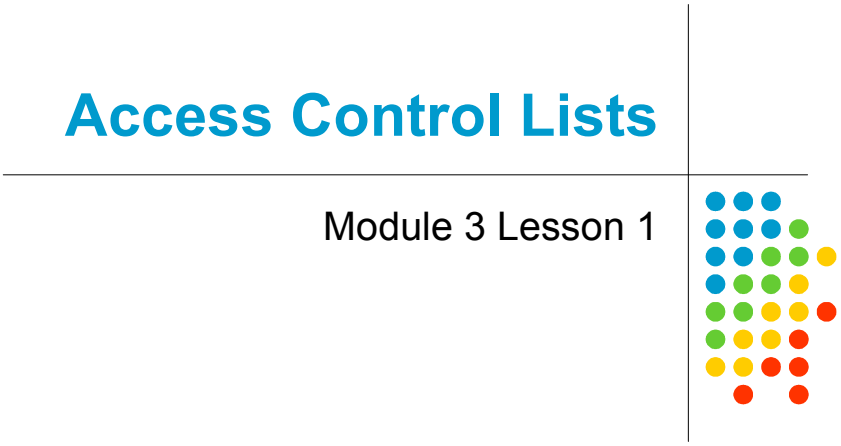
Module 3: Access Control Lists

Lesson 1: Access Control Lists



Overview

In this lesson you will be introduced to the main concepts of Access Control Lists (ACLs) in Windows.

What You Will Learn

After completing this lesson, you will be able to:

- Describe the Access Control features of Microsoft® Windows® 2000 and Microsoft® Windows Server™ 2003.
- Explain the implications of Access Control List inheritance.
- Explain how subordinate explicit grant overrides inherited denial.
- Explain Access Control Entry Inheritance for Active Directory Objects.
- Describe the purpose of AdminSDholder and SDprop.
- Modify the Filtered Properties of an Object.

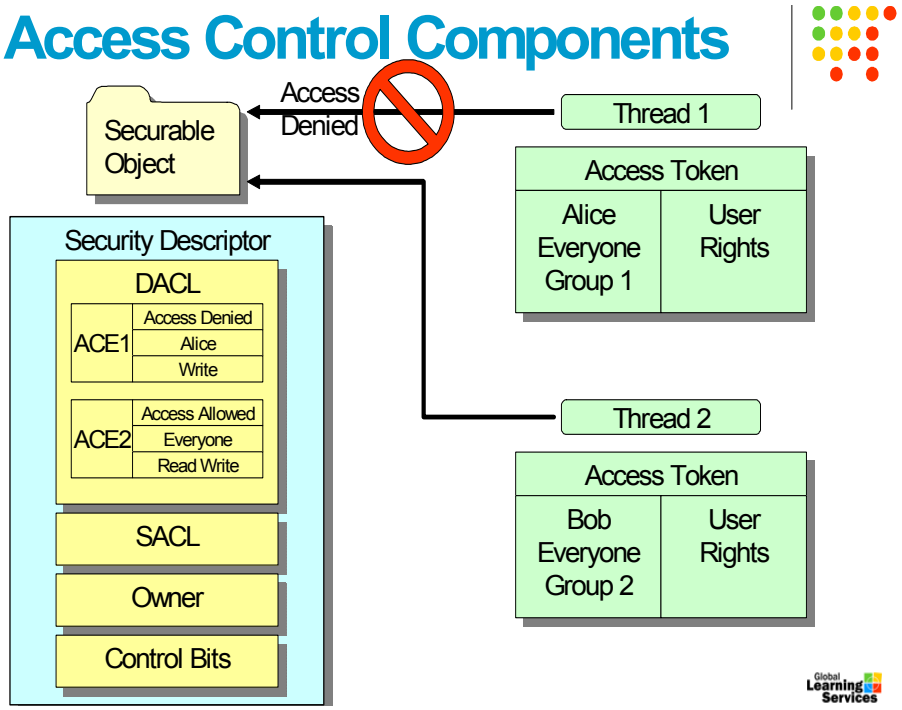
Related Topics Covered in this Lesson

- User Rights and Privileges

Recommended Reading

- Microsoft Corporation, *WindowsServer 2003 Deployment Guide (Windows Server 2003 Resource Kit)*, Microsoft Press, 2003. ISBN: 0-7356-1471-7
- Solomon, David A. & Russinovich, Mark E., *Inside Microsoft® Windows® 2000,* Microsoft Press, 2000. ISBN : 0-7356-1021-5

Access Control Components



There are two basic components of the access control model:

- Access tokens, which contain information about a logged-on user.
- Security descriptors, which contain the security information that protects a securable object.

When a user logs on to the system, it authenticates the user's account name and password or smartcard credentials. If the logon is successful, the system creates an access token. Every process executed on behalf of this user will have a copy of this access token. The access token contains security identifiers that identify the user's account and any group accounts to which the user belongs. The token also contains a list of the privileges held by the user or the user's groups. The system uses this token to identify the associated user when a process tries to access a securable object or perform a system administration task that requires privileges.

Every process has a primary token that describes the security context of the user account associated with the process. By default, the system uses the primary token when a thread of the process interacts with a securable object. Moreover, a thread can impersonate a client account. Impersonation allows the thread to interact with securable objects using the client's security context. A thread that is impersonating a client has both a primary token and an impersonation token.

When a securable object such as a folder on an NTFS volume is created, the system assigns it a security descriptor that contains security information specified by its creator, or default security information if none is specified. Applications can use Windows functions to retrieve and set the security information for an existing object.

A security descriptor identifies the object's owner and can also contain the following:

- A discretionary access control list (*DACL*) that identifies the users and groups allowed or denied access to the object.
- A system access control list (*SACL*) that controls how the system audits attempts to access the object.

- A set of control bits that qualify the meaning of a security descriptor or its individual components.

An access control list (ACL) contains a list of access control entries (ACEs). Each ACE specifies a set of permissions and contains a security identifier (SID) that identifies a trustee for whom the permissions are allowed, denied, or audited. A trustee can be a user account, group account, or logon session.

If the DACL belonging to an object's security descriptor is set to NULL, a null DACL is created. A null DACL grants full access to any user that requests it; normal security checking is not performed with respect to the object. A null DACL should not be confused with an empty DACL. An empty DACL is a properly allocated and initialized DACL containing no ACEs. An empty DACL grants no access to the object it is assigned to.

Each account has a unique SID issued by an authority, such as a Windows domain controller, and stored in a security database such as the Active Directory®. Each time a user logs on, the system retrieves the user's SID from the database and places it in the user's access token. This means that security descriptors are "rename safe," that is, they do not have to be altered when the trustees name attribute is changed, for example, when a user get married. In addition to the uniquely-created, domain-specific SIDs assigned to specific users and groups, there are well-known SIDs that identify generic groups and generic users. For example, the well-known SID, Everyone, identifies a group that includes all users.

A SID consists of the following components:

- The revision level of the SID structure.
- A 48-bit identifier authority value that identifies the authority that issued the SID, such as the domain.
- A variable number of sub-authority or relative identifier (RID) values that uniquely identify the trustee relative to the authority that issued the SID.

