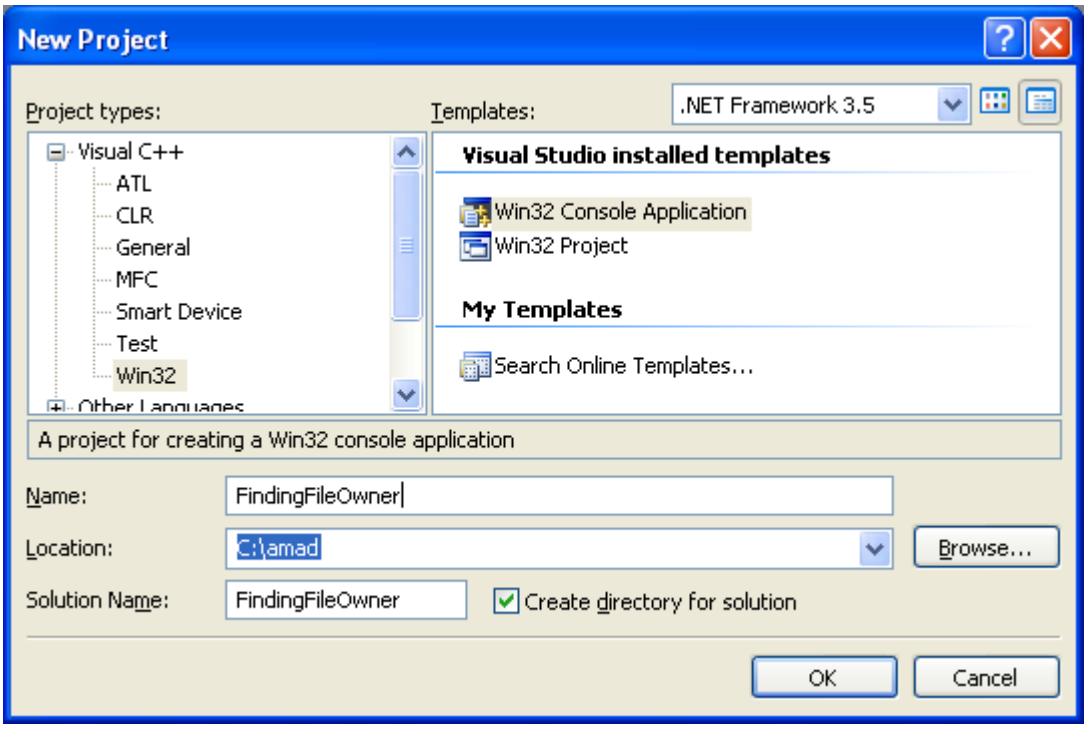
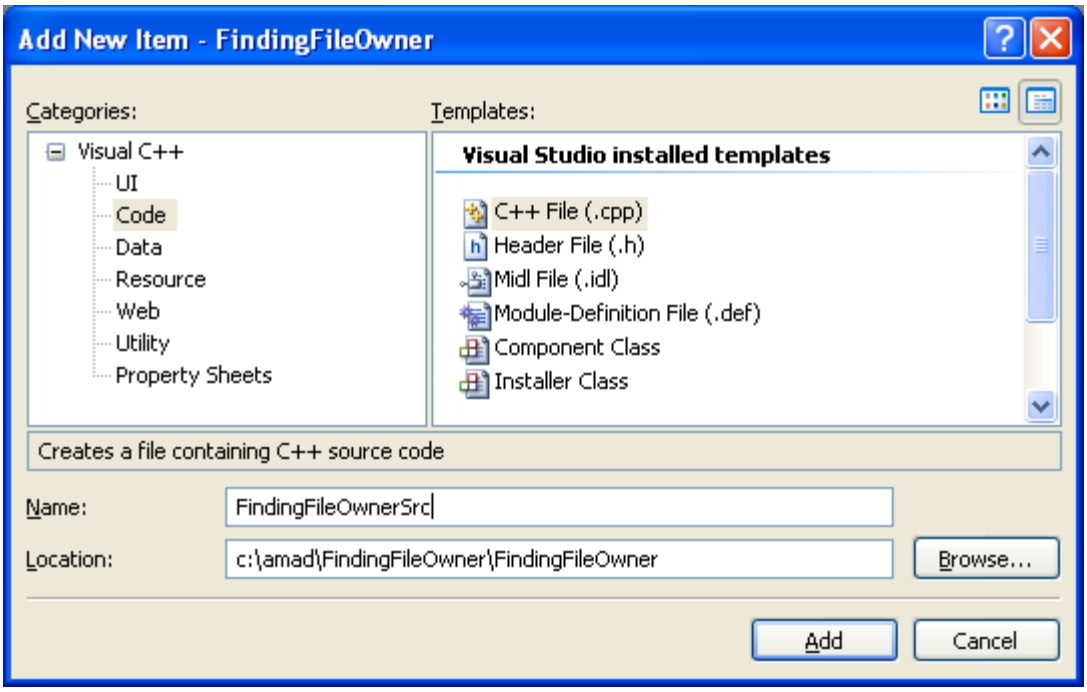
1. Finding the Owner of a File Object Program Example.
2. Taking Object Ownership Program Example.

Finding the Owner of a File Object Program Example

The following example uses the GetSecurityInfo() and LookupAccountSid() functions to find and print the name of the owner of a file. The file exists in C:\JohnnyDir directory on the local machine. The file own by Johnny, however the computer used to run this program logged as Mike spoon.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



Next, add the following source code.

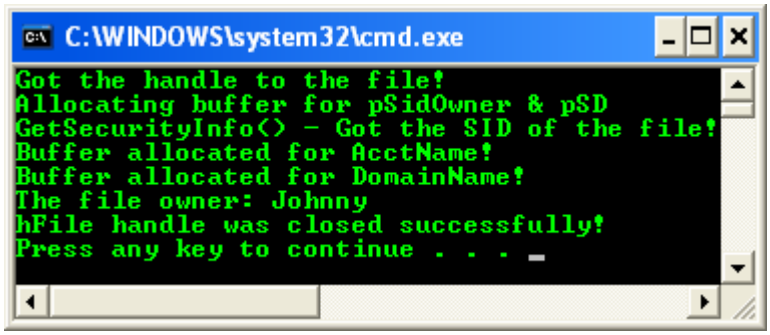
```
// Finding the Owner of a File Object
#include <windows.h>
#include <stdio.h>
#include <aclapi.h>

int wmain(int argc, WCHAR *argv[])
{
	DWORD dwRtnCode = 0;
	PSID pSidOwner;
	BOOL bRtnBool = TRUE;
	// Dummy initial value, no string...
	LPTSTR AcctName = L"", DomainName = L"";
	DWORD dwAcctName = 1, dwDomainName = 1;
	// Dummy initial value, just use the defined one but unknown
	SID_NAME_USE eUse = SidTypeUnknown;
	HANDLE hFile;
	PSECURITY_DESCRIPTOR pSD = {0};

	// Get the handle of the file object.
	hFile = CreateFile(
		L"\\\\?\\C:\\JohnnyDir\\ThisisJohnnyfile.txt",  // The file name
and the path if any
		GENERIC_READ,                 // Access right
		FILE_SHARE_READ,         // Share mode
		NULL,                         // Security attribute for inherited
or not by child
		OPEN_EXISTING,                // Open the file, fail if not exist
		FILE_ATTRIBUTE_NORMAL,  // file attribute flag, Normal
		NULL);                                // Handle to template file if
GENERIC_READ right access
```

```
	// Verify
	if(hFile == INVALID_HANDLE_VALUE)
	{
		wprintf(L"CreateFile() failed, error = %u\n", GetLastError());
		// Just exit, 0 - OK, non-zero - failed
		return -1;
	}
	else
		wprintf(L"Got the handle to the file!\n");

	// Allocate memory for the SID structure.
	wprintf(L"Allocating buffer for pSidOwner & pSD\n");
	pSidOwner = (PSID)GlobalAlloc(GMEM_FIXED, sizeof(PSID));
	// Allocate memory for the security descriptor structure.
	pSD = (PSECURITY_DESCRIPTOR)GlobalAlloc(GMEM_FIXED,
sizeof(PSECURITY_DESCRIPTOR));

	// Get the owner SID of the file. Try for other object such as
	// SE_SERVICE, SE_PRINTER, SE_REGISTRY_KEY, SE_KERNEL_OBJECT,
SE_WINDOW_OBJECT etc.
	// for 2nd parameter and 3rd parameter such as
DACL_SECURITY_INFORMATION,
	// GROUP_SECURITY_INFORMATION and SACL_SECURITY_INFORMATION
	dwRtnCode = GetSecurityInfo(
		hFile,// Handle to the file
		SE_FILE_OBJECT,// Directory or file
		OWNER_SECURITY_INFORMATION, // Owner information of the (file)
object
		&pSidOwner,// Pointer to the owner of the (file) object
		NULL,
		NULL,
		NULL,
		&pSD);// Pointer to the security descriptor of the (file) object

	// Check GetLastError for GetSecurityInfo error condition.
	if(dwRtnCode != ERROR_SUCCESS)
	{
		wprintf(L"GetSecurityInfo() failed, error %u\n", GetLastError());
		return -1;
	}
	else
		wprintf(L"GetSecurityInfo() - Got the SID of the file!\n");

	// First call to LookupAccountSid() to get the buffer size AcctName.
	bRtnBool = LookupAccountSid(
		NULL,			// Local computer
		pSidOwner,		// Pointer to the SID to lookup for
		AcctName,		// The account name of the SID (pSIDOwner)
		(LPDWORD)&dwAcctName,	// Size of the AcctName in TCHAR
		DomainName,		// Pointer to the name of the Domain where the
account name was found
		(LPDWORD)&dwDomainName, // Size of the DomainName in TCHAR
		&eUse);			// Value of the SID_NAME_USE enum type
that specify the SID type

	// Allocate memory for the AcctName.
```

