# Windows Access Control List (ACL) 6

What do we have in this session?

1. Windows Privileges
2. Running with Special Privileges
3. Running with Administrator Privileges
4. Asking the User for Credentials
5. Acquiring user credentials
6. Changing Privileges in a Token
7. Enabling and Disabling Privileges
8. Authorization Constants
9. Account Rights Constants
10. Privilege Constants

The expected abilities that supposed to be acquired in this session are:

1. Able to understand the Windows Privileges (instead of Access Rights).
2. Able to understand and manipulate Windows Privileges in the Access Token.

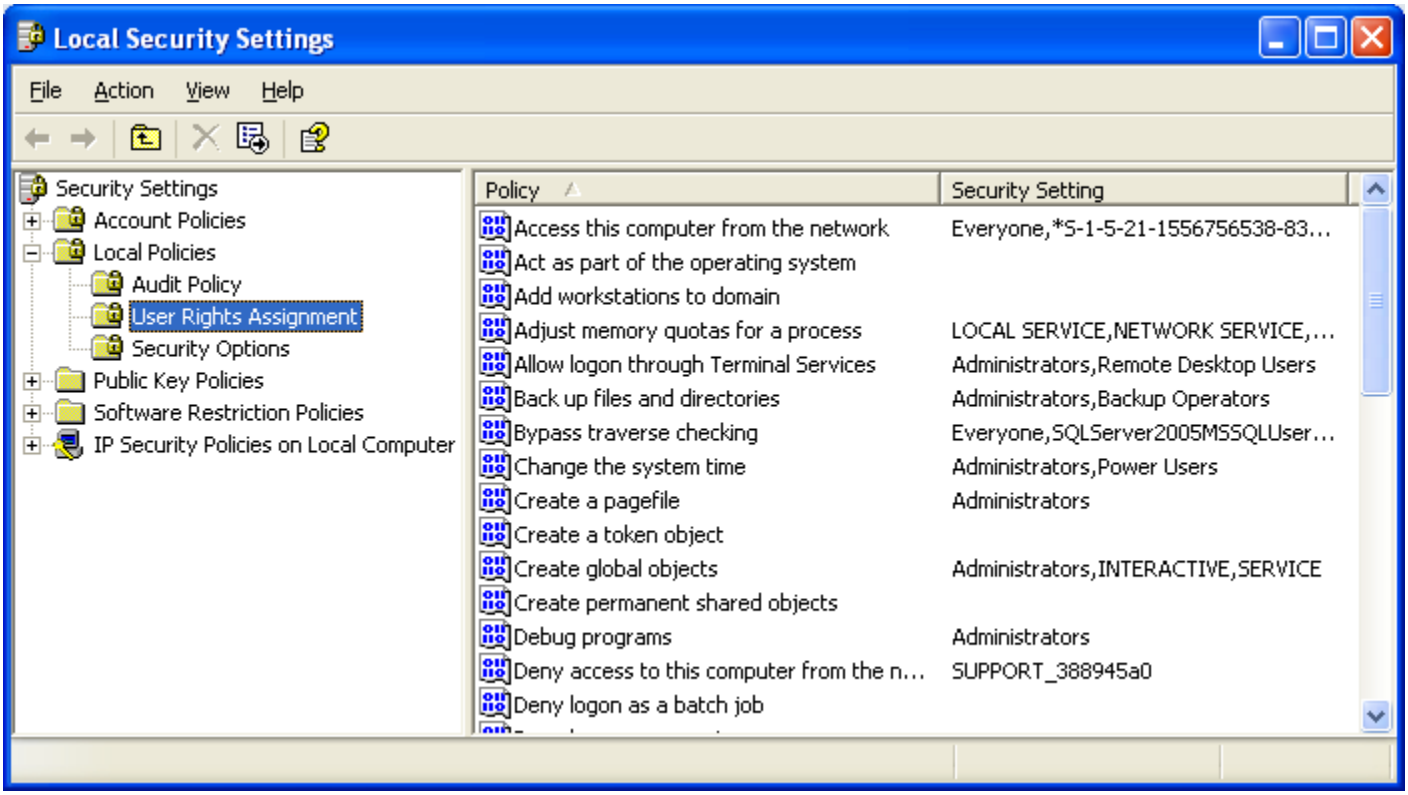
**Windows Privileges**

A privilege **is the right of an account, such as a user or group account, to perform various system-related operations on the local computer, such as shutting down the system, loading device drivers, or changing the system time**. Privileges differ from **access rights** in two ways:

1. Privileges control access to system resources and system-related tasks, whereas access rights control access to securable objects.
2. A system administrator assigns privileges to user and group accounts, whereas the system grants or denies access to a securable object based on the access rights granted in the ACEs of the object's DACL.

Each system has an account database that stores the privileges held by user and group accounts. When a user logs on, the system produces an access token that contains a list of the user's privileges, including those granted to the user or to groups to which the user belongs. Note that the privileges apply only to the local computer; a domain account can have different privileges on different computers. For example, the user Rights assignments on the local computer can be seen through the Local Security Settings (Administrative Tools > Local Security Policy) snap-in.

Keep in mind that for Windows server there are another two security policies: Domain Security Policy and DC Security Policy. We can just enable and disable the Windows Rights.




When the user tries to perform a privileged operation, the system checks the user's access token to determine whether the user holds the necessary privileges, and if so, it checks whether the privileges are enabled. If the user fails these tests, the system does not perform the operation. To determine the privileges held in an access token, call the GetTokenInformation() function, which also indicates which privileges are enabled. Most privileges are disabled by default. The Windows API defines a set of string constants, such as SE_ASSIGNPRIMARYTOKEN_NAME, to identify the various privileges. These constants are the same on all systems and are defined in winnt.h. However, the functions those get and adjust the privileges in an access token use the LUID type to identify privileges. The LUID values for a privilege can differ from one computer to another and from one boot to another on the same computer.

To get the current LUID that corresponds to one of the string constants, use the LookupPrivilegeValue() function. Use the LookupPrivilegeName() function to convert a LUID to its corresponding string constant. The system provides a set of display names that describe each of the privileges. These are useful when you need to display a description of a privilege to the user. You can use the LookupPrivilegeDisplayName() function to retrieve a description string that corresponds to the string constant for a privilege. For example, on systems that use U.S. English, the display name for the SE_SYSTEMTIME_NAME privilege is "Change the

system time". You can use the PrivilegeCheck() function to determine whether an access token holds a specified set of privileges. This is useful primarily to server applications that are impersonating a client.
A system administrator can use administrative tools, such as User Manager, to add or remove privileges for user and group accounts. Administrators can programmatically use the LSA functions to work with privileges. The LsaAddAccountRights() and LsaRemoveAccountRights() functions add or remove privileges from an account. The LsaEnumerateAccountRights() function enumerates the privileges held by a specified account. The LsaEnumerateAccountsWithUserRight() function enumerates the accounts that hold a specified privilege. The information for LUID structure is given in the following Table. Do not manipulate LUID directly. Applications should use functions and structures to manipulate LUID values.

**Running with Special Privileges**

Some functions require special privileges to run correctly. In some cases, the function can only be run by certain users or by members of certain groups. **The most common requirement is that the user be a local administrator**. Other functions require the user's account to have specific privileges enabled. To reduce the possibility of unauthorized code being able to get control, the system should run with the least privilege necessary. Applications that need to call functions that require special privileges can leave the system open to attack by hackers. Such applications should be designed to run for short periods of time and should inform the user of the security implications involved.

**Running with Administrator Privileges**

The first step in establishing which type of account your application needs to run under is to examine what resources the application will use and what privileged APIs it will call. You may find that the applications, or large parts of it, do not require administrator privileges. You can provide the privileges your application needs with **less exposure to malicious attack** by using one of the following approaches:

1. **Run under an account with less privilege**. One way to do this is to use PrivilegeCheck() to determine what privileges are enabled in a token. If the available privileges are not adequate for the current operation, you can disable that code and ask the user to logon to an account with administrator privileges.
2. **Break into a separate application functions that require administrator permissions**. You can provide for the user a shortcut that executes the RunAs command. Programmatically, you can configure the RunAs command under the AppId registry key for your application.