The combination of the identifier authority value and the sub-authority values ensures that no two SIDs will be the same, even if two different SID-issuing authorities issue the same combination of RID values. Each SID-issuing authority issues a given RID only once. SIDs are stored in binary format in a SID structure. The following standardized string notation for SIDs makes it simpler to visualize their components:

`S-R-I-S-S...`

In this notation, the literal character S identifies the series of digits as a SID, *R* is the revision level, *I* is the identifier-authority value, and *S*... is one or more sub-authority values. The following example uses this notation to display the well-known domain-relative SID of the local Administrators group:

```
S-1-5-32-544
```

In this example, the SID has the following components. The constants in parentheses are well-known identifier authority and RID values defined in the header file Winnt.h:

- A revision level of 1
- An identifier-authority value of 5 (SECURITY_NT_AUTHORITY)
- A first sub-authority value of 32 (SECURITY_BUILTIN_DOMAIN_RID)
- A second sub-authority value of 544 (DOMAIN_ALIAS_RID_ADMINS)

An access right is a bit flag that corresponds to a particular set of operations that a thread can perform on a securable object. For example, a registry key has the KEY_SET_VALUE access right, which corresponds to the ability of a thread to set a value under the key. If a thread tries to perform an operation on an object, but does not have the necessary access right to the object, the system does not carry out the operation. An access mask is a 32-bit value whose bits correspond to the access rights supported by an object.

For more information, see also… Platform Software Development Kit, October 2002

Closer Look: Token Size Issue

If a user is a member of many groups either directly or because of group nesting, Kerberos authentication may not work. The Group Policy object (GPO) may not be applied to the user and the user may not be validated to use network resources.

If the Administrator is a member of more than 70 to 80 groups there may be two additional symptoms:

- When logging on to a domain controller, an event ID 1000 will be generated in the system log.
- Running DCPromo to bring up a new domain controller will result in "Access Denied" when entering the domain admin credentials.

If the user has an associated logon script, the script may fail with one of the following error messages:

```
Not enough storage is available to complete this operation.
Hexadecimal values: 800a0007, 8007000e
Decimal values: -2146828281, -2147024882
```

Users will not be able to authenticate because the Kerberos token that is generated during authentication attempts has a fixed maximum size. Transports such as Remote Procedure Call (RPC) and HTTP rely on the MaxTokenSize value when they allocate buffers for authentication. In Windows 2000 (the original released version), the MaxTokenSize value is 8,000 bytes. In Windows 2000 Service Pack 2 (SP2) and Windows Server 2003, the MaxTokenSize value is 12,000 bytes.

If a user is a member of more than 120 groups, the buffer that is determined by the MaxTokenSize value is not large enough. As a result, users cannot authenticate, and they may receive an "out of memory" error message.

Most MaxTokenSize issues can be resolved by simply applying SP2 on the Windows 2000 domain controllers/key distribution centers (KDCs) (no need to apply to downlevel clients). This fix effectively reduces the size of tokens to 1/10 of their original size. This is accomplished by using RIDs instead of SIDs for all of the groups referenced in the token.

Prior to the fix, the following formula could be used to estimate token size.

```
2000 + (g * 80) where g = number of groups
```

After the fix, the formula looks more like the following.

```
1200 + (g * 8) + (n * 40)
```

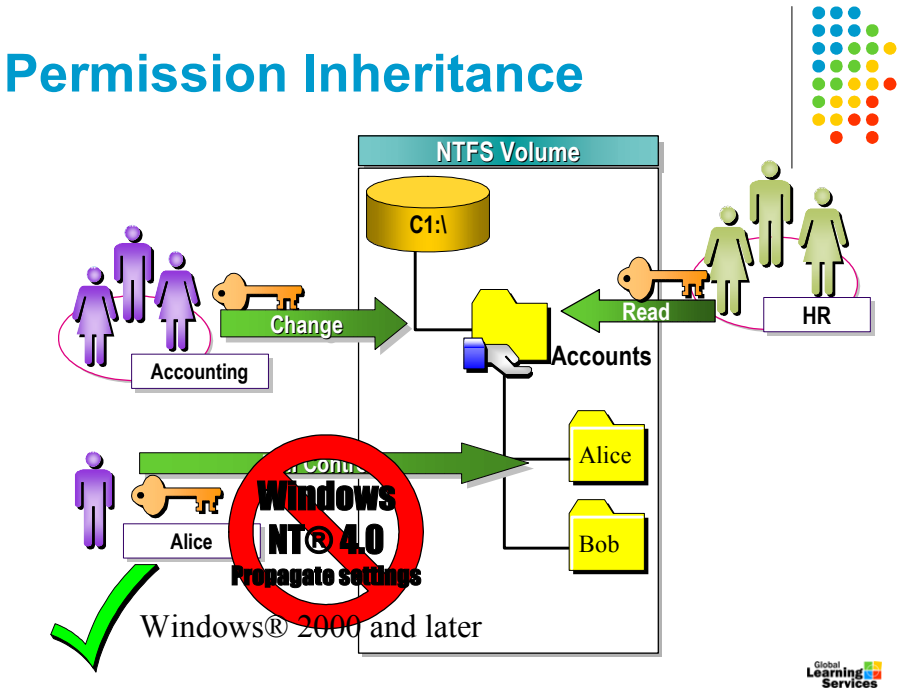
where g = groups in the user's account domain and n = groups not in user's account domain (and groups from SID history) As you can see, the big savings is in (g * 8) vs. (g * 80). The saving was accomplished by specifying only RIDs for groups that were in the user's account domain. For example, if the SID was S-1-5-21-1463437245-1224812800-863842198-1128, only the RID portion (1128) would be included in the token.

Note:

In many scenarios, Windows NTLM authentication works as expected; you may not see the Kerberos authentication problem without analysis. However, scenarios in which Group Policy settings are applied may not work as expected.

For more information, see the following Knowledge Base article: 327825 "New Resolution for Problems That Occur When Users Belong to Many Groups."

Access Control Entry Inheritance Changes



Inheritance

Inheritance Behaviour in Windows NT 4.0

Microsoft ® Windows NT® 4.0 allows for the propagation of access control entry (ACE) changes to subordinate securable objects only at the time the object is created, or when you apply a new access control list (ACL) to the parent container object. Furthermore, Windows NT 4.0 does not differentiate between Aces that are inherited from a superior object, and the Aces applied directly to an object. The only way in the unique identifier (UI) to apply permissions to subordinate objects is to reset them.

For example, the following steps show the limitations of inheritance behavior in Windows NT 4.0:

1. An administrator creates a file server share called Accounting. All members in the Accounting department (whose user accounts are members in a local group named Accounting) have Change permissions to the Accounting share. The location of the share is E:\Files\Accounting. The administrator sets permissions using NTFS file system permissions.
2. In the permissions dialog box, an administrator selects the Replace Permissions on subdirectories check box to configure NTFS permissions inheritance, and then when prompted, the administrator replaces the entire ACL for each subordinate file system object.
3. In the Accounting share is a folder for each user. Each user has been given Full Control permissions on their own folder. The Full Control permissions are assigned manually; using NTFS folder permissions directly on the folder file system objects.

4. At some future date, the administrator provides a "human resources" oversight group permissions to read files in the Accounting share. If they modify NTFS permissions at the root of the share, and propagate settings using the Replace Permissions On sub folders check box, they delete all of the previously configured subordinate ACLs, because Windows NT 4.0 inheritance cannot distinguish between inherited ACEs and directly applied ACEs.