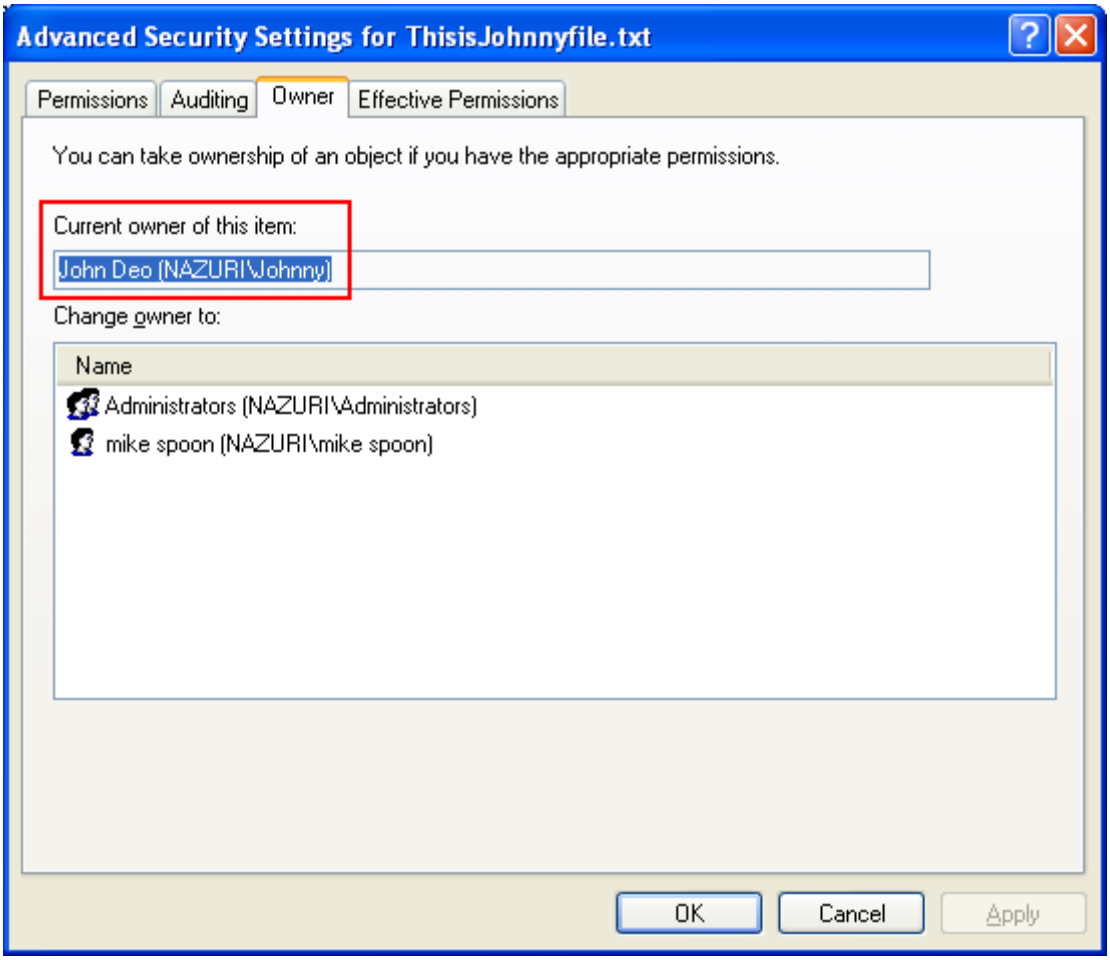
```
	AcctName = (LPTSTR)GlobalAlloc(GMEM_FIXED, dwAcctName);
	// VErify
	if(AcctName == NULL)
	{
		wprintf(L"GlobalAlloc() - Failed to allocate buffer for AcctName,
error %u\n", GetLastError());
		return -1;
	}
	else
		wprintf(L"Buffer allocated for AcctName!\n");

	DomainName = (LPTSTR)GlobalAlloc(GMEM_FIXED, dwDomainName);

	// Check GetLastError() for GlobalAlloc() error condition.
	if(DomainName == NULL)
	{
		wprintf(L"GlobalAlloc() failed to allocate buffer for DomainName,
error %u\n", GetLastError());
		return -1;
	}
	else
		wprintf(L"Buffer allocated for DomainName!\n");

	// Second call to LookupAccountSid() to get the account name.
	bRtnBool = LookupAccountSid(
		NULL,		// name of local or remote computer
		pSidOwner,	// security identifier, SID
		AcctName,	// account name buffer
		(LPDWORD)&dwAcctName,		// size of account name buffer
		DomainName, // domain name
		(LPDWORD)&dwDomainName,		// size of domain name buffer
		&eUse);		// SID type

	// Verify
	if(bRtnBool == FALSE)
	{
		DWORD dwErrorCode = GetLastError();

		if(dwErrorCode == ERROR_NONE_MAPPED)
			wprintf(L"Account owner not found for specified SID.\n");
		else
		{
			wprintf(L"LookupAccountSid() failed, error %u\n",
GetLastError());
			return -1;
		}
	}
	else if (bRtnBool == TRUE)
		// Print the account name.
		wprintf(L"The file owner: %s\n", AcctName);

	if(CloseHandle(hFile) != 0)
		wprintf(L"hFile handle was closed successfully!\n");
	else
		wprintf(L"Failed to close hFile handle, error %u\n",
GetLastError());
```

```
        return 0;
}
```

Build and run the project. The following screenshot is a sample output.

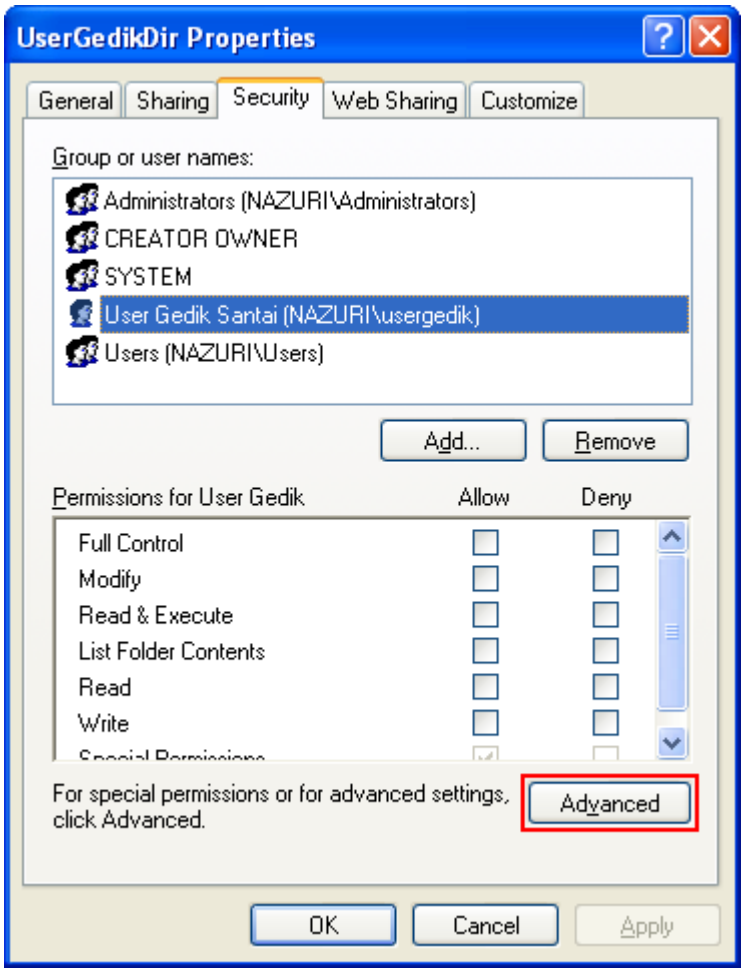


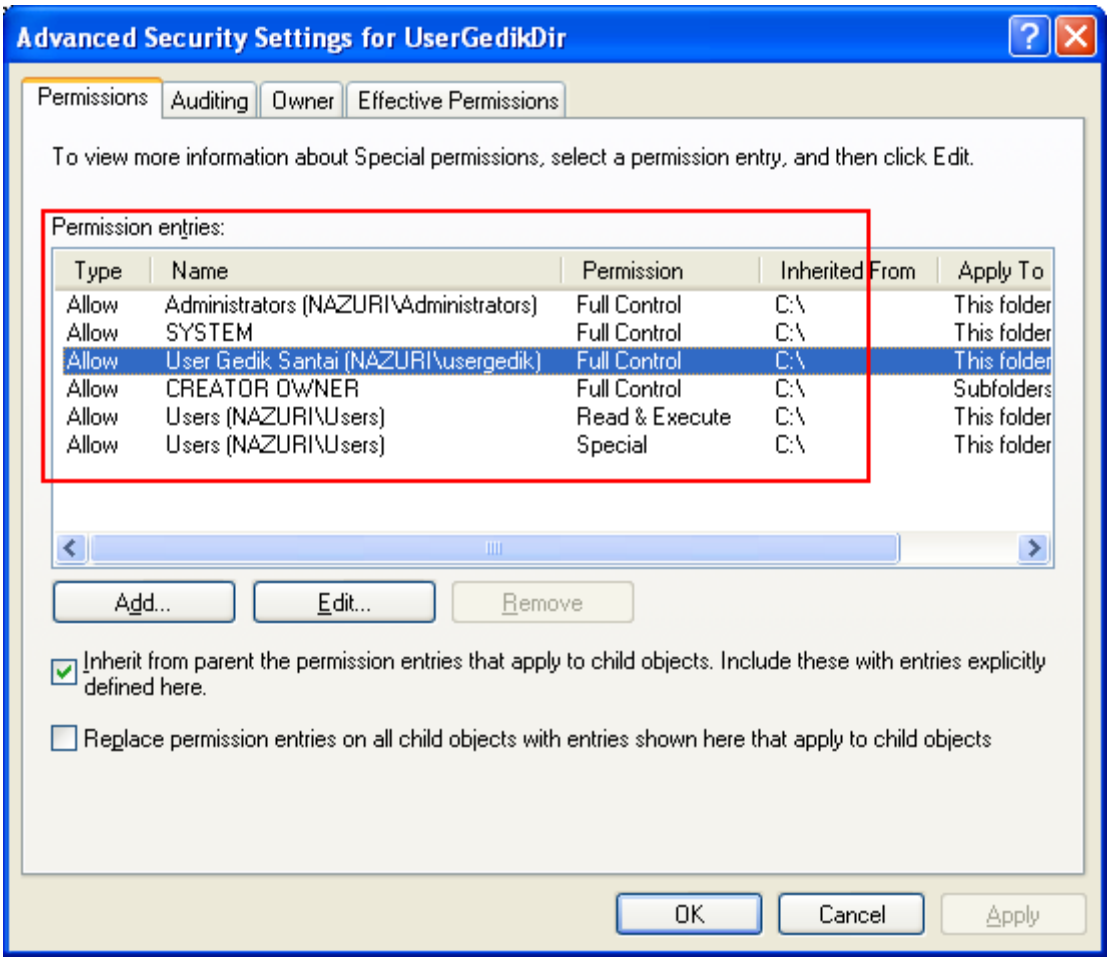
Then, let verify through the ThisisJohnnyfile.txt file property page and it confirmed that the owner is a local user Johnny.