3. **Authenticate the user by calling CredUIPromptForCredentials() (GUI) or CredUICmdLinePromptForCredentials() (command line) to obtain user name and password**.
4. **Impersonate the user**. A process that starts under a highly privileged account like System can impersonate a user account by calling ImpersonateLoggedOnUser() or similar Impersonate functions, thus reducing privilege level. However, if a call to RevertToSelf() is injected into the thread, the process returns to the original System privileges.

If you have determined that your application must run under an account with administrator privileges and that an administrator password must be stored in the software system.

**Asking the User for Credentials**

Your application may need to prompt the user for user name and password information to avoid storing an administrator password or to verify that the token holds the appropriate privileges. However, simply prompting for credentials may train users to supply those to any random, unidentified dialog box that appears on the screen. The following procedure is recommended to reduce that training effect.

**Acquiring user credentials**

The recommended steps to properly acquire user credentials:

1. Inform the user, by using a message that is clearly part of your application, that they will see a dialog box that requests their user name and password. You can also use the CREDUI_INFO structure on the call to CredUIPromptForCredentials() to convey identifying data or a message.
2. Call CredUIPromptForCredentials(). Note that the maximum number of characters specified for user name and password information includes the terminating null character.
3. Call CredUIParseUserName() and CredUIConfirmCredentials() to verify that you obtained appropriate credentials.

The following code snippet shows how to call CredUIPromptForCredentials() to ask the user for a user name and password. It begins by filling in a CREDUI_INFO structure with information about what prompts to use. Next, the code fills two buffers with zeros. This is done to ensure that no information gets passed to the function that might reveal an old user name or password to the user. The call to CredUIPromptForCredentials() brings up the dialog box. For security reasons, this example uses the CREDUI_FLAGS_DO_NOT_PERSIST flag to prevent the operating system from storing the password because it might then be exposed. If there are no

errors, CredUIPromptForCredentials() fills in the pszName and pszPwd variables and returns zero. When the application has finished using the credentials, it should put zeros in the buffers to prevent the information from being accidentally revealed.

```
#include <windows.h>
#include <wincred.h>

CREDUI_INFO cui;
WCHAR pszName[CREDUI_MAX_USERNAME_LENGTH+1];
WCHAR pszPwd[CREDUI_MAX_PASSWORD_LENGTH+1];
BOOL fSave;
DWORD dwErr;

cui.cbSize = sizeof(CREDUI_INFO);
cui.hwndParent = NULL;
//  Ensure that MessageText and CaptionText identify what credentials to use
and which application requires them.
cui.pszMessageText = L"Enter administrator account information";
cui.pszCaptionText = L"CredUITest";
cui.hbmBanner = NULL;
fSave = FALSE;
SecureZeroMemory(pszName, sizeof(pszName));
SecureZeroMemory(pszPwd, sizeof(pszPwd));
dwErr = CredUIPromptForCredentials(
                  &cui,                           // CREDUI_INFO structure
                  L"TheServer",                           // Target for
credentials, usually a server
                  NULL,                           // Reserved
                  0,                              // Reason
                  pszName,                        // User name
                  CREDUI_MAX_USERNAME_LENGTH+1,// Max number of char for user
name
                  pszPwd,                         // Password
                  CREDUI_MAX_PASSWORD_LENGTH+1,// Max number of char for
password
                  &fSave,                         // State of save check box
                  CREDUI_FLAGS_GENERIC_CREDENTIALS |  // flags
                  CREDUI_FLAGS_ALWAYS_SHOW_UI |
                  CREDUI_FLAGS_DO_NOT_PERSIST);

if(!dwErr)
{
    //  TODO: Put code that uses the credentials here.
    //  When you have finished using the credentials, erase them from memory.
    SecureZeroMemory(pszName, sizeof(pszName));
    SecureZeroMemory(pszPwd, sizeof(pszPwd));
}
```

### Changing Privileges in a Token

You can change the privileges in either a primary or an impersonation token in two ways:

1. Enable or disable privileges by using the AdjustTokenPrivileges() function.

2. Restrict or remove privileges by using the CreateRestrictedToken() function.

AdjustTokenPrivileges() cannot add or remove privileges from the token. It can only **enable** existing privileges that are currently disabled or **disable** existing privileges that are currently enabled. CreateRestrictedToken() has more extensive capabilities as follows:

1. **Removing a privilege**. Note that removing a privilege is not the same as disabling one. After a privilege is removed from a token, it cannot be put back.
2. **Attaching the deny-only attribute to SIDs in the token**. This has the effect of disallowing specific groups or accounts, for example, denying the Everyone group delete access to a particular file.
3. **Specifying a list of restricting SIDs in the token**.

**Enabling and Disabling Privileges**

Enabling a privilege in an access token allows the process to perform system-level actions that it could not previously. Your application should thoroughly verify that the privilege is appropriate to the type of account, especially for the following powerful privileges:

| Privilege constant/string | Display name |
|---|---|
| SE_ASSIGNPRIMARYTOKEN_NAME<br>SeAssignPrimaryTokenPrivilege | Replace a process level token. |
| SE_BACKUP_NAME<br>SeBackupPrivilege | Backup files and directories. |
| SE_DEBUG_NAME<br>SeDebugPrivilege | Debug programs. |
| SE_INCREASE_QUOTA_NAME<br>SeIncreaseQuotaPrivilege | Adjust memory quotas for a process. |
| SE_TCB_NAME<br>SeTchPrivilege | Act as part of the operating system. |

Table 13

Before enabling any of these potentially dangerous privileges, determine that functions or operations in your code actually require the privileges. For example, very few functions in the operating system actually require the SeTchPrivilege.

**Authorization Constants**

Authorization constants are categorized according to usage as follows:

1. Account Rights Constants.
2. Privilege Constants.
3. Account Rights Constants

Account rights, like privileges, determine the operations that a user account can perform. An administrator assigns account rights to user and group accounts. Each user's account rights include those granted to the user or to groups to which the user belongs. A system administrator can use the Local Security Authority (LSA) functions to work with account rights. The LsaAddAccountRights() and LsaRemoveAccountRights() functions add or remove account rights from an account. The LsaEnumerateAccountRights() function enumerates the account rights held by a specified account. The LsaEnumerateAccountsWithUserRight() function enumerates the accounts that hold a specified account right. All of the LSA functions mentioned above support both account rights and Privilege Constants. Unlike privileges, however, account rights are not supported by the LookupPrivilegeValue() and LookupPrivilegeName() functions. The GetTokenInformation() function will obtain information on account rights if TokenGroups, and not TokenPrivileges, is specified as the value of the TokenInformationClass parameter. The following account right constants are used to control the logon ability of an account. The LogonUser() or LsaLogonUser() functions fail if the account being logged on does not have the account rights required for the type of logon being performed. The SE_DENY rights override the corresponding account rights. An administrator can assign an SE_DENY right to an account to override any logon rights that an account might have as a result of a group membership. For example, you could assign the SE_NETWORK_LOGON_NAME right to Everyone but assign the SE_DENY_NETWORK_LOGON_NAME right to Administrators to prevent remote administration of computers.

| Account right constant | Description |
|---|---|
| SE_BATCH_LOGON_NAME | Required for an account to log on using the batch logon type. |
| SE_INTERACTIVE_LOGON_NAME | Required for an account to log on using the interactive logon type. |
| SE_NETWORK_LOGON_NAME | Required for an account to log on using the network logon type. |
| SE_SERVICE_LOGON_NAME | Required for an account to log on using the service logon type. |
| SE_DENY_BATCH_LOGON_NAME | Explicitly denies an account the right to log on using the batch logon type. |
| SE_DENY_INTERACTIVE_LOGON_NAME | Explicitly denies an account the right to log on using the interactive logon type. |

| | |
|---|---|
| SE_DENY_NETWORK_LOGON_NAME | Explicitly denies an account the right to log on using the network logon type. |
| SE_DENY_SERVICE_LOGON_NAME | Explicitly denies an account the right to log on using the service logon type. |

Table 14

The preceding account right constants are defined as strings in ntsecapi.h. For example, the SE_INTERACTIVE_LOGON_NAME constant is defined as "SeInteractiveLogonRight".