Inheritance Behaviour in Windows 2000 and later

Microsoft® Windows® 2000 and later support automatic propagation of inheritable ACEs. In addition, ACEs that are directly applied to file system objects are given a higher priority than inherited ACEs. The directly applied ACEs are applied before any conflicting inherited ACEs. Using the scenario detailed above, the following steps show the new behaviour:

1. The administrator creates the Accounting file share at E:\Files\Accounting, and then assigns the accounting group Change permissions.
2. By default, every subordinate file system object inherits the permissions set on the parent level container.
3. Users can add explicit ACL entries directly upon subordinate objects. Because these ACEs are explicit, they have precedence over inherited ACEs.
4. When making changes to the ACL on the top level folder, the explicit ACLs defined on the subordinate file system objects are not deleted.

Propagating ACLs

When a user or group is given permissions in the ACL Editor dialog box, by default these permissions are restricted to the container object itself, and the child objects within the container are not affected by the permission change. These child objects do, however, have default explicit permissions of their own. For example, an administrator creates an Organizational Unit within the domain named "OU1." Within OU1, several user objects exist. The administrator adds a user to the permissions list for OU1 and grants that user Full Control. When the user logs on and attempts to modify one of the user objects within OU1, the user receives an access denied error message. This is because the user was only given permissions on the container object and not on the child objects of that container.

The administrator can either:

- Change the scope of the user's permissions on the container object, and allow the child objects to inherit the permissions from the parent container.

-or-

- Add the user to the permissions list of each object within the container.

By default, child objects of a container will allow the inheritance of permissions from the parent container. When adding or modifying a permission on the parent container, perform the following steps to allow those permissions to propagate to child objects.

1. Open the **Properties** for the container object in the Directory, and select the **Security** tab.
2. Click **Advanced**. This will display the Advanced Security Settings dialog box.
3. Select the user or group to modify the permissions for, and click **Edit**. If the user is not already present, click **Add** to add the user before continuing.
4. In the drop down menu for "Apply Onto," select **This object and all child objects**, and customize the permissions appropriately.

5. Clear the “Apply these permissions to objects and/or containers within this container only” check box. This setting determines if the permissions will only be propagated to the immediate child objects of this container or allowed to flow past the immediate children to other containers within the parent.
6. Click **OK** and close the remaining dialog windows.

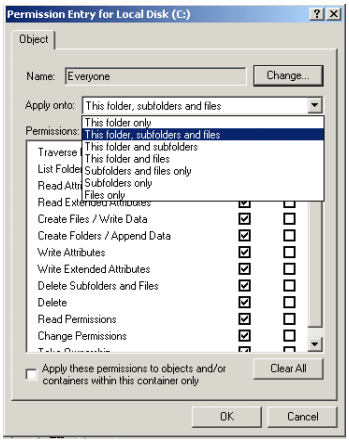
As stated above, child objects of a container will allow the inheritance of permissions from the parent container by default. On Windows 2000, the administrator can confirm this by opening the **Properties** for a given object, selecting the **Security** tab, and noting the state of the "Allow inheritable permissions from parent to propagate to this object" check box at the bottom of the property page. To view the inherited permissions, click **Advanced**, and note that the user who was given permissions at the container level is listed in the permissions list, but with a discoloured icon.

On Microsoft® Windows Server™ 2003, the administrator must click the **Advanced** button on the Security page in order to view the “Allow inheritable permissions from the parent to propagate to this object and all child objects” check box.

To disable a particular object's inheritance of the parent container's permissions, clear the "Allow inheritable permissions" check box. When this is done, the users and groups that were given permissions at the parent container level are now displayed as active entries in the permissions list. The administrator may remove these entries before closing the dialog box.

Understanding Container Access Inheritance Flags

Inheritance Flags



- IO: Inherit Only - This flag indicates that this ACE does not apply to the current object.
- CI: Container Inherit - This flag indicates that subordinate containers will inherit this ACE.
- OI: Object Inherit - This flag indicates that subordinate files will inherit the ACE.
- NP: Non-Propagate – This flag indicates that the subordinate will not propagate the ACE any further

For the purposes of access control inheritance, there are two types of objects in Windows NT and later: containers and non-containers. Access control entries on container objects can be configured to propagate to subordinate objects. This propagation is accomplished using container access inheritance flags, which are written to specific access control entries that are applied on the container itself.

In the file system, administrators can configure this information by accessing the advanced dialog box of the Access Control Editor. Container inheritance is present in the Apply Onto box of the box displaying ACE entries. When an administrator adds a new ACE to the access control list, he or she can select the scope of the entry's inheritance. The following are specific to the NTFS file system:

- "This folder only" Apply Onto value, no ACE flags: No inheritance applies to ACE.
- "This folder, subfolders, and files" Apply Onto value, (OI), (CI) ACE flags: All subordinate objects inherit this ACE, unless they are configured to block ACL inheritance altogether.
- "This folder and subfolders" Apply Onto value, (CI) ACE flag: ACE propagates to subfolders of this container, but not to files within this container.
- "This folder and files" Apply Onto value, (OI) ACE flag: ACE propagates to files within this container, but not to subfolders.
- "Subfolders and files only" Apply Onto value, (IO), (CI), (OI) ACE flags: ACE does not apply to this container, but does propagate to both subfolders and files contained within.
- "Subfolders only" Apply Onto value, (IO), (CI) ACE flags: ACE does not apply to this container, but propagates to subfolders. It does not propagate to contained files.
- "Files only" Apply Onto value, (IO), (OI) ACE flags: ACE does not apply to this container, but propagates to the files it contains. Subfolders do not receive this ACE.
- "Apply these permissions to objects and/or containers within this container only" Apply Onto value, adds (NP) ACE flag: This flag limits inheritance only to those sub-objects that are immediately subordinate to the current object.

The ACL flags have the following meanings:

- IO: Inherit Only – This flag indicates that this ACE does not apply to the current object.
- CI: Container Inherit – This flag indicates that subordinate containers will inherit this ACE.
- OI: Object Inherit – This flag indicates that subordinate files will inherit the ACE.
- NP: Non-propagate – This flag indicates that the subordinate object will not propagate the inherited ACE any further.

Subordinate Explicit Grant Overrides Inherited Denial

When is a DENY not a DENY?

- Inherited ACE vs. Explicit ACE
- Explicit entries always take precedence over inherited entries.
- "DENY" access control entries may be pre-empted by the existence of explicit "ALLOW" entries on subordinate objects.
- This is a different outcome from that in Microsoft® Windows NT® 4.0.

The ACL inheritance model introduced in Windows provides for the dynamic inheritance of ACEs from parent container objects. This dynamic behaviour allows ACEs to be changed on a superior container, and to have those changes automatically propagated to subordinate objects. In fact, such detachment (also known as ACL "protection") is required if one wants to remove superior ACEs from applying to the subordinate file system objects altogether.

