Tools for managing ACLs

Jesper M Johansson

In Windows, access control lists (ACLs) give you extremely fine control over the ability of users and processes to use resources such as files and folders. Managing ACLs can be one of the more complicated tasks related to protecting the security of your users' systems. Fortunately, there are a number of useful utilities that help automate and simplify tasks surrounding permissions and ACLs.

Most readers are familiar with the venerable cacls.exe tool that has been in every version of Windows NT since it first came out. If you run cacls.exe in Windows Vista, you are greeted with this message:

```
Microsoft Windows [Version 6.0.5744]
Copyright (c) 2006 Microsoft Corporation. All
rights reserved.

C:\Windows\system32>cacls

 NOTE: Cacls is now deprecated, please use
Icacls.
```

In addition to updates to ACLs in Windows Vista (see technetmagazine.com/issues/2007/06), Microsoft has also updated some of the tools that you use to manage ACLs. Interestingly, the most significant of these updates are command-line tools.

Icacls.exe is new in Windows Vista (and also down-level in Windows Server 2003 Service Pack 2). It will eventually replace cacls.exe, which, as you may be aware, was never completely updated to support the more granular permissions introduced with Windows 2000, making this an update that is about seven years overdue.

Surprisingly, in spite of being deprecated, cacls.exe does actually include some new features. First, it is aware of both junction points and symbolic links, and knows to traverse those. Second, it can both print and set an ACL using a Security Description Definition Language (SDDL) string.

However, in spite of the updates to cacls.exe, you really want the functionality available in icacls.exe.

Saving and restoring ACLs

A constant wish – at least of mine – for the past 10 years has been the ability to save an ACL so that you can restore it at a later date. This turns out to be one of the most complicated operations you can perform on ACLs. Except in certain special cases, it is unlikely to get back to exactly where you would have been had you not ruined the ACLs in the first place. Nevertheless, this functionality can be quite useful.

Before I go into how to save or restore ACLs, let me first explore why it is so difficult. Let's say you have a hierarchy containing user data for students at a local college. You save the ACL on 1 February. On 17 April you discover that somehow the ACL has been corrupted and you go to restore it from the saved copy. What complications could there be with this operation?

First, the new quarter started on 2 April. About 15 per cent of your students graduated; consequently, their directories no longer exist. Thus, you have ACLs in the backup file that are invalid. You also have a batch of new students, another 15 per cent, that started with the new quarter. They have home directories now, but no ACLs in the backup file. What should their ACLs be? Then of course you have the 70 per cent that are still there, but they have created new files and folders and deleted others. You can ignore the deleted ones, but how do you configure the new folders they created? What if a student decided to share a folder with his friends on 4 March? Will you break that when you restore the ACL?

Saving and restoring ACLs is not nearly as simple an operation as many people would have you believe. You need to tread extremely lightly when doing so. It is very possible, even likely, that you will have some undefined behaviour as a result. Restoring ACLs is definitely a last resort; the longer it has been since you backed it up, the more likely it is that something will break when you restore it.

If you nevertheless wish to try this functionality, run icacls.exe with the /save switch:

```
icacls <target> /save acls.bin /t
```

This would save the ACLs to a file called acls.bin. The file will contain one line for each object with an ACL followed by one line specifying the ACL in SDDL. Using the /t switch the command will operate on the object you specified and all objects and containers underneath it.

The save functionality is a welcome addition to the toolkit, but it has a few flaws. For example, it only saves the discretionary access control list (DACL), not the system access control list (SACL). In fact, if there is a SACL, it saves a dummy value that indicates this, but it does not actually save any part of the SACL.

In addition, the way the ACL is saved creates an interesting problem. Before you open the saved ACL in your favourite text editor, remember:

Do Not Edit Your Saved ACL!

If you were to open the file containing your saved ACL in a text editor, you would find that it appears to be a Unicode (UTF-16) formatted text file. In fact, that is almost exactly what it is. This might lead you to think you could edit it and save it from a text editor. Don't do it!

If you open the file that contains the saved ACLs in a text editor and then save it, you will not be able to restore the ACLs from that file. It turns out that it is not actually a Unicode text file after all. Such a file must begin with the 2 bytes 0xfffe. If you save the file with a text editor, such as Notepad, it will in fact put that flag into the file in the first 2 bytes. The icacls.exe tool, however, expects the ACL data to start at byte 0 in the file. Consequently, the tool will be unable to parse the ACLs in the file as it expects the first two bytes to be part of the string specifying the object to operate on. Your backup file will be unusable.