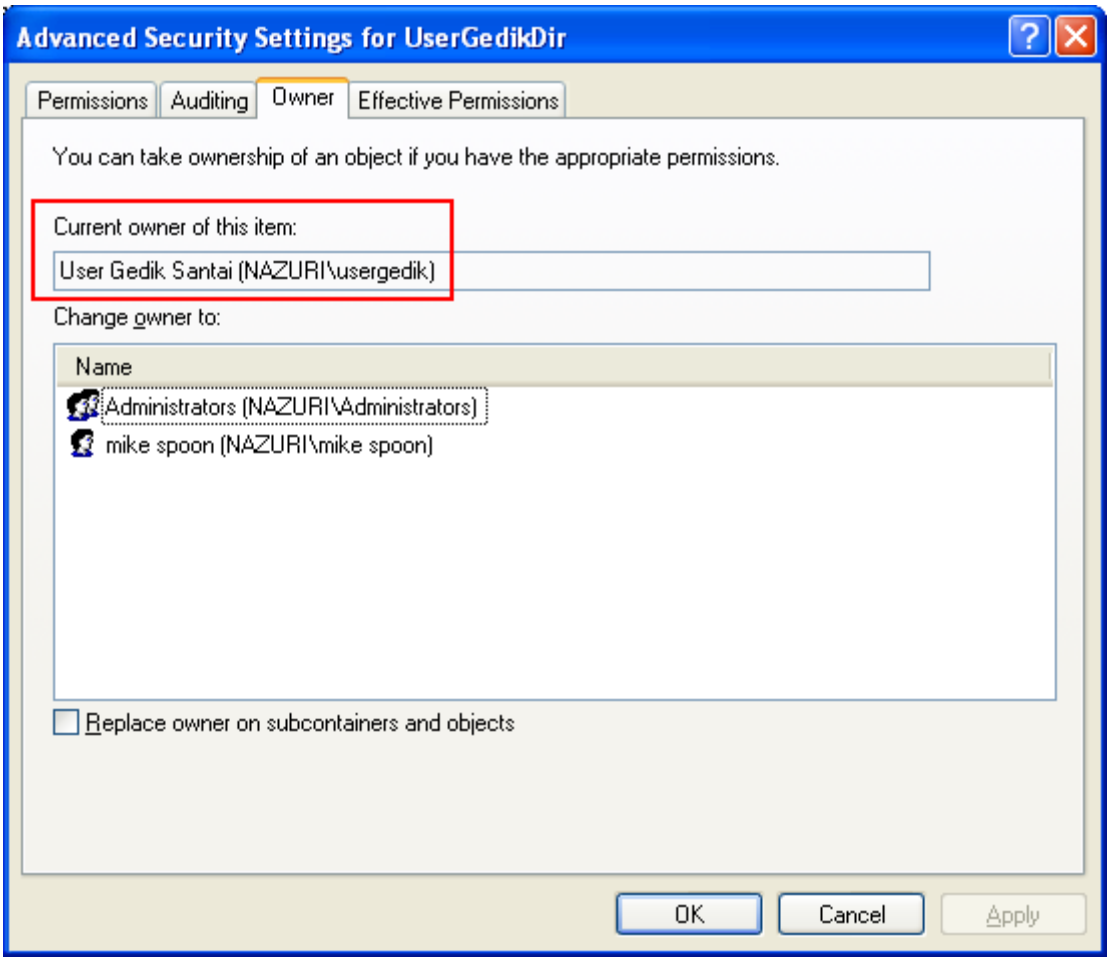
Taking the Object Ownership Program Example

The following example tries to change the DACL of a file object by taking ownership of that object. This will succeed only if the caller has WRITE_DAC access to the object or is the owner of the object. If the initial attempt to change the DACL fails, an administrator can take ownership of the object. To give the administrator ownership, the example enables the SE_TAKE_OWNERSHIP_NAME privilege in the caller's access token, and makes the local system's Administrators group the new owner of the object. If the caller is a member of the Administrators group, the code will then be able to change the object's DACL. Firstly we login as usergedik, a normal user. Then we create a folder named UserGedikDir and verify the folder through the UserGedikDir property page as shown below.

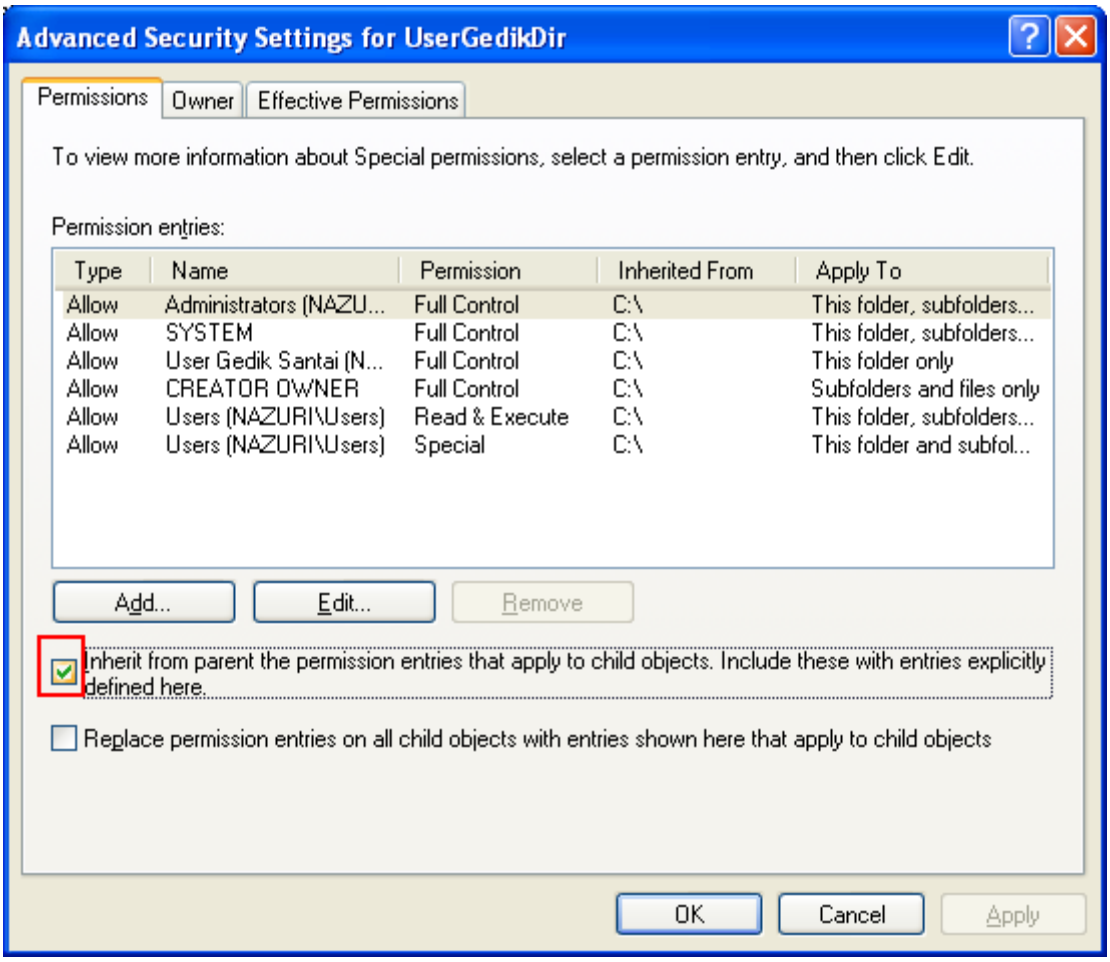




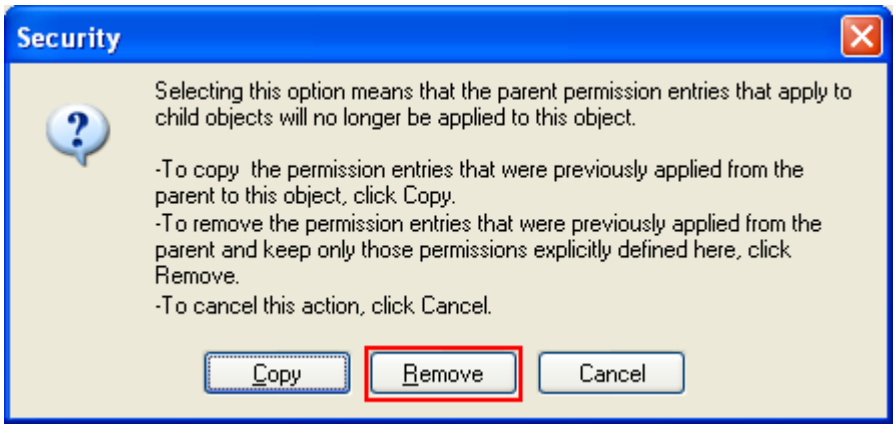
The previous figure shows the property page for the UserGedikDir directory. It is confirmed that it is created by normal user, usergedik.