**Privilege Constants**

The following privilege constants are defined as strings in winnt.h. For example, the SE_AUDIT_NAME constant is defined as "SeAuditPrivilege". The functions that get and adjust the privileges in an access token use the LUID type to identify privileges. Use the LookupPrivilegeValue() function to determine the LUID on the local system that corresponds to a privilege constant. You can use the LookupPrivilegeName() function to convert a LUID to its corresponding string constant. The operating system represents a privilege by using the string that follows "User Right" in the Description column of the following table.

| Privilege constant | Description |
|---|---|
| SE_ASSIGNPRIMARYTOKEN_NAME<br>SeAssignPrimaryTokenPrivilege | Required to assign the primary token of a process. User Right: Replace a process-level token. |
| SE_AUDIT_NAME<br>SeAuditPrivilege | Required to generate audit-log entries. Give this privilege to secure servers. User Right: Generate security audits. |
| SE_BACKUP_NAME<br>SeBackupPrivilege | Required to perform backup operations. This privilege causes the system to grant all read access control to any file, regardless of the ACL specified for the file. Any access request other than read is still evaluated with the ACL. This privilege is required by the RegSaveKey() and RegSaveKeyEx() functions. The following access rights are granted if this privilege is held:<br>1. READ_CONTROL<br>2. ACCESS_SYSTEM_SECURITY<br>3. FILE_GENERIC_READ<br>4. FILE_TRAVERSE |