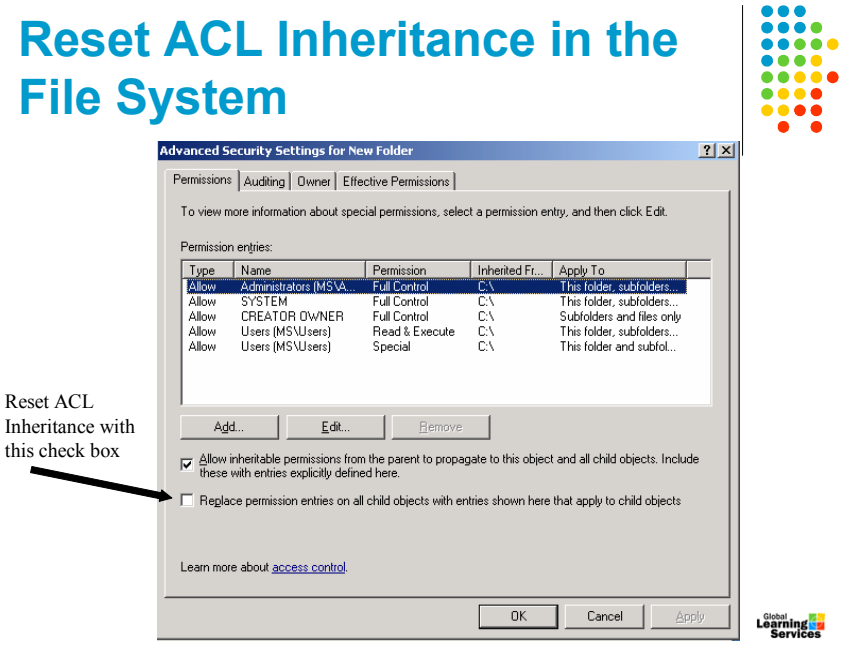
In addition, the access control mechanism distinguishes between inherited access control entries and explicit entries, so that explicit entries always take precedence over inherited entries.

This may lead to a situation in which inherited "DENY" access control entries may be pre-empted by the existence of explicit "ALLOW" entries on subordinate objects. This is a different outcome from that in Microsoft Windows NT 4.0.

Windows NT 4.0 Server does not implement ACL inheritance; instead, it gives administrators the ability to propagate explicit access control entries down a file system hierarchy. This means that all access control entries are explicit entries. DENY entries are always sorted to the top of the list, and have precedence over ALLOW entries.

This inheritance is also of concern with respect to the security of Active Directory objects. Active Directory objects automatically have explicit access control entries assigned upon their creation, based on settings in the Schema. For example, suppose an administrator creates an Organisational Unit with specific DENY permissions. If users are created within that container, they have the default explicit ALLOW permissions that override the inherited permissions.

Reset ACL Inheritance in the File System



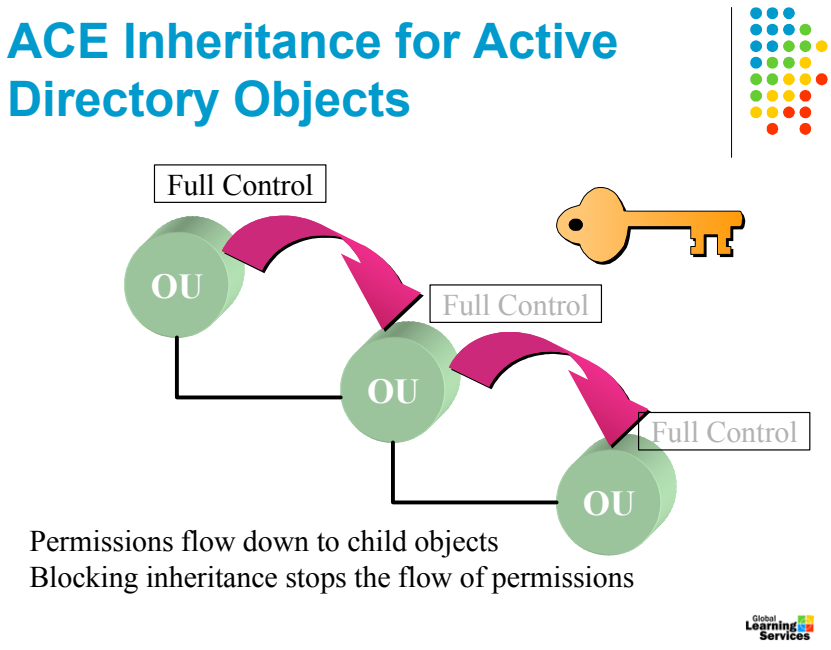
Administrators may need a way to blast through a file system and reset inheritance. This is provided through the "Replace permissions entries on all child objects" option so child objects will default back to inheriting permissions from their parent object.

The "Replace permissions entries on all child objects" option is available for file system object permissions, and for auditing. To reactivate ACL inheritance and remove all custom assigned access control entries for a complete file system tree, perform the following tasks:

1. Open the **Properties** dialog box for the top-most object of the file system tree you want to reset, and select the **Security** tab.
2. Click **Advanced** to display the Advanced Access Control Settings dialog box.
3. Select (check) "Replace permissions entries on all child objects" to reset all subordinate file system objects. To reset audit permission, a "Replace auditing entries on all child objects" option exists under the Auditing tab.
4. A confirmation dialog box will present itself upon application of the new Access Control setting, to ensure awareness that all subordinate, explicitly defined access control entries will be destroyed by this action. Only inherited access control entries and legacy access control entries (entries that are left from a pre-upgrade installation of Microsoft Windows NT) will remain on subordinate objects.

Resetting permissions is the only mechanism to change permissions on subordinate objects in Windows NT 4.0, and takes place by default. However, this is a destructive mechanism which alters all lower level entries. To get the same effect in Windows 2000 and later, the Reset check box needs to be selected. This option will fail if the operator does not have appropriate permissions at the lower level.

ACE Inheritance for Active Directory Objects



Active Directory objects have security settings similar to security settings for file system objects on volumes using the NTFS file system. Administrators can also apply access control permissions to properties of a specific Active Directory object. This functionality provides the administrator detailed control over what users can do in their environment.

This information is configured in the Permission Entry dialog box. To view the Permission Entry dialog box, first make sure that **Advanced Features** from the **View** menu in Active Directory Users and Computers is enabled. Right-click the object in question, click **Properties**, click the **Security** tab, click **Advanced**, and then click **Edit** on the **View** menu.

In the file system, there are two types of objects: files and folders. In the directory service, there are many types of object such as users, computers and printers. The directory service can be extended to store additional classes of objects.

Active Directory objects have all of the inheritance options present for file system objects. They also have more options in the Apply Onto box: the Object Specific ACE. This flag, when set, dictates that this ACE applies only if the object type of the subordinate object is an identical match with the object type listed in the Object Specific ACE.

Active directory can have inheritance based on whether an object is a child object or a container. It can also be based on the specific type of container that an object resides in. This initial set of permissions for a class of objects is gathered from the schema.

For example, Active Directory® Organizational Units are container objects that can contain contact, computer, group, and site container objects, as well as a long list of other object types. It is possible, using the ACL editor in the context of the Active Directory, to define access control list entries for which inheritance is determined by the specific sub-object type. In this example, therefore, it is possible to create an access control entry on an organizational unit that only grants inheritance to subordinate contact objects.

AdminSDHolder

AdminSDHolder

- Protected Groups
  - Delegation
  - Inheritance
- AdminSDHolder
- SDProp thread

After upgrading to Windows Server 2003, the following symptoms may be experienced:

- Delegated permissions are not available to all users in an organizational unit.
- Inheritance is automatically disabled on some user accounts approximately once per hour.
- Users, who previously had delegated permissions, no longer have them.

Note:

This problem may also occur after you apply the hotfix described in Microsoft Knowledge Base article 327825 to Microsoft Windows 2000 Server.