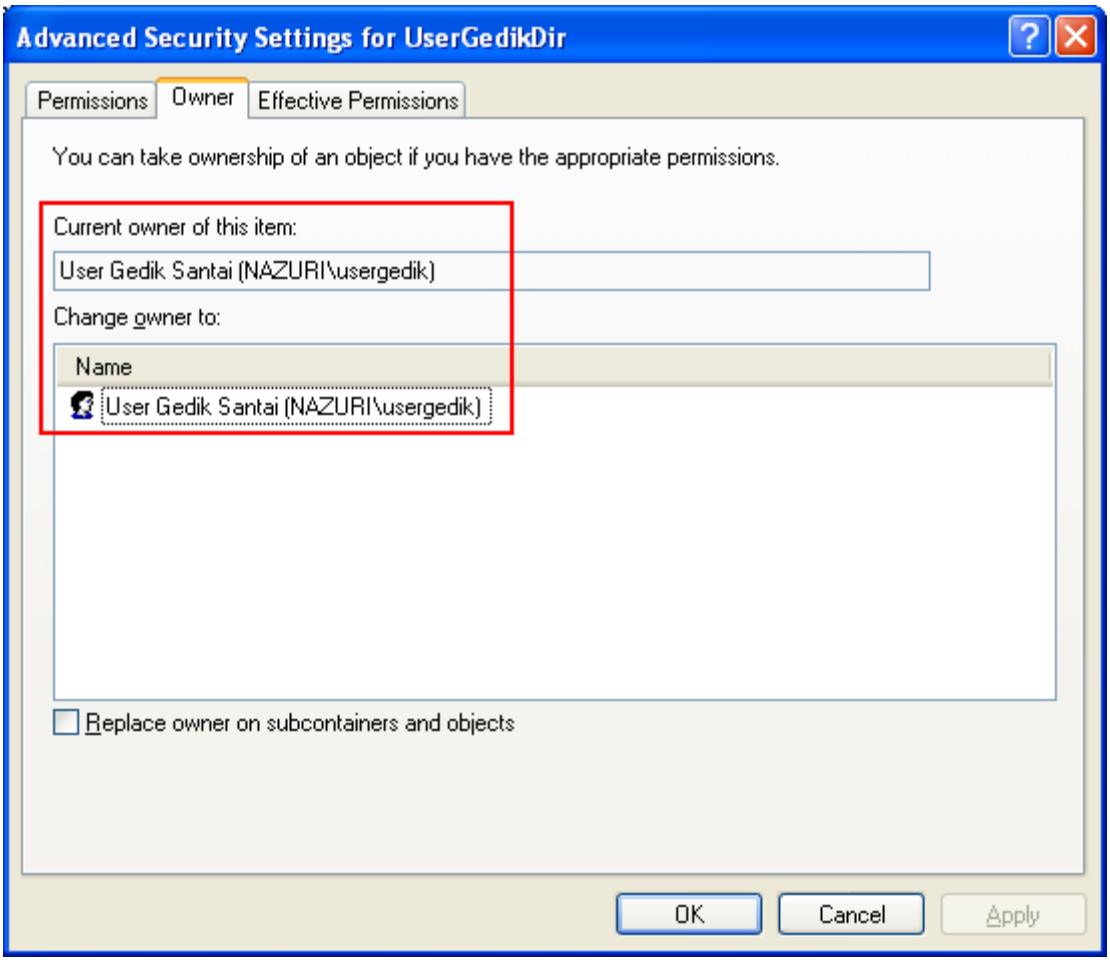
The Advanced property page for the UserGedikDir directory shows that the file was created by normal user, usergedik which of course the owner. To make thing 'worse' we remove all the permissions except usergedik.
Un-tick the "Inherit from parent the permission entries that apply to the child object. Include these with entries explicitly defined here" tick box.



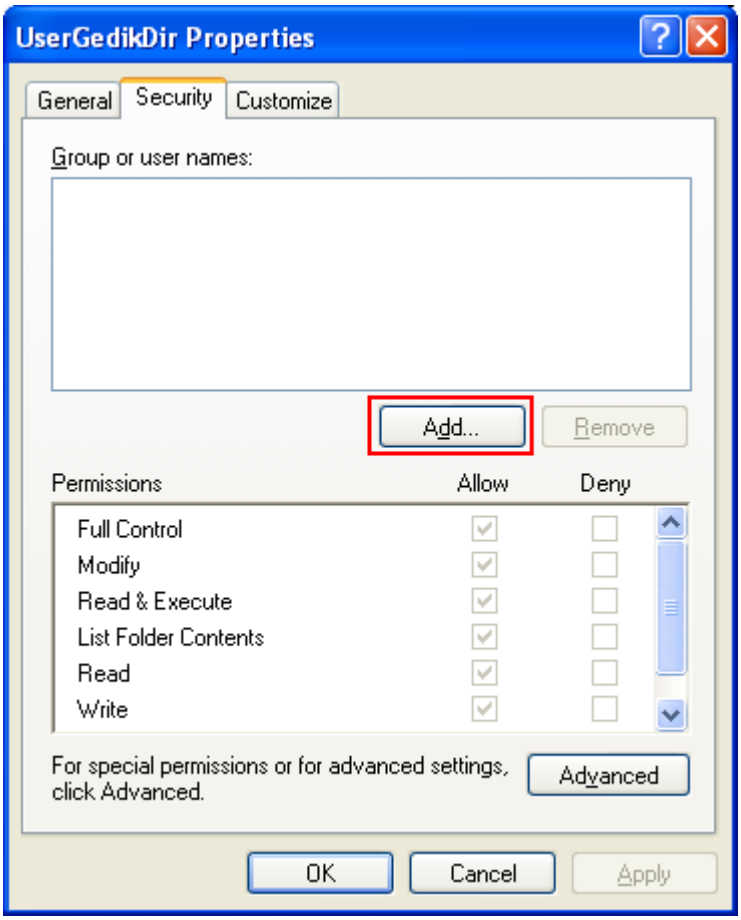
Click Remove button for the following Security message box.

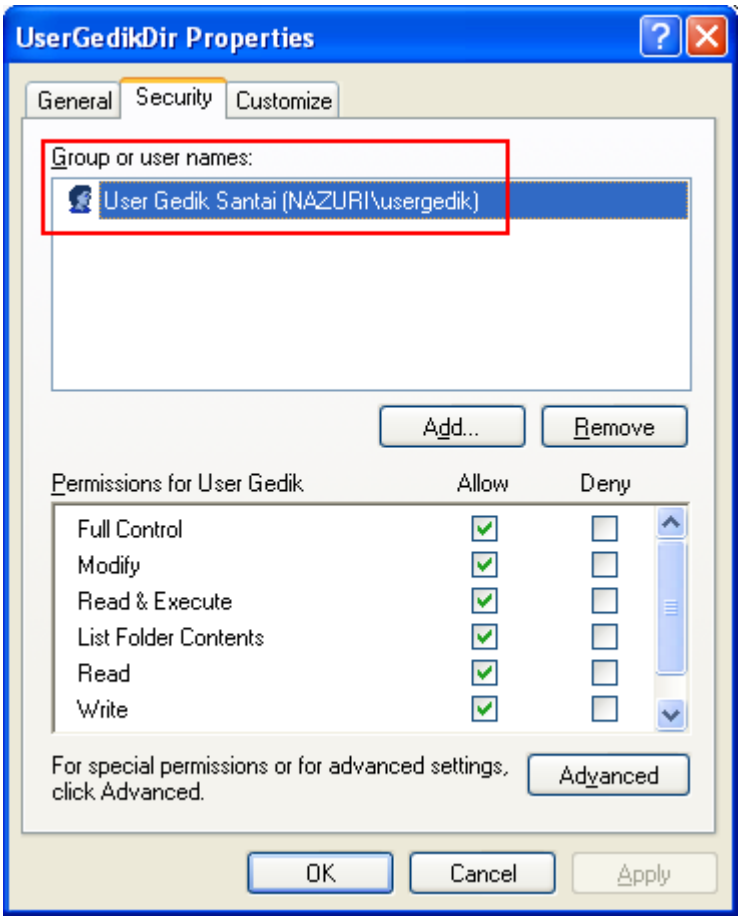


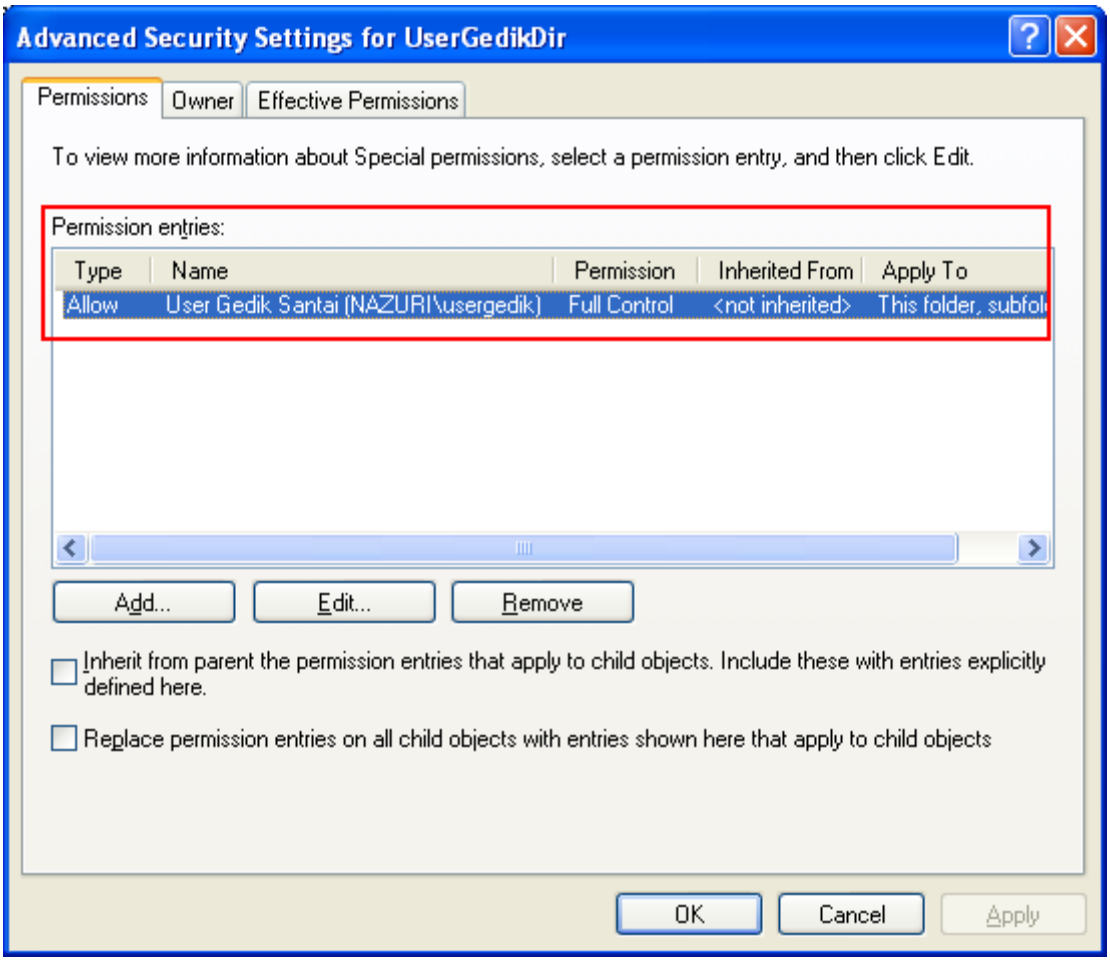
Then, only user usergedik has the full control of the folder.