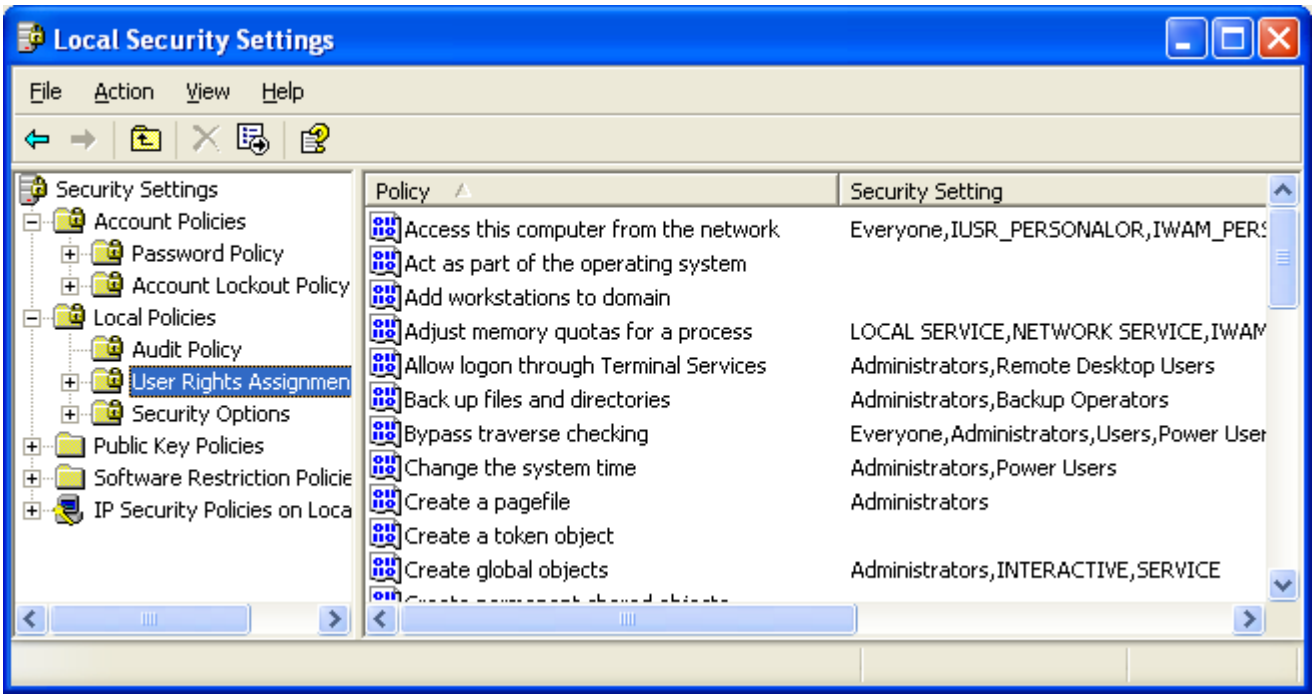
| | |
|---|---|
| | User Right: Back up files and directories. |
| SE_CHANGE_NOTIFY_NAME<br>SeChangeNotifyPrivilege | Required to receive notifications of changes to files or directories. This privilege also causes the system to skip all traversal access checks. It is enabled by default for all users. User Right: Bypass traverse checking. |
| SE_CREATE_GLOBAL_NAME<br>SeCreateGlobalPrivilege | Required to create named file mapping objects in the global namespace during Terminal Services sessions. This privilege is enabled by default for administrators, services, and the local system account. User Right: Create global objects. Windows XP, Windows 2000 SP3 and earlier, and Windows NT: This privilege is not supported. |
| SE_CREATE_PAGEFILE_NAME<br>SeCreatePagefilePrivilege | Required to create a paging file. User Right: Create a pagefile. |
| SE_CREATE_PERMANENT_NAME<br>SeCreatePermanentPrivilege | Required to create a permanent object. User Right: Create permanent shared objects. |
| SE_CREATE_TOKEN_NAME<br>SeCreateTokenPrivilege | Required to create a primary token. User Right: Create a token object. |
| SE_DEBUG_NAME<br>SeDebugPrivilege | Required to debug and adjust the memory of a process owned by another account. User Right: Debug programs. |
| SE_ENABLE_DELEGATION_NAME<br>SeEnableDelegationPrivilege | Required to mark user and computer accounts as trusted for delegation. User Right: Enable computer and user accounts to be trusted for delegation. |
| SE_IMPERSONATE_NAME<br>SeImpersonatePrivilege | Required to impersonate. User Right: Impersonate a client after authentication. Windows XP, Windows 2000 SP3 and earlier, and Windows NT: This privilege is not supported. |
| SE_INC_BASE_PRIORITY_NAME<br>SeIncreaseBasePriorityPrivilege | Required to increase the base priority of a process. User Right: Increase scheduling priority. |
| SE_INCREASE_QUOTA_NAME<br>SeIncreaseQuotaPrivilege | Required to increase the quota assigned to a process. User Right: Adjust memory quotas for a process. |
| SE_LOAD_DRIVER_NAME<br>SeLoadDriverPrivilege | Required to load or unload a device driver. User Right: Load and unload device drivers. |
| SE_LOCK_MEMORY_NAME<br>SeLockMemoryPrivilege | Required to lock physical pages in memory. User Right: Lock pages in memory. |
| SE_MACHINE_ACCOUNT_NAME<br>SeMachineAccountPrivilege | Required to create a computer account. User Right: Add workstations to domain. |

| | |
|---|---|
| SE_MANAGE_VOLUME_NAME<br>SeManageVolumePrivilege | Required to enable volume management privileges. User Right: Manage the files on a volume. |
| SE_PROF_SINGLE_PROCESS_NAME<br>SeProfileSingleProcessPrivilege | Required to gather profiling information for a single process. User Right: Profile single process. |
| SE_REMOTE_SHUTDOWN_NAME<br>SeRemoteShutdownPrivilege | Required to shut down a system using a network request. User Right: Force shutdown from a remote system. |
| SE_RESTORE_NAME<br>SeRestorePrivilege | Required to perform restore operations. This privilege causes the system to grant all write access control to any file, regardless of the ACL specified for the file. Any access request other than write is still evaluated with the ACL. Additionally, this privilege enables you to set any valid user or group SID as the owner of a file. This privilege is required by the RegLoadKey() function. The following access rights are granted if this privilege is held:<br>WRITE_DAC<br>WRITE_OWNER<br>ACCESS_SYSTEM_SECURITY<br>FILE_GENERIC_WRITE<br>FILE_ADD_FILE<br>FILE_ADD_SUBDIRECTORY<br>DELETE<br><br>User Right: Restore files and directories. |
| SE_SECURITY_NAME<br>SeSecurityPrivilege | Required to perform a number of security-related functions, such as controlling and viewing audit messages. This privilege identifies its holder as a security operator. User Right: Manage auditing and security log. |
| SE_SHUTDOWN_NAME<br>SeShutdownPrivilege | Required to shut down a local system. User Right: Shut down the system. |
| SE_SYNC_AGENT_NAME<br>SeSyncAgentPrivilege | Required for a domain controller to use the LDAP directory synchronization services. This privilege enables the holder to read all objects and properties in the directory, regardless of the protection on the objects and properties. By default, it is assigned to the Administrator and LocalSystem accounts on domain controllers. User Right: Synchronize directory service data. |