Microsoft is aware of this problem, but as it was only reported very late in the beta cycle for Windows Vista, this flaw was not fixed prior to release. At this point in time, we do not know when, or even whether, it will be fixed. So for now, the best advice is to not edit your saved ACLs. If you need to do so, save the file as a .bin file and use a hex editor, such as your favourite development environment.

Once you have saved an ACL using the /save switch, you can restore it using icacls.exe with the /restore switch. The restore command in its simplest form uses this syntax:

```
icacls <directory> /restore acls.bin
```

The restore procedure does not operate on files. To see this, look at the sequence of commands shown in **Figure 1**. Here I create a save file by pointing icacls.exe at a file. I then try to restore it by simply substituting /restore for /save. This fails because the restore command only operates on directories. The files it is supposed to modify are specified in the acls.bin file already. To restore the ACL, I point it at the directory instead of the file. This is what I do in the last command where I specify "." as the object to operate on.

Note also that the restore command must run elevated! You can run the save command from a command prompt running as a low admin or even a standard user, so long as you have the right to read the ACL. However, to restore the ACL you need to have a complete, unadulterated administrative token.

Figure 1 **Saving and restoring ACLs**

```
C:\Users\Jesper>icacls test.txt /save acls.bin
processed file: test.txt
Successfully processed 1 files; Failed processing 0 files

C:\Users\Jesper>icacls test.txt /restore acls.bin
test.txt\test.txt: The system cannot find the path specified
Successfully processed 0 files; Failed processing 1 files

C:\Users\Jesper>icacls . /restore acls.bin
processed file: .\test.txt
Successfully processed 1 files; Failed processing 0 files
```

Figure 2 **Substituting SIDs during restore**

```
C:\Users\Jesper>icacls test.txt
test.txt VistaRC2-Test\foo:(R,W)
    VistaRC2-Test\Jesper:(I)(F)
    NT AUTHORITY\SYSTEM:(I)(F)
    BUILTIN\Administrators:(I)(F)

Successfully processed 1 files; Failed processing 0 files

C:\Users\Jesper>icacls test.txt /save acls.bin
processed file: test.txt
Successfully processed 1 files; Failed processing 0 files

C:\Users\Jesper>icacls. /substitute foo bar /restore acls.bin
processed file: .\test.txt
Successfully processed 1 files; Failed processing 0 files

C:\Users\Jesper>icacls test.txt
test.txt VistaRC2-Test\bar:(R,W)
    VistaRC2-Test\Jesper:(I)(F)
    NT AUTHORITY\SYSTEM:(I)(F)
    BUILTIN\Administrators:(I)(F)

Successfully processed 1 files; Failed processing 0 files
```

Substituting SIDs

Another feature that's very useful in icacls.exe is the ability to replace the permissions for one user with permissions for a different user. This is done during restore using the /substitute switch. The documentation for the substitute switch says it needs an SID, but later it explains that this can actually also be a regular user name. Thus, the sequence shown in **Figure 2** works.

As you can see, I end up with the same ACL I had before, but the ACE that used to specify permissions for foo now specifies them for bar instead.

Change owner

The chown tool has been a staple on UNIX systems for years. Windows originally had no built-in way to change the owner of an object. Then came the setowner tool in the Resource Kit. Then we got the takeown.exe tool in Windows NT 4.0, but this utility only allowed you to take ownership, not grant it to others unless you had their password. Now icacls.exe gives you the built-in ability to set the owner of objects which you have permission to set the owner on:

```
C:\>icacls c:\test /setowner foo
processed file: c:\test
Successfully processed 1 files; Failed
processing 0 files
```

Unfortunately, icacls.exe can't show you the owner of an object. There is no way to actually see, from the command line, who the owner of an object is. Furthermore, if you save the ACL for an object, it does not save the owner of the object.

There is also a bug in icacls.exe in that it does not invoke SeTakeOwnershipPrivilege to change the owner. If you have the right to change the owner of an object based on the ACL, then icacls.exe works as it should. However, if you are an administrator, but you do not have permission to change the owner based on the object's ACL, you cannot use icacls.exe to do so because of that bug. In that case, you need to use the takeown.exe tool, which does invoke the SeTakeOwnershipPrivilege, but can only change the owner to your account or the Administrators group, not to an arbitrary account:

```
C:\>takeown /f c:\test

SUCCESS: The file (or folder): "c:\test" now
owned by user "JJ-VistaRTMTst\Jesper".
```

Of course, it should be noted that the subinacl tool, which is downloadable from the Microsoft Download center, also has a setowner switch. Subinacl actually works more intuitively than icacls in many cases, but it is also far more complicated to use.