Then, add the usergedik user for the permission.

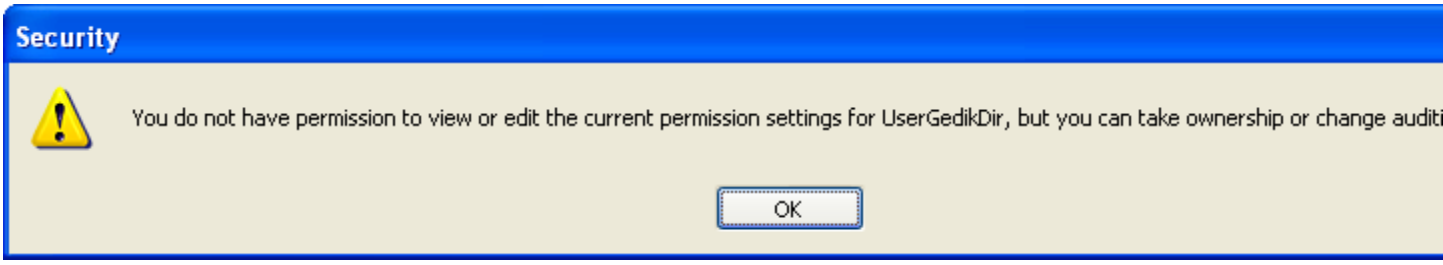

UserGedikDir Properties
General
Security
Customize
Group or user names:
Add...
Remove
Permissions
Allow
Deny
Full Control
Modify
Read & Execute
List Folder Contents
Read
Write
For special permissions or for advanced settings, click Advanced.
Advanced
OK
Cancel
Apply


UserGedikDir Properties
General
Security
Customize
Group or user names:
User Gedik Santai (NAZURI\usergedik)
Add...
Remove
Permissions for User Gedik
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

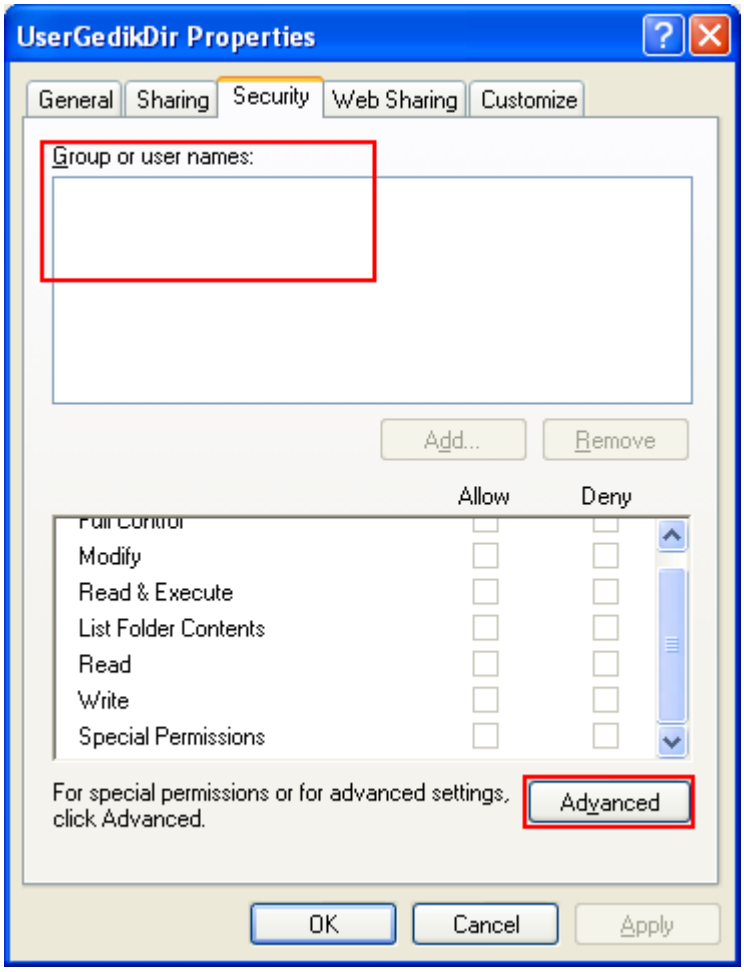


Then we log off and re login as Mike spoon, a user that is a member of Administrators group. Mike as Administrators group when trying to open (double click) the folder (owned solely by usergedik), was denied the access.

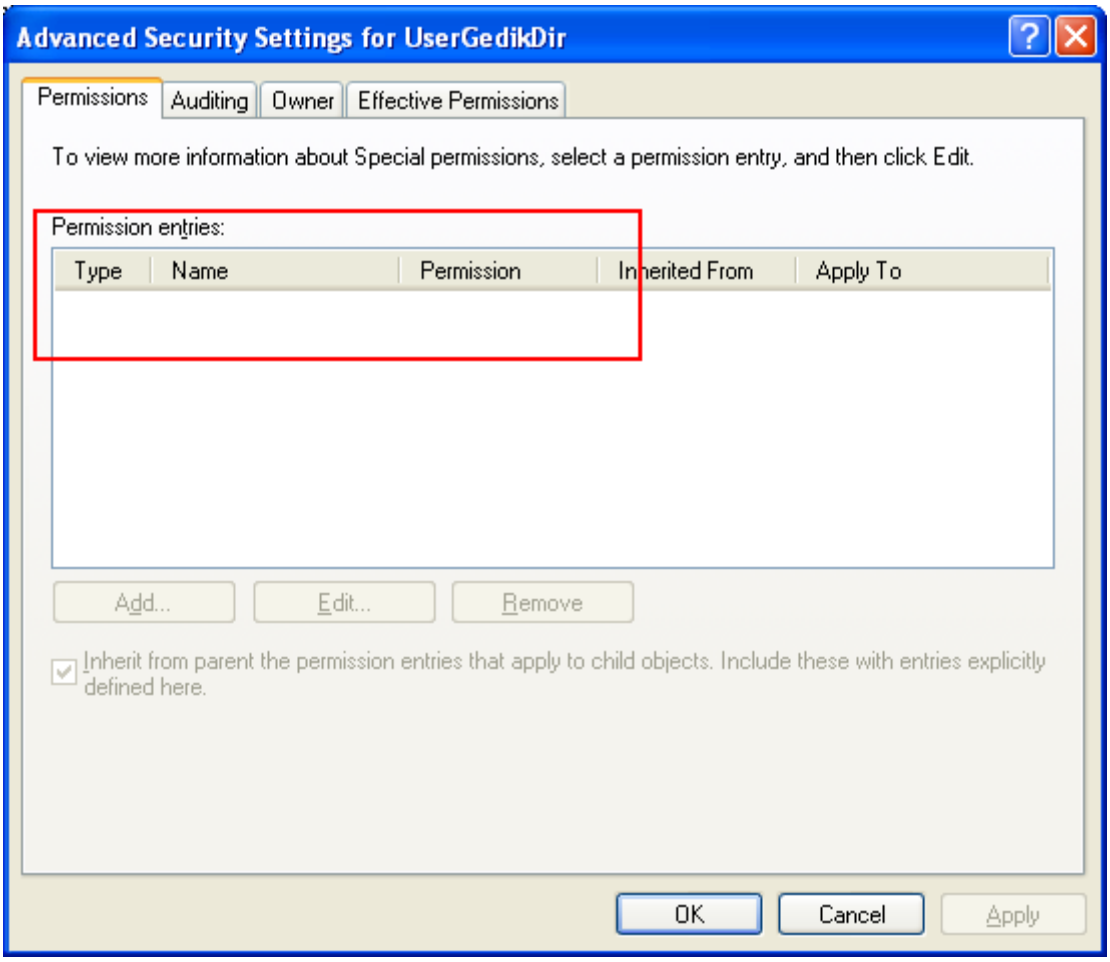
When opening the folder's property page, the following security alert should be displayed.



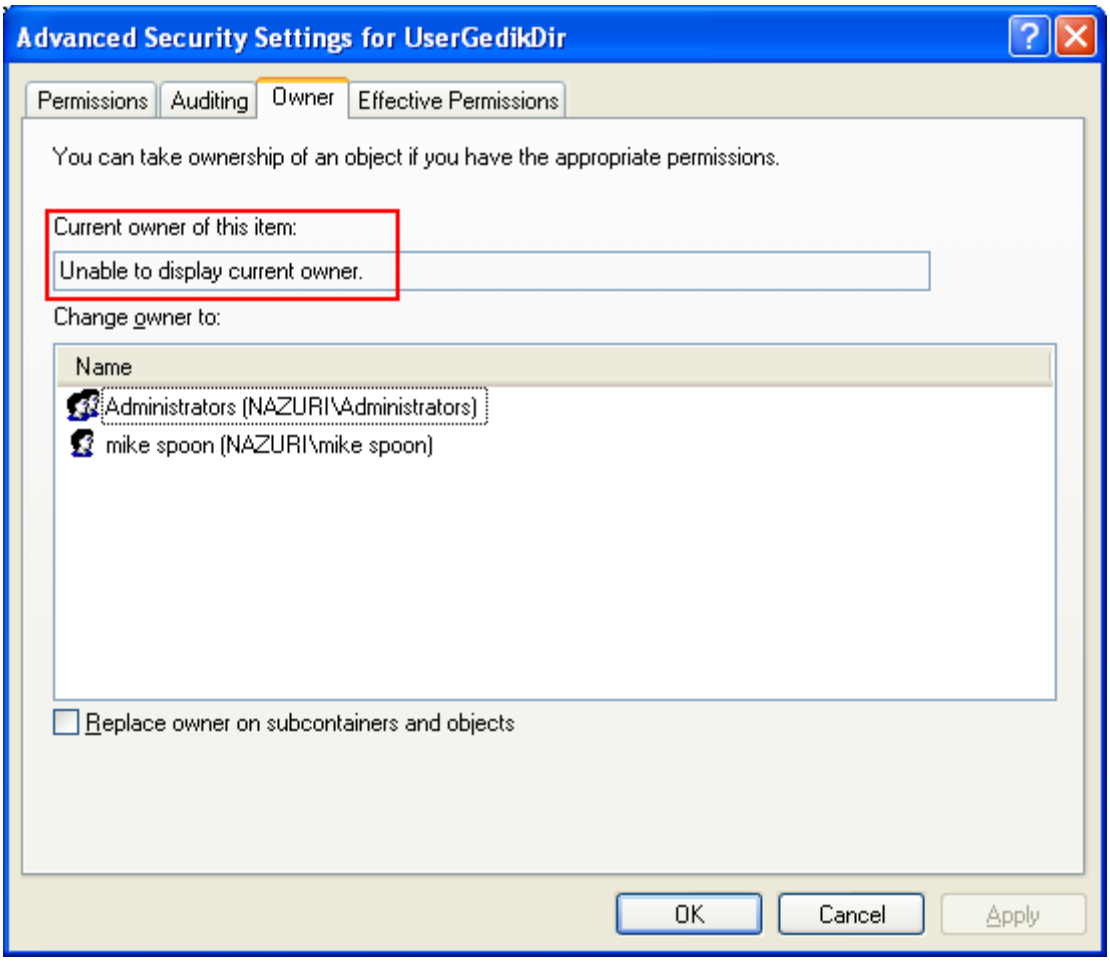
The Security page does not show any user. Click the Advanced button.