When delegating permissions using the Delegation of Control wizard, these permissions rely on the user object that inherits the permissions from the parent container. Members of protected groups do not inherit permissions from the parent container. As a result, if permissions are set using the Delegation of Control wizard, these permissions are not applied to members of protected groups.

Note:

Membership in a protected group is defined as either direct membership or transitive membership using one or more security or distribution groups. Distribution groups are included because they can be converted to security groups.

To work around this problem, use one of the following methods.

Method 1: Make Sure Members Are Not Members of a Protected Group

If you using permissions that are delegated at the organizational unit level, make sure that all users who require the delegated permissions are not members of one of the protected groups. For users who were previously members of a protected group, the inheritance flag is not automatically reset when the user is removed from a protected group. It is necessary to restore inheritance on the user manually by using either Active Directory Users and Computers or a script that uses Dsacls.exe. This method is preferred and does not weaken existing security.

Method 2: Enable Inheritance on the AdminSDHholder Container

If inheritance is enabled on the adminSDHolder container, all members of the protected groups have inherited permissions enabled. For security reasons, Microsoft does not recommend this method.

If inheritance is enabled on the adminSDHolder container, one of the protective access control list mechanisms is disabled. The default permissions are applied. However, all members of protected groups inherit permissions from the organizational unit and any parent organizational units if inheritance is enabled at the organizational unit level.

To provide inheritance protection for administrative users, move all administrative users (and other users who require inheritance protection) to their own organizational unit. At the organizational unit level, remove inheritance and then set the permissions to match the current ACLs on the adminSDHolder container. Because the permissions on the adminSDHolder container may vary (for example, Microsoft® Exchange Server adds some permissions or the permissions may have been modified), review a member of a protected group for the current permissions on the adminSDHolder container.

You can enable inheritance or change the permissions on protected groups by editing the security of the adminSDholder container. The path of the adminSDHolder container is

```
CN=AdminSDHolder,CN=System,DC=<MyDomain>,DC=<Com>
```

Adminsdholder

Active Directory uses a protection mechanism to make sure that ACLs are set correctly for members of sensitive groups. The operations master compares the ACL on the user accounts that are members of protected groups against the ACL on the AdminSDHolder object.

Note that the user interface (UI) does not display all permissions on the adminSDHolder container. When viewing the ACL on the Security tab of the AdminSDholder object properties in the Active Directory Users and Computers snap-in or ADSI Edit tool, it is not possible to configure fields that are associated with user accounts or groups. Advanced fields, such as Change Password, Reset Password, Receive As, and Send As are not displayed as expected. This behaviour occurs because the AdminSDHolder object is a container object that is used only as a template to store permissions. Even though the permissions that are applied to it are intended to be applied to the user or group objects, the ACL editor only displays the ACE for the type of object that it is currently editing (the container object). It is necessary to use Dsacls.exe to view all permissions on the adminSDHolder container.

Active Directory uses the Security Descriptor propagator (SDProp) background process that runs every hour to implement the protection of administrative groups. This process first computes the set of memberships including nested groups for all administrative groups. It then examines each object in the list that it has created and evaluates whether the security descriptor on the objects is a well-known protected security descriptor. If the well-known protected security descriptor is not set, it sets this security descriptor on the object.

This task runs only on the primary domain controller (PDC) emulator Flexible Single Master Operation (FSMO) role holder. It is possible to force the SDProp process to execute manually using LDP.

Every hour, the PDC emulator compares the ACL on the user accounts present for its domain in Active Directory that are in administrative groups against the ACL on the following object:

CN=AdminSDHolder,CN=System,*DC=MyDomain,DC=Com*

(Replace "*DC=MyDomain,DC=Com*" in this path with the distinguished name (DN) of your domain.)

If the ACL is different, the ACL on the user object is overwritten to reflect the security settings of the AdminSDHolder object (which includes disabling ACL inheritance). This process protects these accounts from being modified by unauthorized users if the accounts are moved to a container or organizational unit where a malicious user has been delegated administrative credentials to modify user accounts.

Note:

When a user is removed from the administrative group, the process is not reversed and must be manually changed.

The following list describes the protected groups in Windows 2000:

- Enterprise Admins
- Schema Admins
- Domain Admins
- Administrators

The following list describes the protected groups in Windows Server 2003 and in Windows 2000 after applying the 327825 hotfix:

- Administrators
- Account Operators
- Server Operators
- Print Operators
- Backup Operators
- Domain Admins
- Schema Admins
- Enterprise Admins
- Cert Publishers

Additionally the following users are also considered protected:

- Administrator
- Krbtgt

Note that membership in distribution groups does not populate a user token. As a result, you cannot use tools such as "whoami" to successfully determine group membership.

For more information, see the following Knowledge Base article: 232199 “Description and Update of the Active Directory AdminSDHolder Object.”

For more information, see the following Knowledge Base article: 318180 “AdminSDHolder Thread Affects Transitive Members of Distribution Groups.”

For more information, see the following Knowledge Base article: 817433 "Delegated Permissions Are Not Available and Inheritance Is Automatically Disabled."

Container Inheritable ACEs

Container Inheritable ACEs

- Excessive use of Container Inherit ACEs can cause growth in the Active Directory® database
  - Security Descriptor Propagation
  - Root Naming Context
- Single Instance Store
  - Objects store links to security descriptor
  - Only requires local domain controller upgrade
  - De-fragment to reclaim disk space

Setting ACL and audit operations on the schema, configuration and particularly domain naming contexts can cause massive increase in the size of Active Directory databases, particularly those in forests hosting a large number of objects. This is especially true when the ACE applies to all objects or objects of a particular class in a container. An example might be the right to reset user passwords against a particular organizational unit.

Active Directory security descriptor propagation copies inherited ACEs to all child objects, which makes access checks very fast. This is because all the security information can be discovered by looking directly at the object rather than traversing up the directory checking each container object. Class specific ACEs are also copied to all child objects. For example, an ACE used to delegate the right to reset user passwords is also copied to computer and container objects in that container. This propagation means that each object will grow in size when new ACEs are set on their containers. This will have the greatest impact if set on a root naming context such as a domain. On an organizational unit, there will be a lower impact dependent on the number of objects in the organizational unit.

In Windows 2000 it is recommended to avoid using container inheritable ACEs on the root of the domain naming context and to add organizational units as appropriate for users, groups and computers. The container inheritable ACEs on these organizational units will have less impact.

In Windows Server 2003 the Active Directory makes use of a Single Instance Store (SIS) to improve the storage efficiency of objects and their associated security data. The SIS reviews existing permissions on objects stored in Active Directory, and then applies a more efficient security descriptor on those objects. The SIS starts automatically (identified by event 1953 in the directory service event log) when upgraded domain controllers first start the Windows Server 2003 operating system.

After SIS processing has completed, objects have links to security descriptors rather than storing unique copies. If a container inheritable ACE changes subsequently, then only one security descriptor is modified, thus having minimal impact on the size of the database.

The SIS does *not* require all domain controllers in the domain to be upgraded to Windows Server 2003 to take advantage of the benefits it brings. SD propagation takes place on the local domain controller and is transparent to other domain controllers in the domain or forest.