Figure 3 **Grant, deny and remove operations**

```
C:\>icacls c:\test /grant foo:(F)
processed file: c:\test

Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test
c:\test JJ-VistaRTMTst\foo:(F)
    BUILTIN\Administrators:(I)(F)
    BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test /remove:g foo
processed file: c:\test
Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test
c:\test BUILTIN\Administrators:(I)(F)
    BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test /deny foo:(F)
processed file: c:\test
Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test
c:\test JJ-VistaRTMTst\foo:(N)
    BUILTIN\Administrators:(I)(F)
    BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test /remove:d foo
processed file: c:\test
Successfully processed 1 files; Failed processing 0 files

C:\>icacls c:\test
c:\test BUILTIN\Administrators:(I)(F)
    BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files
```

Find files for a particular user

Icacls has another useful function: it can find files that have permissions for particular users. For example:

```
C:\windows\system32>icacls "c:\program files"
/findsid jesper /t

SID Found: c:\program files\Passgen\
passgen.exe.
Successfully processed 1808 files; Failed
processing 0 files
```

This could be helpful if you are trying to find out what a particular user might have access to.

Resetting and changing ACLs

If an ACL has become corrupted or destroyed, icacls.exe has a way to reset it to inherit from its parent. This is something that would have been extremely useful during a zero-day security issue that hit in the autumn of 2006.

One method used to mitigate the issue in a Windows component was to deny Everyone access to the object. That part was easy, but removing that Access Control Entry (ACE) was nearly impossible using built-in tools in Windows XP and Windows Server 2003. However, in Windows Vista, you would simply run this command:

```
C:\windows\system32>icacls "c:\program files\
passgen\passgen.exe" /reset

processed file: c:\program files\passgen\
passgen.exe
Successfully processed 1 files; Failed
processing 0 files
```

Icacls.exe, of course, has all the standard grant/deny/remove operations. The only really new item in that list is remove. Using this switch, you can remove all allow ACEs, all deny ACEs, or both for a given user from a specific object or hierarchy. **Figure 3** shows examples of some different grant, remove and deny operations.

The ability to remove only the deny ACEs can be useful if you want to open access to an object up to a particular group or user.

Setting integrity levels

Icacls.exe also has the ability to display and set integrity levels. Windows Vista supports putting integrity labels on objects. Icacls.exe is the command line tool to use to do so:

```
C:\>icacls c:\test /setintegritylevel high

processed file: c:\test
Successfully processed 1 files; Failed
processing 0 files

C:\>icacls c:\test
c:\test BUILTIN\Administrators:(I)(F)
    BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
    Mandatory Label\High Mandatory Level:(NW)

Successfully processed 1 files; Failed
processing 0 files
```

Note that icacls.exe only displays an integrity label if an object has one explicitly set. Most files do not, so it is rare that you see one.

Finally, icacls.exe can verify that an object has a canonical ACL. As mentioned earlier, third-party tools have been known to put ACEs in the wrong order in ACLs. Icacls.exe can verify, and fix, those types of problems, as shown here:

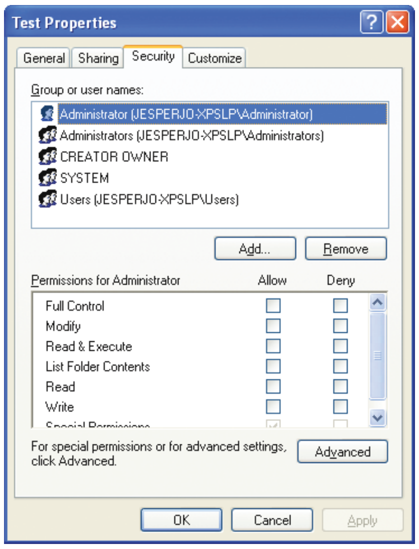
```
C:\>icacls c:\test /verify
processed file: c:\test

Successfully processed 1 files; Failed
processing 0 files
```

ACL UI

ACL UI, or the ACL User Interface, has been modified slightly from the one in Windows XP. **Figures 4** and **5** show the ACL UI dialog from Windows XP and Windows Vista, respectively.

As you can see, there are a few changes here. First, the dialog finally shows the object you are operating on very clearly, which can be useful if you are working on several objects at a time. Second, there is a hyperlink to some help at the bottom.