The Permission of the Advanced page also does not show any user or group.

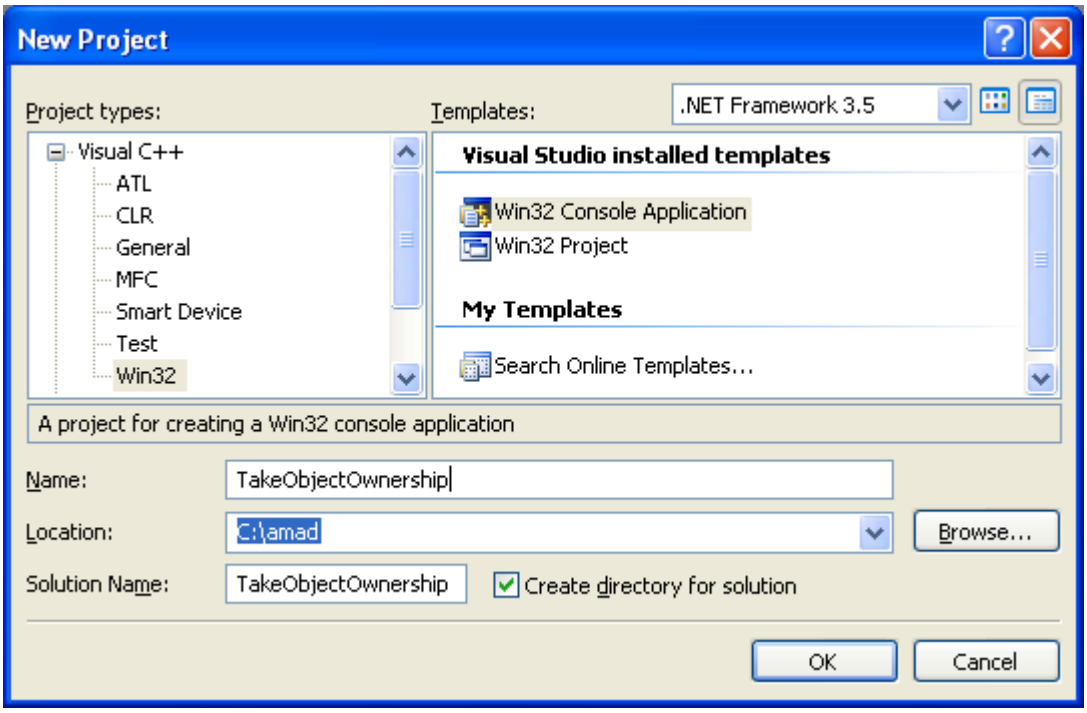


The owner also cannot be displayed. All this thing happened because Administrators group (which include Mike spoon) has been denied the access to the folder.

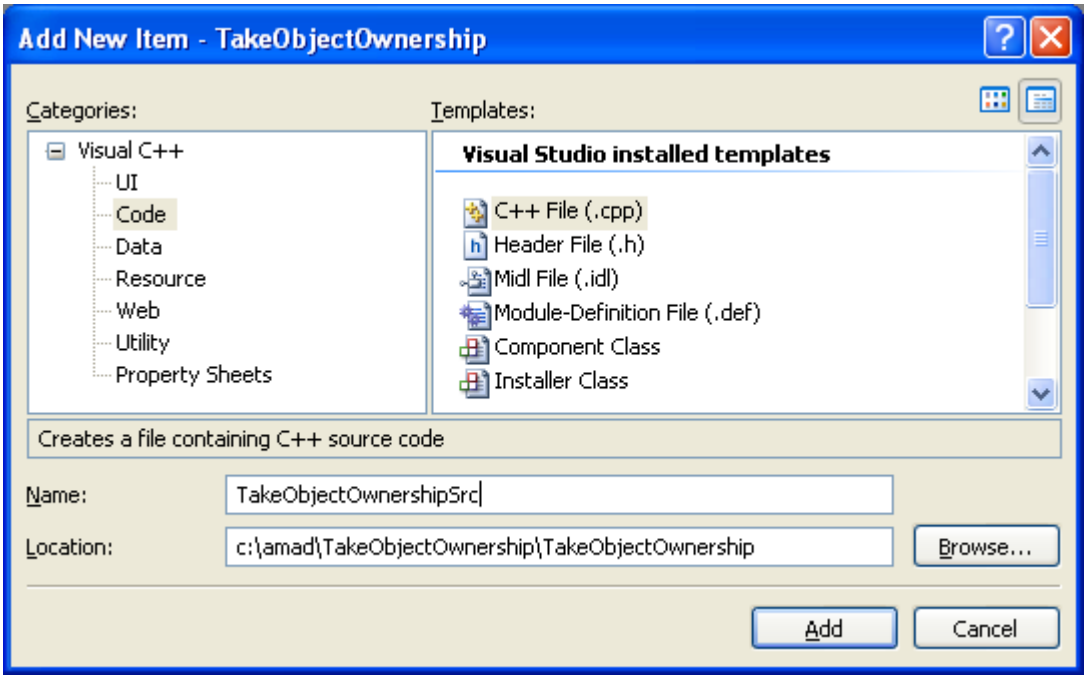


Then, Mike spoon run the following program to take the ownership of the UserGedikDir and then modify the DACL by adding Everyone group can write and Administrators group with full control.

Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.



Then, add the source file and give it a suitable name.



Next, add the following source code.

```
// Taking object ownership
#include <windows.h>
#include <stdio.h>
```

```
#include <accctrl.h>
#include <aclapi.h>

//****** Cleanup routine *******
void Cleanup(PSID pSIDAdmin, PSID pSIDEveryone, PACL pACL, HANDLE hToken)
{
      if(pSIDAdmin)
            FreeSid(pSIDAdmin);
    if(pSIDEveryone)
        FreeSid(pSIDEveryone);
    if(pACL)
       LocalFree(pACL);
    if(hToken)
       CloseHandle(hToken);
}

//************ Enable/disable the privilege ***************
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 = {0};
      LUID luid;

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

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

      // Don't forget to disable the privilege after you enable them,
      // Change the bEnablePrivilege to TRUE or FALSE to enable or
      // disable the privilege respectively
      if(bEnablePrivilege)
      {
            tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
            wprintf(L"Privilege was enabled!\n");
      }
      else
      {
            tp.Privileges[0].Attributes = 0;
            wprintf(L"Privilege was disabled!\n");
      }

      // Adjust the privilege
```

```
      if(!AdjustTokenPrivileges(
            hToken,
            FALSE,      // If TRUE, function disables all privileges,
                        //if FALSE the function modifies privileges based on
the tp
            &tp,
            sizeof(TOKEN_PRIVILEGES),
            (PTOKEN_PRIVILEGES) NULL,
            (PDWORD) NULL))
      {
            wprintf(L"AdjustTokenPrivileges() failed, error: %u\n",
GetLastError());
            return FALSE;
      }
      else
      {
            wprintf(L"AdjustTokenPrivileges() - token privilege was
adjusted!\n");
            return TRUE;
      }
}

//***** Take ownership routine ******
BOOL TakeOwnership(LPTSTR lpszDir)
{
      // Return value
    BOOL bRetval = FALSE;
    // Handle to token
    HANDLE hToken = NULL;
    // Pointer to SID for Administrators group
    PSID pSIDAdmin = NULL;
    // Pointer to SID for Everyone group
    PSID pSIDEveryone = NULL;
    // Pointer to access control list
    PACL pACL = NULL;
    SID_IDENTIFIER_AUTHORITY SIDAuthWorld = SECURITY_WORLD_SID_AUTHORITY;
    SID_IDENTIFIER_AUTHORITY SIDAuthNT = SECURITY_NT_AUTHORITY;
    const int NUM_ACCESS  = 2;
    // Explicit access arrays structure.  Here only two entries. Change the
array size for more entries
    EXPLICIT_ACCESS ea[NUM_ACCESS];
    // Result
    DWORD dwRes;

    // Specify the DACL to use.  Allow write for Everyone and full
      // control for Administrators group. Create a SID for the Everyone
group.
    if(!AllocateAndInitializeSid(&SIDAuthWorld, 1,
                    SECURITY_WORLD_RID,
                    0,
                    0, 0, 0, 0, 0, 0,
                    &pSIDEveryone))
      {
            wprintf(L"AllocateAndInitializeSid() - Failed to allocate and
initilize Everyone SID, error %u\n", GetLastError());
            Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
      }
```