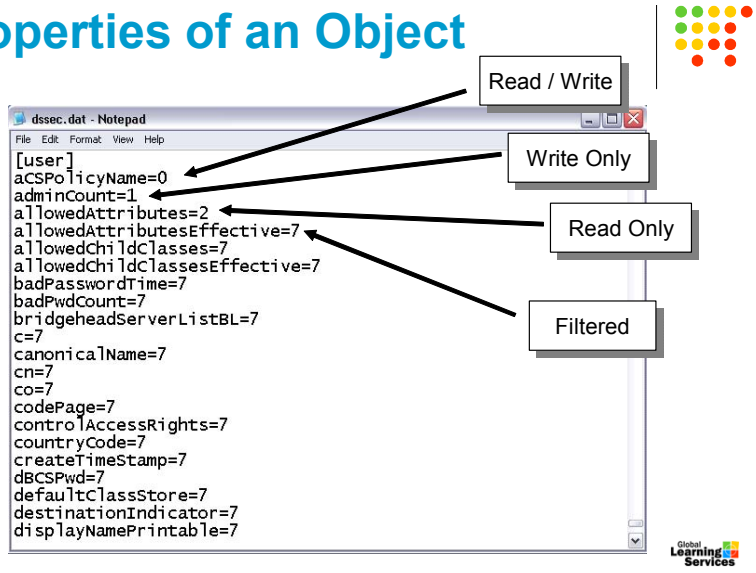
The improved security descriptor store benefits only occur when event ID 1966 is logged in the directory service event log:

```
Event Type: Information
Event Source: NTDS SDPROP
Event Category: Internal Processing
Event ID: 1966
Date: MM/DD/YYYY
Time: HH:MM:SS AM|PM
User: NT AUTHORITY\ANONYMOUS LOGON
Computer: <computername>
Description: The security descriptor propagator has
completed a full propagation pass.
Allocated space (MB): XX Free space (MB): XX
This may have increased free space in the Active Directory
database. User Action Consider defragmenting the database
offline to reclaim the free space that may be available in
the Active Directory database. For more information, see
Help and Support Center at
http://go.microsoft.com/fwlink/events.asp.
```

This event message indicates that the single instance store operation has completed and serves as an indicator to the administrator to perform offline de-fragmentation of the Ntds.dit using NTDSUTIL.EXE.

The offline defragmentation after SIS processing can reduce the size of a Windows Server 2003 Ntds.dit file by up to 40 percent, improves Active Directory performance, and updates the pages in the database for more efficient storage of Link Valued attributes. The performance impact of the SIS adds additional processing while a security descriptor is searched for but the more efficient memory use offsets this overhead.

How to Modify the Filtered Properties of an Object



The Per-Property Permissions tab for a user object that you view through Active Directory Users and Computers may not display every property of the user object. This is because the user interface for access control filters out object and property types to make the list easier to manage. While the properties of an object are defined in the schema, the list of filtered properties that are displayed is stored in the Dssec.dat file that is located in the %systemroot%\System32 folder on all domain controllers. It is possible to edit the entries for an object in the file to display the filtered properties through the user interface.

A filtered property looks like this in the Dssec.dat file:

```
[User]
propertyname=7
```

To display the read and write permissions for a property of an object, it is possible to edit the filter value to display one or both of the permissions. To display both the read and write permission for a property, change the value to zero (0):

```
[User]
propertyname=0
```

To display only the write permission for a property, change the value to 1:

```
[User]
propertyname=1
```

To display only the read permissions for a property, change the value to 2:

```
[User]
propertyname=2
```

After editing the Dssec.dat file, quit and restart Active Directory Users and Computers to see the properties that are no longer filtered.

Lesson 2: ACL Management Tools

ACL Management Tools

Module 3 Lesson 2

Overview

A variety of ACL Management tools are available in Windows Server 2003 for managing file system and directory service objects.

What You Will Learn

After completing this lesson, you will be able to:

- Use DSACLS.EXE
- Use ACLDIAG.EXE
- Use XCACLS
- Use SHOWACLS
- Use SUBINACL

Directory Service ACLs

Dsacls

Dsacls displays or modifies permissions (or ACLs) of an Active Directory object. This tool facilitates management of ACLs for directory services. Dsacls enables administrators to query and manipulate security attributes on Active Directory objects. It is the command-line equivalent of the Security page on various Active Directory snap-in tools.

Dsacls provides security configuration and diagnosis functionality on Active Directory objects from the command prompt. This is useful for setting permissions on objects or properties that have no user interface such as those created by schema extensions.

File required

Dsacls.exe

Source

Support Tools

Usage

DSACLS object [/I:TSP] [/N] [/P:YN] [/G <group/user>:<perms> [...]]
[/R <group/user> [...]] [/D <group/user>:<perms> [...]]
[/S] [/T] [/A]

object Path to the Active Directory object for which to display or manipulate the ACLs. Path is the RFC 1779 format of the name, as in:

CN=John Doe,OU=Software,OU=Engineering,DC=Widget,DC=com

A specific Active Directory can be denoted by prepending \\server\ to the object, as in:
\\ADSERVER\CN=John Doe,OU=Software,OU=Engineering,DC=Widget,DC=US

no options Displays the security on the object.

/I Inheritance flags:
T: This object and sub objects
S: Sub objects only
P: Propagate inheritable permissions one level only.

/N Replaces the current access on the object, instead of editing it.
/P Mark the object as protected
Y:Yes
N:No
If /P option is not present, current protection flag is maintained.

/G <group/user>:<perms>
Grant specified group (or user) specified permissions.
See below for format of <group/user> and <perms>

/D <group/user>:<perms>
Deny specified group (or user) specified permissions.
See below for format of <group/user> and <perms>

/R <group/user>
Remove all permissions for the specified group (or user).
See below for format of <group/user>

/S Restore the security on the object to the default for
that object class as defined in Active Directory Schema.

/T Restore the security on the tree of objects to the
default for the object class.
This switch is valid only with the /S option.

/A When displaying the security on an Active Directory object, display
the ownership and auditing information as well as the permissions.

<user/group> should be in the following forms:
group@domain or domain\group
user@domain or domain\user

<perms> should be in the following form:

[Permission bits];[Object/Property];[Inherited Object Type]

Permission bits can have the following values concatenated together:

Generic Permissions

GR Generic Read
GE Generic Execute
GW Generic Write
GA Generic All

Specific Permissions

SD Delete
DT Delete an object and all of its children
RC Read security information
WD Change security information
WO Change owner information
LC List the children of an object

CC Create child object
DC Delete a child object

For the CC and DC permissions, if [Object/Property] is not specified to define a specific child object type, they apply to all types of child objects. Otherwise they apply to that specific child object type.

WS Write to self object

Meaningful only on Group objects and when [Object/Property] is filled in as "member."

WP Write property
RP Read property

For these two permissions, if [Object/Property] is not specified to define a specific property, they apply to all properties of the object. Otherwise they apply to that specific property of the object.

CA Control access right

For this permission, if [Object/Property] is not specified to define the specific "extended right" for control access, it applies to all control accesses meaningful on the object. Otherwise it applies to the specific extended right for that object.

LO List the object access. Can be used to grant list access to a specific object if List Children (LC) is not also granted to the parent. Can also be denied on specific objects to hide those objects if the user or group has LC on the

well can denied on specific objects to hide those objects
parent.
NOTE: Active Directory does NOT enforce this permission
by default, it has to be configured to start checking for
this permission.

[Object/Property]
Represents the display name of the object type or the property.
For example, "user" (without the quotation marks) is the display name for user objects and "telephone number" (without the quotation marks) is the display name for telephone number property.