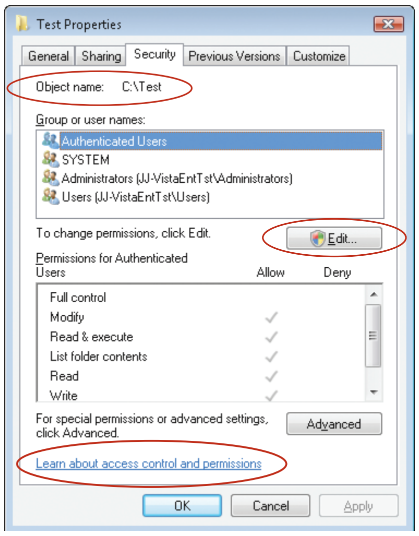
However, the most notable change is in the removal of the Add and Remove buttons, replacing them with an Edit button that is largely present in support of the User Account Control (UAC) in Windows Vista. As you can see by the shield on the button, the user that launched this dialog does not have permission to edit



Test Properties
General
Sharing
Security
Customize
Group or user names:
Administrator (JESPERJO-XPSLP\Administrator)
Administrators (JESPERJO-XPSLP\Administrators)
CREATOR OWNER
SYSTEM
Users (JESPERJO-XPSLP\Users)
Add...
Remove
Permissions for Administrator
Allow
Deny
Full Control
Modify
Read & Execute
List Folder Contents
Read
Write
For special permissions or for advanced settings, click Advanced.
Advanced
OK
Cancel
Apply

Figure 4 **ACL UI dialog in Windows XP**

the ACL, and therefore will require elevation before she can edit the permissions. Note that this would not be the default if you create a folder in the root of the C: drive and immediately check properties on it. As the owner of the folder you would have implicit permissions to edit the ACL and the shield would be missing. The shield signifies a COM Moniker, which is a mechanism used to launch an elevation prompt for a portion of a running process. If you click the Edit button, you get the familiar elevation dialog, and if you click Continue in the elevation dialog,



Test Properties
General
Sharing
Security
Previous Versions
Customize
Object name: C:\Test
Group or user names:
Authenticated Users
SYSTEM
Administrators (JJ-VistaEntTst\Administrators)
Users (JJ-VistaEntTst\Users)
To change permissions, click Edit.
Edit...
Permissions for Authenticated Users
Allow
Deny
Full control
Modify
Read & execute
List folder contents
Read
Write
For special permissions or advanced settings, click Advanced.
Advanced
Learn about access control and permissions
OK
Cancel
Apply

Figure 5 **The ACL UI dialog box in Windows Vista**

you get a dialog that is almost identical to the one you get in Windows XP by clicking the Add button. The ACL UI follows this dual-dialog concept in several places; you get one dialog until you elevate, and once you do, you get the familiar old dialog from earlier versions of Windows.

If you click the Advanced button and then the Auditing tab, you will always get an elevation button, as shown in **Figure 6**. You cannot modify auditing without elevating, even if you have full control over the object and you are the owner. That's because the ability to modify an object SACL is governed by the SE_SECURITY_NAME privilege, known as 'Manage auditing and security log' in the GUI tools. Only administrators have that right by default. However, the privilege is removed from an administrator in admin approval mode (when UAC is enabled), necessitating the elevation even if you are an administrator.

One final note on elevation needs when you modify ACLs: all these statements are contingent on your not disabling UAC. If you disable UAC, all behaviour reverts to what you had in Windows XP, with the exception of dialogs that look different. However, no elevation will be required, provided you log on as an administrator since your token will now have the Administrators SID in it all the time.

Other tools

Icacls.exe is a very useful tool, and a grand improvement over cacls.exe, but it still suffers from some shortcomings. Perhaps the most notable is that it cannot manage access to any objects other than files and directories. Windows Vista makes sparse changes to the ACLs of other objects compared to Windows XP, but there are still occasions when you want to manage ACLs on a service, a registry key, or even an Active Directory object.

If you are a command-line junkie, you need subinacl.exe to do this. Subinacl.exe is in the support tools, and is also available as a download (www.microsoft.com/uk/securitywatch).

I have to warn you though, subinacl.exe is not easy to use. In fact, it is downright obtuse at times. However, subinad.exe is an incredibly powerful tool for managing access control. Every power admin needs this tool.

Registry ACLs

The registry ACLs have undergone changes just like the file system ACLs. The changes are much smaller in scope than the changes to the file system, though. The most obvious difference from earlier versions of Windows is that, due to the deprecation of Power Users, almost all the Power User ACEs are gone. Power Users are not supposed to be any more powerful than any other users. It is a testament to just how complicated ACLs really are, though, that not all the ACEs for Power Users are actually gone. A few were, unfortunately, missed.