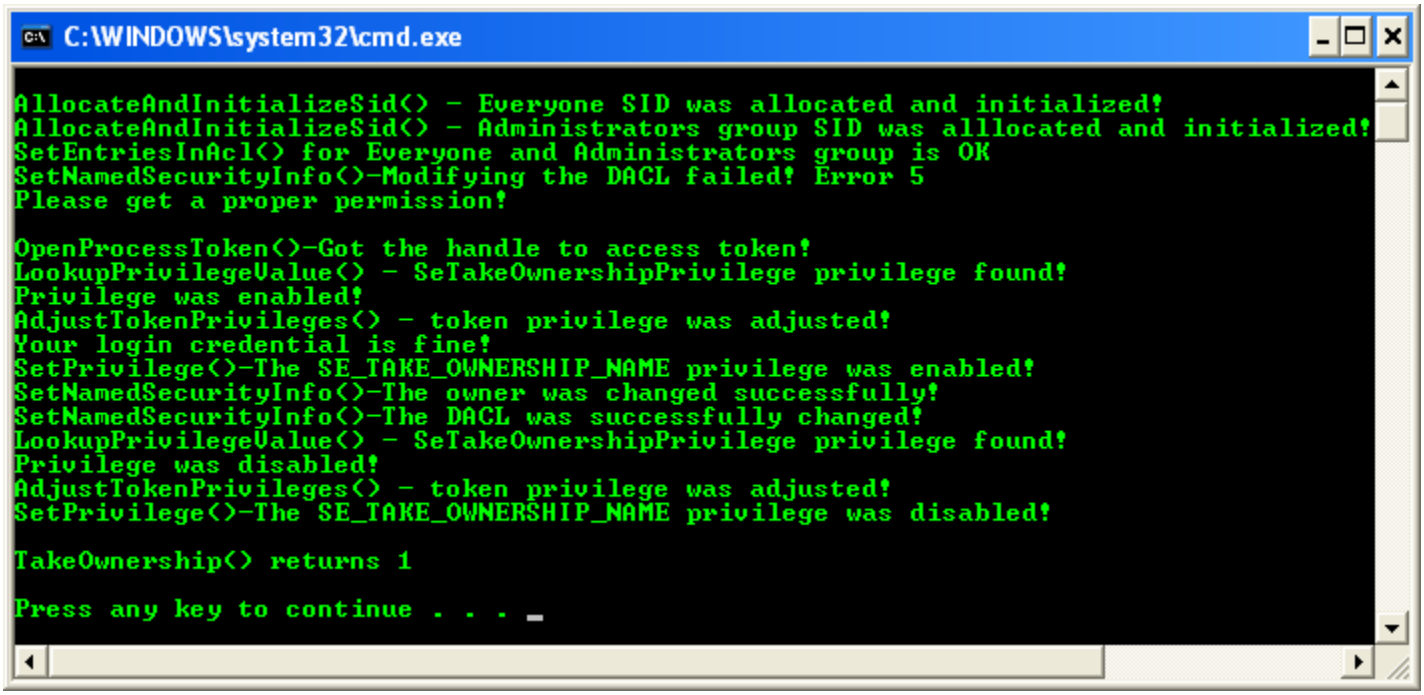
```
      else
            wprintf(L"AllocateAndInitializeSid() - Everyone SID was allocated
and initialized!\n");

      // Create a SID for the BUILTIN\Administrators group.
      if(!AllocateAndInitializeSid(&SIDAuthNT, 2,
            SECURITY_BUILTIN_DOMAIN_RID,
            DOMAIN_ALIAS_RID_ADMINS,
            0, 0, 0, 0, 0, 0,
            &pSIDAdmin))
      {
            wprintf(L"AllocateAndInitializeSid() - failed to allocate and
initialize Administrators group SID, error %u\n", GetLastError());
            Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
      }
      else
            wprintf(L"AllocateAndInitializeSid() - Administrators group SID
was alllocated and initialized!\n");

      SecureZeroMemory(&ea, NUM_ACCESS * sizeof(EXPLICIT_ACCESS));
    //****************** EXPLICIT ACCESS structure ******************
    // Construct the structure, add other entries as needed by
      // changing the array size. Set write access for Everyone.
    ea[0].grfAccessPermissions = GENERIC_WRITE;
    ea[0].grfAccessMode = SET_ACCESS;
    ea[0].grfInheritance = NO_INHERITANCE;
    ea[0].Trustee.TrusteeForm = TRUSTEE_IS_SID;
    ea[0].Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;
    ea[0].Trustee.ptstrName = (LPTSTR) pSIDEveryone;
    // Set full control for Administrators.
    ea[1].grfAccessPermissions = GENERIC_ALL;
    ea[1].grfAccessMode = SET_ACCESS;
    ea[1].grfInheritance = NO_INHERITANCE;
    ea[1].Trustee.TrusteeForm = TRUSTEE_IS_SID;
    ea[1].Trustee.TrusteeType = TRUSTEE_IS_GROUP;
    ea[1].Trustee.ptstrName = (LPTSTR) pSIDAdmin;

    //***************************************************************
    if(SetEntriesInAcl(NUM_ACCESS, ea, NULL, &pACL) != ERROR_SUCCESS)
    {
        wprintf(L"SetEntriesInAcl() for Everyone and Administrators group
failed, error %u\n", GetLastError());
        Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
    }
    else
       wprintf(L"SetEntriesInAcl() for Everyone and Administrators group is
OK\n");
    //***************************************************************

    // Try to modify the object's DACL.
    dwRes = SetNamedSecurityInfo(
        lpszDir,                   // name of the object
        SE_FILE_OBJECT,            // type of object, directory
        DACL_SECURITY_INFORMATION, // change only the object's DACL
        NULL, NULL,                // do not change owner or group
        pACL,                      // DACL specified
        NULL);                     // do not change SACL
```

```
    if(dwRes == ERROR_SUCCESS)
    {
            wprintf(L"SetNamedSecurityInfo()-Modifying the DACL is OK!\n");
            // No more processing needed. Just return/exit
       bRetval = TRUE;
       Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
      }
      else if (dwRes == ERROR_ACCESS_DENIED)
      {
            wprintf(L"SetNamedSecurityInfo()-Modifying the DACL failed! Error
%u\n", dwRes);
            wprintf(L"Please get a proper permission!\n\n");
            // If the previous call failed because access was denied,
            // enable the SE_TAKE_OWNERSHIP_NAME privilege, create a SID for
            // the Administrators group, take ownership of the object, and
            // disable the privilege. Then try again to set the object's
DACL.


      //*****************************************************************
*
            // Open a handle to the access token for the calling process
            // that is the currently login access token
            if(!OpenProcessToken(GetCurrentProcess(),
TOKEN_ADJUST_PRIVILEGES, &hToken))
            {
                  wprintf(L"OpenProcessToken() failed, error %u\n",
GetLastError());
                  Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
            }
            else
                  wprintf(L"OpenProcessToken()-Got the handle to access
token!\n");

      //*****************************************************************
*

            // Enable the SE_TAKE_OWNERSHIP_NAME privilege.
            if(!SetPrivilege(hToken, SE_TAKE_OWNERSHIP_NAME, TRUE))
            {
                  // Verify the login
                  wprintf(L"You must logged on as Administrator.\n");
                  wprintf(L"SetPrivilege()-Enable the SE_TAKE_OWNERSHIP_NAME
privilege failed, error %u\n", GetLastError());
                  Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
            }
            else
            {
                  wprintf(L"Your login credential is fine!\n");
                  wprintf(L"SetPrivilege()-The SE_TAKE_OWNERSHIP_NAME
privilege was enabled!\n");
            }


      //*****************************************************************
            // Set the new owner in the object's security descriptor.
```

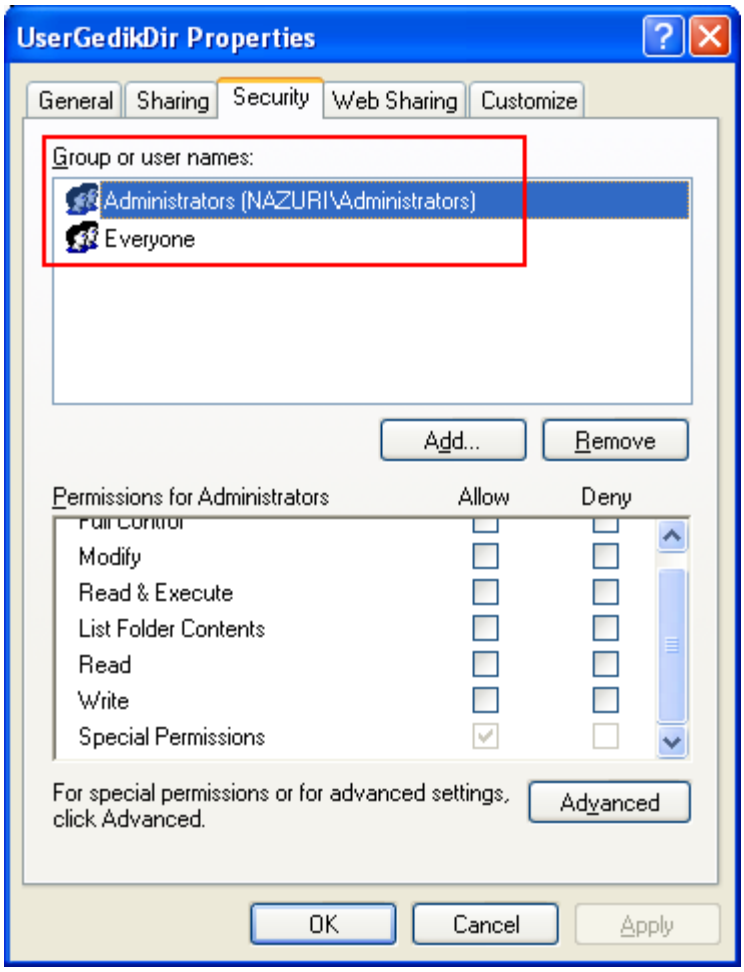
```
            dwRes = SetNamedSecurityInfo(
                  lpszDir,          // name of the object
                  SE_FILE_OBJECT,   // type of object
                  OWNER_SECURITY_INFORMATION,   // change only the object's
owner
                  pSIDAdmin,        // SID of Administrator group
                  NULL,
                  NULL,
                  NULL);

            if(dwRes != ERROR_SUCCESS)
            {
                  wprintf(L"SetNamedSecurityInfo()-Could not set the new
owner, error: %u\n", dwRes);
                  Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
            }
            else
                  wprintf(L"SetNamedSecurityInfo()-The owner was changed
successfully!\n");


      //*****************************************************************
**
            // Try again to modify the object's DACL, now that we are the
owner.
            dwRes = SetNamedSecurityInfo(
                  lpszDir,          // name of the object
                  SE_FILE_OBJECT,   // type of object
                  DACL_SECURITY_INFORMATION,    // change only the object's
DACL
                  NULL, NULL,       // do not change owner or group
                  pACL,             // DACL specified
                  NULL);                  // do not change SACL

            if(dwRes == ERROR_SUCCESS)
            {
                  wprintf(L"SetNamedSecurityInfo()-The DACL was successfully
changed!\n");
                  bRetval = TRUE;
            }
            else
            {
                  wprintf(L"SetNamedSecurityInfo()-Failed to change the DACL,
error %u\n", dwRes);
            }


      //*****************************************************************
*
            // Verify that the SE_TAKE_OWNERSHIP_NAME privilege is disabled.
            if(SetPrivilege(hToken, SE_TAKE_OWNERSHIP_NAME, FALSE))
            {
                  wprintf(L"SetPrivilege()-The SE_TAKE_OWNERSHIP_NAME
privilege was disabled!\n");
                  Cleanup(pSIDAdmin, pSIDEveryone, pACL, hToken);
            }
            else
```