[Inherited Object Type]
Represents the display name of the object type by which the permissions are expected to be inherited. The permissions MUST be Inherit Only.
NOTE: This must only be used when defining object specific permissions that override the default permissions defined in the Active Directory schema for that object type. USE THIS WITH CAUTION and ONLY IF YOU UNDERSTAND object specific permissions.

Examples of a valid <perms> would be:

SDRCWDWO;;user
means:
Delete, Read security information, Change security information and Change ownership permissions on objects of type "user."

CCDC;group;
means:
Create child and Delete child permissions to create/delete objects of type group.

RPWP;telephonenumber;
means:
Read property and write property permissions on telephone number property.

You can specify more than one user in a command.

For more information, see the following Knowledge Base article: 281146 “How to Use Dsacls.exe in Windows 2000.”

ACL Diagnostics

AclDiag

AclDiag displays a detailed description of the security settings on an object in Active Directory and verifies the security settings against delegation template settings. This command-line tool helps diagnose and troubleshoot problems with permissions on Active Directory objects. It reads security attributes from access control lists (ACLs) and writes information in either readable or tab-delimited format. The latter can be uploaded into a text file for searches on particular permissions, users, or groups, or into a spreadsheet or database for reporting. The tool also provides some simple cleanup functionality.

With AclDiag, It is possible to:

- Display the access control entries (ACEs) in the ACL, and inheritance and audit settings.
- Compare the ACL on a directory services object to the permissions defined in the schema defaults.
- Check or fix standard delegations performed using templates from the Delegation of Control wizard in the Active Directory Users and Computers snap-in.
- Display the effective permissions granted to a specific user or group, or to all users and groups that show up in the ACL.

AclDiag displays only the permissions of objects the user has the right to view. Because Group Policy objects are virtual objects that have no distinguished name, this tool cannot be used on them.

File required

Acldiag.exe

Source

Support Tools

Usage

acldiag ObjectDN [/schema] [/chkdeleg] [/geteffective:{UserOrGroup | *}]
[/fixdeleg] [/skip] [/tdo]

ObjectDN	The distinguished name (DN) of the object that is checked.
/schema	Checks whether the security on the object includes the default security settings that are defined in the Active Directory schema.
/chkdeleg	Checks whether the security on the object includes any of the delegation templates currently in use by the Delegation Wizard.
/geteffective:{UserOrGroup | *}	Displays the effective rights of the specified user or group. If * is specified, the effective rights of all users and groups in the object's access control list are displayed.
/fixdeleg	Fix delegation templates reported by /chkdeleg as not completely applied. Requires the /chkdeleg option.
/skip	Do not display the security description.
/tdo	Output a tab delimited output.

Examples:

To display the security settings for the user MikeDan in the domain Contoso.com, type:

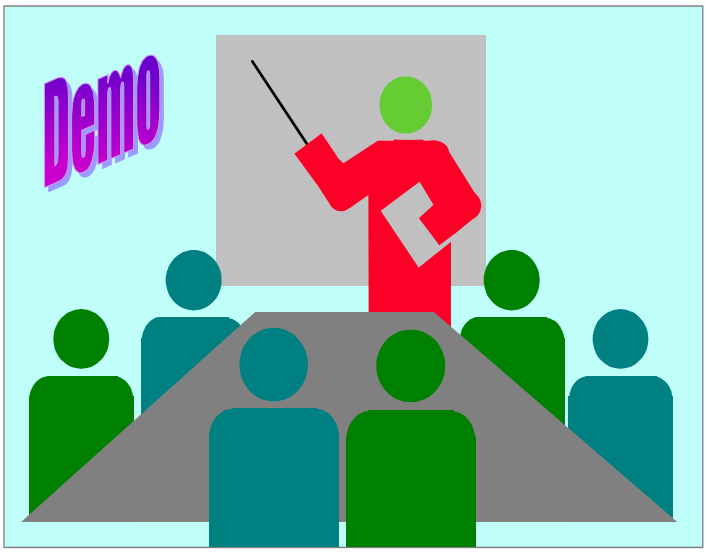
```
acldiag CN=MikeDan,CN=Users,DC=Contoso,DC=com
```

To determine whether delegation templates are applied to user MikeDan in the Contoso.com domain, and re-apply any incomplete templates, type:

```
acldiag CN=MikeDan,CN=Users,DC=Contoso,DC=com /chkdeleg /fixdeleg
```

Extended Change Access Control List Tool

XCACL



Xcacls

This tool allows the file system security options accessible in Windows Explorer to be set from the command line. Xcacls does this by displaying and modifying the ACLs of files.

Xcacls is especially useful in unattended installations of Windows. With this tool, the initial access rights can be set for folders in which the operating system resides. When distributing software to servers or workstations, Xcacls also offers one-step protection against deletion of directories or files by users.

Xcacls Examples

```
XCACLS *.* /G administrator:RW /Y
```

This command replaces the ACL of all files and directories found in the current directory, without scanning any subdirectories and without confirmation.

```
XCACLS *.* /G TestUser:RWED;RW /E
```

This command edits the ACL of a file or a directory, but its effect on a directory is different. The ACE added to the directory is also an inherit ACE for new files created in this directory.

In this example, the command gives TestUser read, write, run, and delete rights on all new files created in this directory, but only read and write permissions on the directory itself.

```
XCACLS *.* /G TestUser:R;TRW /E
```

This command grants read and write permissions on a directory without creating an inherit entry for new files. Therefore, in this example, new files created in this directory get no ACE for TestUser. For existing files, an ACE with read permissions is created.

File required

Xcacls.exe

Source

Support Tools

Usage

XCACLS filename [/T] [/E|/X] [/C] [/G user:perm;spec] [/R user [...]]
[/P user:perm;spec [...]] [/D user [...]] [/Y]

Parameter List:

filename Displays ACLs.

/T Changes ACLs of specified files in the current directory and all subdirectories.

/E Edits ACL instead of replacing it.

/X Same as /E except it only affects the ACEs that the specified users already own.

/C Continues on access denied errors.

/G user:perm;spec

Grants specified user access rights.

Perm can be:

R Read
C Change (write)
F Full control
P Change Permissions (Special access)
O Take Ownership (Special access)
X EXecute (Special access)
E REad (Special access)
W Write (Special access)
D Delete (Special access)

Spec can be the same as perm and will only be applied to a directory. In this case, Perm will be used for file inheritance in this directory. By default, Spec=Perm.

Special values for Spec only:

T Valid for only for directories. At least one right has to Follow. Entries between ';' and T will be ignored.

/R user Revokes specified user's access rights.

/P user:perm;spec

Replaces specified user's access rights. Access right specification is same as /G option.

/D user Denies specified user access.

/Y Replaces user's rights without verify.

NOTES:

Wildcards can be used to specify more than one file.

