
List Folder / Read Data	ReadData
Read Attributes	ReadAttributes
Read Extended Attributes	ReadExtendedAttributes
Create Files / Write Data	CreateFiles
Create Folders / Append Data	AppendData
Write Attributes	WriteAttributes
Write Extended Attributes	WriteExtendedAttributes
Delete Subfolders and Files	DeleteSubdirectoriesAndFiles
Delete	Delete
Read Permissions	ReadPermissions

There are also permissions sets of basic access rights that can be applied:

Access Right	Access Right's Name in PowerShell	Name of the Set in PowerShell
Read	List Folder / Read Data Read Attributes Read Extended Attributes Read Permissions	Read
Write	Create Files / Write Data Create Folders / Append Data Write Attributes Write Extended Attributes	Write
Read and Execute	Traverse folder / Execute File List Folder / Read Data Read Attributes Read Extended Attributes Read Permissions	ReadAndExecute
Modify	Traverse folder / Execute File List Folder / Read Data Read Attributes Read Extended Attributes Create Files / Write Data Create Folders / Append Data Write Attributes Write Extended Attributes Delete Read Permissions	Modify

Remove User Permissions

To remove a permission, use the “RemoveAccessRule” parameter. Let’s delete the “Allow FullControl” permission for T.Simpson to the “Sales” folder:

```
$acl = Get-Acl \\fs1\shared\sales

$AccessRule = New-Object System.Security.
AccessControl.FileSystemAccessRule("ENTER-
PRISE\T.Simpson","FullControl","Allow")

$acl.RemoveAccessRule($AccessRule)

$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$AccessRule = New-Object System.Security.AccessControl.FileSystemAccessRule ("ENTERPRISE\T.Simpson","FullControl","Allow")
$acl.RemoveAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\sales
True

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\T.Simpson Deny  FullControl
         ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Notice that T.Simpson still has the “Deny FullControl” permission. To remove it, let’s use the command “PurgeAccessRules”, which will completely wipe T.Simpson’s permissions to the “Sales” folder:

```
$acl = Get-Acl \\fs1\shared\sales

$usersid = New-Object System.Security.Principal.Ntaccount ("ENTERPRISE\T.Simpson")

$acl.PurgeAccessRules($usersid)

$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$usersid = New-Object System.Security.Principal.Ntaccount ("ENTERPRISE\T.Simpson")
$acl.PurgeAccessRules($usersid)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Note that "PurgeAccessRules" doesn't work with a string user name; it works only with SIDs. Therefore, we used the "Ntaccount" class to convert the user account name from a string into a SID. Also note that "PurgeAccessRules" works only with explicit permissions; it does not purge inherited ones.

Disable or Enable Permissions Inheritance

To manage inheritance, we use the "SetAccessRuleProtection" method. It has two parameters:

- The first parameter is responsible for blocking inheritance from the parent folder. It has two states: "$true" and "$false".
- The second parameter determines whether the current inherited permissions are retained or removed. It has the same two states: "$true" and "$false".

Let's disable inheritance for the "Sales" folder and delete all inherited permissions as well:

```
$acl = Get-Acl \\fs1\shared\sales

$acl.SetAccessRuleProtection($true,$false)

$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($true,$false)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
```

Now we have only one access permission left (because it was added explicitly); all inherited permissions were removed.

Let's revert this change and enable inheritance for the folder "Sales" again:

```
$acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($false,$true)
$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($false,$true)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Change File and Folder Ownership

If you want to set an owner for a folder, you need to run the "SetOwner" method. Let's make "ENTERPRISE\J.Carter" the owner of the "Sales" folder:

```
$acl = Get-Acl \\fs1\shared\sales
$object = New-Object System.Security.Principal.Ntaccount("ENTERPRISE\J.Carter")
$acl.SetOwner($object)
$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$object = New-Object System.Security.Principal.Ntaccount("ENTERPRISE\J.Carter")
$acl.SetOwner($object)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : ENTERPRISE\j.carter
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Notice that we again used the "Ntaccount" class to convert the user account name from a string into a SID.

Note that the "SetOwner" method does not enable you to change the owner to any account you want; the account must have the "Take Ownership", "Read" and "Change Permissions" rights.

As you can see, it is very easy to manage NTFS permissions with PowerShell. But don't forget to audit NTFS permissions as well — it's critical for security to track all changes made to your file servers in order to reduce data leakage and combat the insider threat and other IT security risks. Here is a basic guide on how to audit NTFS permissions with PowerShell.