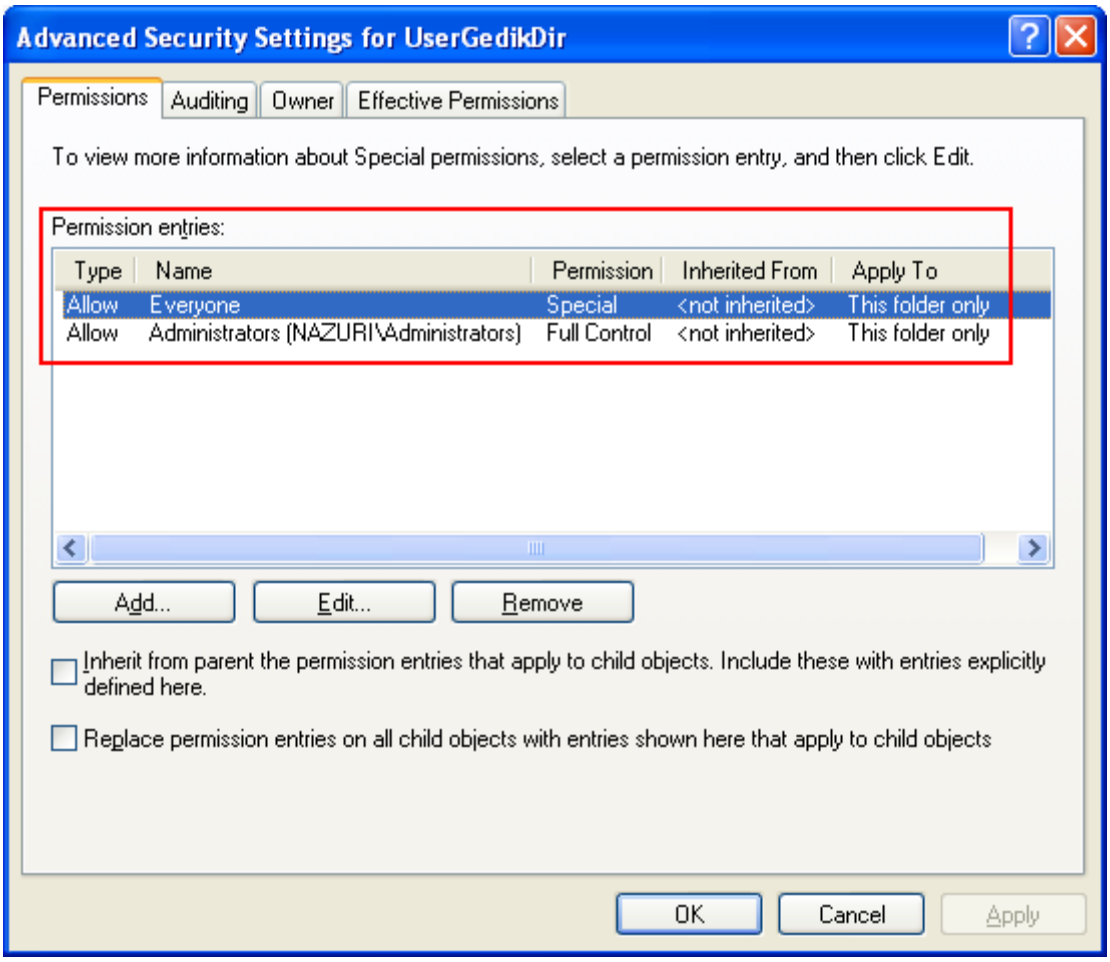
```
            {
                  wprintf(L"SetPrivilege()-Failed to disable the
SE_TAKE_OWNERSHIP_NAME privilege!\n");
                  wprintf(L"Or the privilege might be already disabled, error
%u\n", GetLastError());
            }
      }
      return bRetval;
}

//***** wmain() *******
int wmain(int argc, WCHAR **argv)
{
       LPTSTR lpszDir = L"\\\\?\\C:\\UserGedikDir";

         wprintf(L"\n");
       BOOL bRetVal = TakeOwnership(lpszDir);
       wprintf(L"\nTakeOwnership() returns %u\n\n", bRetVal);

       return 0;
}
```

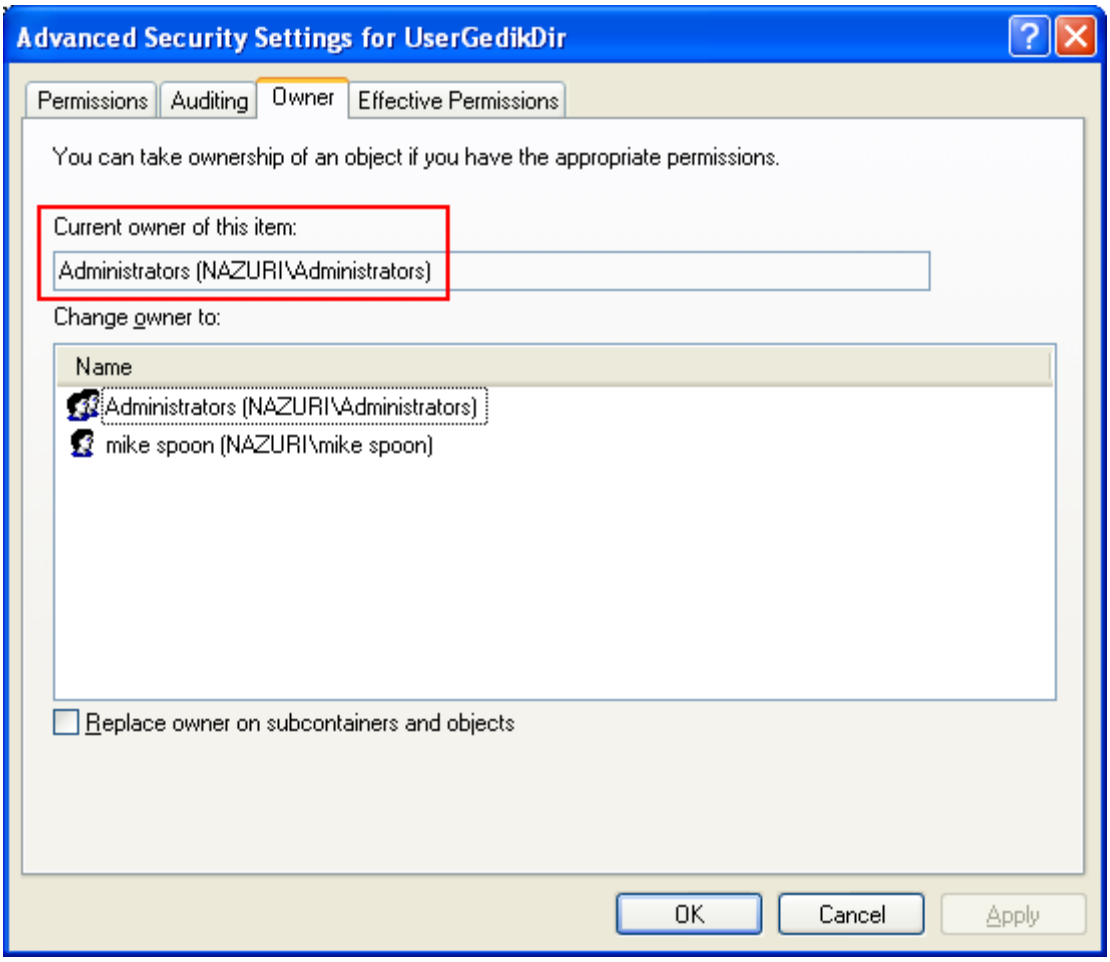
Build and run the project. The following screenshot is a sample output.



Well, let verify through the UserGedikDir property page. It seems that Mike spoon successfully accomplished his mission!


UserGedikDir Properties
General
Sharing
Security
Web Sharing
Customize
Group or user names:
Administrators (NAZURI\Administrators)
Everyone
Add...
Remove
Permissions for Administrators
Allow
Deny
Modify
Read & Execute
List Folder Contents
Read
Write
Special Permissions
For special permissions or for advanced settings, click Advanced.
Advanced
OK
Cancel
Apply


Advanced Security Settings for UserGedikDir
Permissions
Auditing
Owner
Effective Permissions
To view more information about Special permissions, select a permission entry, and then click Edit.
Permission entries:
Type
Name
Permission
Inherited From
Apply To
Allow
Everyone
Special
<not inherited>
This folder only
Allow
Administrators (NAZURI\Administrators)
Full Control
<not inherited>
This folder only
Add...
Edit...
Remove
Inherit from parent the permission entries that apply to child objects. Include these with entries explicitly defined here.
Replace permission entries on all child objects with entries shown here that apply to child objects
OK
Cancel
Apply


Advanced Security Settings for UserGedikDir
Permissions
Auditing
Owner
Effective Permissions
You can take ownership of an object if you have the appropriate permissions.
Current owner of this item:
Administrators (NAZURI\Administrators)
Change owner to:
Name
Administrators (NAZURI\Administrators)
mike spoon (NAZURI\mike spoon)
Replace owner on subcontainers and objects
OK
Cancel
Apply