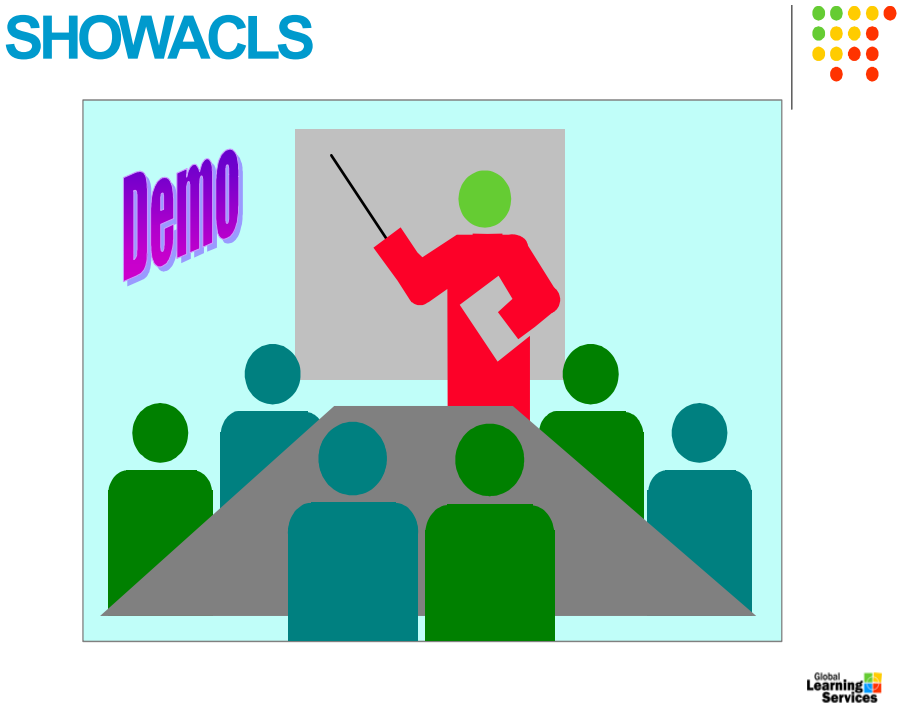
More than one user can be specified.

Access rights can be combined.

Examples:

```
XCACLS /?
XCACLS TEMP.DOC /G ADMINISTRATOR:RC
XCACLS *.TXT /G ADMINISTRATOR:RC /Y
XCACLS *.* /R ADMINISTRATOR /Y
XCACLS TEST.DLL /D ADMINISTRATOR /Y
XCACLS TEST.DLL /P ADMINISTRATOR:F /Y
XCACLS *.* /G ADMINISTRATOR:F;TRW /Y
XCACLS *.* /G ADMINISTRATOR:F;TXE /C /Y
```

SHOWACLS



Showacls

This command-line tool enumerates access rights for files, folders, and trees. It allows masking to enumerate only specific ACLs. Showacls works on NTFS partitions only.

The most useful feature of Showacls is the ability to show permissions for a particular user. The method that Showacls uses to perform this is by enumerating the local and global groups that the particular user belongs to, and matching the user's SID and the SIDs of the groups the users belongs to, to the SIDs in each ACE entry.

The latest version has adopted the "standard" permissions, Full, Change and Read-Only where appropriate. If a mask does not match these predefined values, a raw dump of the mask is performed.

Example

```
Showacls C:\Program Files
    BUILTIN\Users              Special Access [RX]
    BUILTIN\Power Users        Special Access [RWXD]
    BUILTIN\Administrators     Special Access [A]
    NT AUTHORITY\SYSTEM        Special Access [A]
    CREATOR OWNER              Special Access [A]
    NT AUTHORITY\TERMINAL SERVER USER Special Access [RWXD]
```

File required

Showacls.exe
Showacls.doc

Source

Resource Kit

Usage

Showacls /s /u:domain\user filespec

/s include sub-directories
/u specify domain\user

ACE header values:

0x1 – Object Inherit ACE
0x2 – Container Inherit ACE
0x4 – No Propagate Inherit ACE
0x8 – Inherit Only ACE

Access mask values:

A – Generic All
R – Generic Read
W – Generic Write
X – Generic Execute
w – File Write
fx – File Executive
rE – Read EA
l – List Directory
d – Read Data
S – Synchronize
r – File Read
a – File Append
D – Delete
rW – Write EA

SUBINACL

Subinacl.exe

With this command-line tool, administrators can obtain security information on files, registry keys, and services, and transfer this information from user to user, from local or global group to group, and from domain to domain.

For example, if a user has moved from one domain (DOMA) to another (DOMB), the administrator can replace DOMA\User with DOMB\User in the security information for the user's files. This gives the user access to the same files from the new domain.

Subinacl enables administrators to:

- Display security information associated with files, registry keys, or services, including owner, group, permissions ACL, DACL, and SACL.
- Change the owner of an object.
- Replace the security information for one identifier (account, group, well-known SID) with that of another identifier.
  The /replace and /changedomain options change security information in the Owner, System ACL, and Discretionary ACL fields, but the Primary Group information is never replaced. For example,

  ```
  /replace=MS\ChairMan=NEWMS\NewChairMan
  ```

  Replaces all ACEs and owners containing MS\ChairMan with the NewChairMan SID retrieved from NEWMS domain.
- Migrate security information on objects.
  This is useful if you have reorganized a network's domains and need to migrate the security information on files from one domain to another. For example,

  ```
  /changedomain=OldDomainName=NewDomainName
  ```

  replaces all ACEs with a SID from OldDomainName with the equivalent SID found in NewDomainName.

This tool is designed for use by administrators. Some actions may fail or generate error messages if the user does not have the following privileges:

- SeBackupPrivilege (Back up files and directories.)
- SeChangeNotifyPrivilege (Bypass traverse checking.)
- SeRestorePrivilege (Restore files and directories.)
- SeSecurityPrivilege (Manage auditing and security log.)
- SeTakeOwnershipPrivilege (Take ownership of files or other objects.)
- SeTcbPrivilege (Act as part of the operating system.)

File Required

Subinacl.exe

Source

Resource Kit

Usage

SubInAcl [/option...] /object_type object_name [[/action[=parameter]...]

/options:

/outputlog=FileName	/errorlog=FileName
/noverbose	/verbose (default)
/notestmode (default=/notestmode)	/testmode
/alternatesamserver=SamServer	/offlinesam=FileName
/stringreplaceonoutput=string1=string2	
/expandenvironmentsymbols (default)	/noexpandenvironmentsymbols
/statistic (default)	/nostatistic
/dumpcachedsids=FileName	/separator=character

/object_type:

/service	/keyreg	/subkeyreg
/file	/subdirectories[=directoriesonly|filesonly]	
/clustershare	/kernelobject	/metabase
/printer	/onlyfile	/process
/share	/samobject	

/action :

/display[=dacl|sacl|owner|primarygroup|sdsize|sddl] (default)
/setowner=owner
/replace=[DomainName\]OldAccount=[DomainName\]New_Account
/accountmigration=[DomainName\]OldAccount=[DomainName\]New_Account
/changedomain=OldDomainName=NewDomainName[=MappingFile[=Both]]
/migratetodomain=SourceDomain=DestDomain=[MappingFile[=Both]]

/findsid=[DomainName\]Account[=stop|continue]
/suppresssid=[DomainName\]Account
/confirm
/ifchangecontinue
/cleandeletedsidsfrom=DomainName[=dacl|sacl|owner|primarygroup|all]
/testmode
/accesscheck=[DomainName\]Username
/setprimarygroup=[DomainName\]Group
/grant=[DomainName\]Username[=Access]
/deny=[DomainName\]Username[=Access]
/sgrant=[DomainName\]Username[=Access]
/sdeny=[DomainName\]Username[=Access]
/revoke=[DomainName\]Username
/perm
/audit
/compactsecuritydescriptor
/pathexclude=pattern
/objectexclude=pattern
/sddl=sddl_string

For more information, run

Subinacl /help

at the command prompt.