| | |
|---|---|
| SE_SYSTEM_ENVIRONMENT_NAME<br>SeSystemEnvironment | Required to modify the nonvolatile RAM of systems that use this type of memory to store configuration information. User Right: Modify firmware environment values. |
| SE_SYSTEM_PROFILE_NAME<br>SeSystemProfilePrivilege | Required to gather profiling information for the entire system. User Right: Profile system performance. |
| SE_SYSTEMTIME_NAME<br>SeSystemtimePrivilege | Required to modify the system time. User Right: Change the system time. |
| SE_TAKE_OWNERSHIP_NAME<br>SeTakeOwnershipPrivilege | Required to take ownership of an object without being granted discretionary access. This privilege allows the owner value to be set only to those values that the holder may legitimately assign as the owner of an object. User Right: Take ownership of files or other objects. |
| SE_TCB_NAME<br>SeTcbPrivilege | This privilege identifies its holder as part of the trusted computer base. Some trusted protected subsystems are granted this privilege. User Right: Act as part of the operating system. |
| SE_UNDOCK_NAME<br>SeUndockPrivilege | Required to undock a laptop. User Right: Remove computer from docking station. |
| SE_UNSOLICITED_INPUT_NAME<br>SeUnsolicitedInputPrivilege | Required to read unsolicited input from a terminal device. User Right: Not applicable. |

Table 15

The operating system displays the user right strings in the Policy column of the User Rights Assignment node of the Local Security Settings Microsoft Management Console (MMC) snap-in as shown below (Administrative Tools → Local Security Policy).




The following code snippet shows how to enable or disable a privilege in an access token. The example calls the LookupPrivilegeValue() function to get the LUID that the local system uses to identify the privilege. Then the example calls the AdjustTokenPrivileges() function, which either enables or disables the privilege that depends on the value of the bEnablePrivilege parameter.

```
BOOL SetPrivilege(
    HANDLE hToken,          // access token handle
    LPCTSTR lpszPrivilege,  // name of privilege to enable/disable
    BOOL bEnablePrivilege   // to enable or disable privilege
    )
{
      TOKEN_PRIVILEGES tp;
      LUID luid;
      lpszPrivilege = L"Generate security audits";

      if(!LookupPrivilegeValue(
            NULL,           // lookup privilege on local system
            lpszPrivilege, // privilege to lookup
            &luid))         // receives LUID of privilege
      {
            wprintf(L"LookupPrivilegeValue error: %u\n", GetLastError());
            return FALSE;
      }

      tp.PrivilegeCount = 1;
      tp.Privileges[0].Luid = luid;

      if(bEnablePrivilege)
            tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
```

```
	else
		tp.Privileges[0].Attributes = 0;

	// Enable the privilege or disable all privileges
	if(!AdjustTokenPrivileges(
		hToken,
		FALSE,
		&tp,
		sizeof(TOKEN_PRIVILEGES),
		(PTOKEN_PRIVILEGES) NULL,
		(PDWORD) NULL))
	{
		wprintf(L"AdjustTokenPrivileges error: %u\n", GetLastError());
		return FALSE;
	}

	if(GetLastError() == ERROR_NOT_ALL_ASSIGNED)
	{
		wprintf(L"The token does not have the specified privilege. \n");
		return FALSE;
	}
	return TRUE;
}
```

We will explore the working program examples in other sessions.