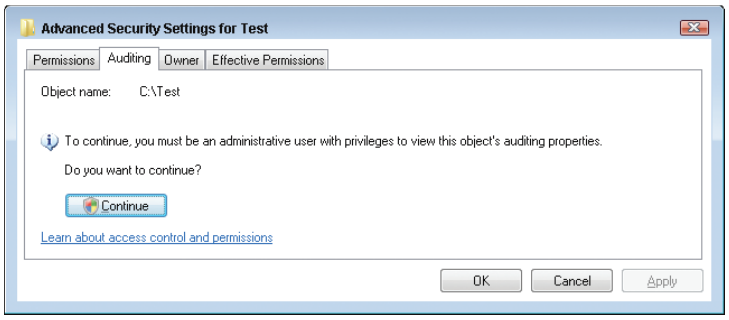
While you are looking at ACLs in the registry, in a few places you will see an ACE for a SID called RESTRICTED. This is not new to Windows Vista, but it is an interesting and not well understood SID. The RESTRICTED SID denotes any process that presents a restricted token. A restricted token is created by using a special feature of the CreateRestrictedToken API. Such a token has one or more restricting SIDs – SIDs that are used in a separate access check.

Let's say we have a process running with a restricted token. If that process attempts to access an object with an ACE for the RESTRICTED SID, Windows actually performs two access checks. The first is the normal access check. The second one works exactly like the first, but takes place only against the restricting SIDs in the token. Both access checks must pass.

Currently, there are several ACLs that use the RESTRICTED SID, particularly on the registry. A screen shot of such an ACL is shown in **Figure 7**.

At this time, few processes make use of the restricted token functionality, particularly with respect to restricting SIDs. One example of a process that does is the service process that hosts the Windows Firewall, the Base Filtering Engine and the Diagnostic Policy Service. It also uses a write restricted token. Based on my findings, only nine services currently use RESTRICTED and write restricted tokens in Windows Vista.

As with recent previous versions of Windows, the best practice with respect to registry permissions is to tread very carefully. Except in exceptional – and highly targeted – circumstances, do not modify permissions in the registry. Given the complicated inheritance model and the sensitive operations performed on the registry, you run an unacceptably high likelihood of fatal failure if you modify ACLs in the registry carelessly.

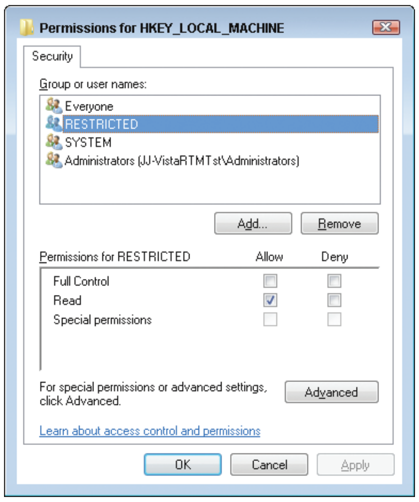

Advanced Security Settings for Test
Permissions
Auditing
Owner
Effective Permissions
Continue
Learn about access control and permissions
OK
Cancel
Apply

Object name: C:\Test

To continue, you must be an administrative user with privileges to view this object's auditing properties.

Do you want to continue?

Figure 6 **Modifying the audit settings in Windows Vista always requires elevation**


Permissions for HKEY_LOCAL_MACHINE
Security
Group or user names:
Add...
Remove
Advanced
Learn about access control and permissions
OK
Cancel
Apply

Everyone
RESTRICTED
SYSTEM
Administrators (JJ-VistaRTMTst\Administrators)

Permissions for RESTRICTED	Allow	Deny
Full Control	☐	☐
Read	✓	☐
Special permissions	☐	☐

For special permissions or advanced settings, click Advanced.

Figure 7 **The Registry ACLs include an ACE for RESTRICTED in several places**

Summary

As with most versions of Windows, there are some subtle changes to access control in Windows Vista. However, unlike some of the other recent versions, there are actually a lot of minor changes that add up to a fairly significant change in behaviour. UAC, in particular, required several changes, such as the integrity labels and modifications to the ACL UI. In addition, we have the first major clean-up of ACLs in recorded history.

In many ways, the default ACLs on Windows actually got simpler in Windows Vista, something that has never happened before. As with previous versions, however, you should really tread carefully around access control until you understand it thoroughly. This is especially true with new versions of the OS. Hopefully, using the tools outlined in this article, you will be able to make your exploration of ACLs far less painful. ■

JESPER M JOHANSSON *is a Principal Security Engineer working on software security issues and is a contributing editor to* TechNet Magazine. *He holds a PhD in MIS and has over 20 years experience in computer security. This column is adapted from Roger Grimes and Jesper Johansson's new book,* Windows Vista Security: Securing Vista Against Malicious Attacks *(Wiley, 2007).*