# Setting up a protected folder in Windows

## Overview

This document describes how to create a protected folder in Windows that can only be accessed by an application and cannot be accessed by the current logged on user.

## Protection

This approach creates a Windows folder which can only be accessed by a specified Windows account. The current logged on user is denied access to the folder. Also, a Windows user account is created (which belongs to a specific Windows group), this user group has permission to make changes to the specified folder.

## Impersonation

Impersonation is used in the application that requires access to the protected folder. Any operations required on the protected folder can be made in code insider a using impersonation block.

## Example Code

The example code below (.NET 4 console application), illustrates the process of creating and protecting a folder:

The application:

1. Creates a user and user group which will have access to the protected folder.
2. Creates and protects a specific folder by allowing access to the user group in 1 and denying access to the current logged on user.

As this application makes changes to the users, the application must be run as administrator

using System;

using System.Collections.Generic;

using System.DirectoryServices.AccountManagement;

using System.IO;

using System.Linq;

using System.Security.AccessControl;

using System.Text;

namespace ProtectFolder

{

class Program

{

static void Main(string[] args)

{

// This application creates a new user with username "MyAssaysAccess" which belongs to the group "MyAssays Protection"

// The application then creates and protects a folder "c:\protected" - that folder than only be accessed by impersonation using the MyAssaysAccess

// the current logged on user cannot access this folder.

var context = new PrincipalContext(ContextType.Machine);

SetupSecurityGroupAndUser(context, "MyAssays Protection", "MyAssays Protection", "MyAssaysAccess", "password", "MyAssays Access");

string currentUserName = System.Security.Principal.WindowsIdentity.GetCurrent().Name;

ProtectFolder(@"c:\\protected", "MyAssaysAccess", currentUserName, true);

}

private static void SetupSecurityGroupAndUser(PrincipalContext context, string groupName, string groupDescription, string userName, string userPassword, string userDescription)

{

var group = GetOrCreateGroup(context, groupName, groupDescription);

var accountAlreadyExists = group.Members.FirstOrDefault(x => x.Name == groupName) != null;

if (!accountAlreadyExists)

{

// Add account

var userAccount = CreateNewUser(context, userName, userPassword, userDescription);

if (userAccount != null) group.Members.Add(userAccount);

group.Save();

}

}

private static GroupPrincipal GetOrCreateGroup(PrincipalContext context, string groupName, string description)

{

GroupPrincipal group = GroupPrincipal.FindByIdentity(context, groupName);

if (@group == null)

{

@group = new GroupPrincipal(context, groupName)

{

// This setting does not work on earlier versions of .NET so it has been removed here

// Description = description

};

}

return @group;

}

private static UserPrincipal CreateNewUser(PrincipalContext context, string username, string password, string description)

{

UserPrincipal existingUser = UserPrincipal.FindByIdentity(context,

IdentityType.SamAccountName,

username);

if (existingUser != null)

{

return null;

}

var user = new UserPrincipal(context)

{

Name = username,

DisplayName = username,

Description = description,

UserCannotChangePassword = true,

PasswordNeverExpires = true

};

user.SetPassword(password);

user.Save();

return user;

}

private static void ProtectFolder(string folderToProtect, string accountFullAccessAllowed, string accountDenied, bool allowDeniedToList)

{

EnsureFolderExists(folderToProtect);

var directoryInfo = new DirectoryInfo(folderToProtect);

// Get a DirectorySecurity object that represents the current security settings.

DirectorySecurity directorySecurity = directoryInfo.GetAccessControl();

// Apply the account has FULL access to this folder

directorySecurity.AddAccessRule(

new FileSystemAccessRule(accountFullAccessAllowed,

fileSystemRights: FileSystemRights.FullControl,

inheritanceFlags:

InheritanceFlags.ContainerInherit | InheritanceFlags.ObjectInherit,

propagationFlags: PropagationFlags.None,

type: AccessControlType.Allow)

);

if (!string.IsNullOrEmpty(accountDenied))

{

// Other users can only list files

if (allowDeniedToList)

{

directorySecurity.AddAccessRule(

new FileSystemAccessRule(accountDenied, // Instead of just AccountUserNameWizUser

fileSystemRights: FileSystemRights.ListDirectory,

inheritanceFlags:

InheritanceFlags.ContainerInherit | InheritanceFlags.ObjectInherit,

propagationFlags: PropagationFlags.None,

type: AccessControlType.Allow)

);

}

directorySecurity.AddAccessRule(

new FileSystemAccessRule(accountDenied,

fileSystemRights:

FileSystemRights.Delete |

FileSystemRights.CreateFiles |

FileSystemRights.CreateDirectories |

FileSystemRights.AppendData |

FileSystemRights.ReadExtendedAttributes |

FileSystemRights.WriteExtendedAttributes |

FileSystemRights.ExecuteFile |

// FileSystemRights.Traverse |

FileSystemRights.DeleteSubdirectoriesAndFiles |

FileSystemRights.WriteAttributes |

FileSystemRights.ChangePermissions |

FileSystemRights.TakeOwnership,

inheritanceFlags:

InheritanceFlags.ContainerInherit | InheritanceFlags.ObjectInherit,

propagationFlags: PropagationFlags.None,

type: AccessControlType.Deny)

);

}

// Set the new access settings.

directoryInfo.SetAccessControl(directorySecurity);

}

private static void EnsureFolderExists(string folder)

{

if (!Directory.Exists(folder))

{

Directory.CreateDirectory(folder);

}

}

}

}