Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Защита информации в сети Internet»

на тему «Атрибуты безопасности в ОС Windows»

Вариант 2

Выполнили  
студенты группы 18ВВ1:

Коротов В.О.

Максимова Д.Р.

Приняли:

к.т.н., доцент Дубравин А.В.

к.т.н., доцент Карамышева Н.С.

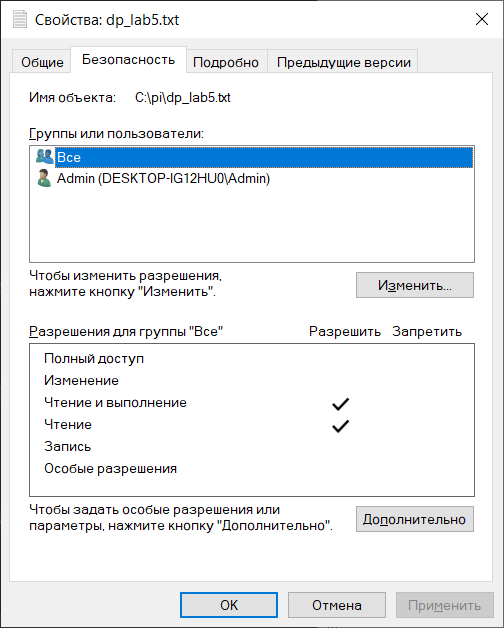
**2020**

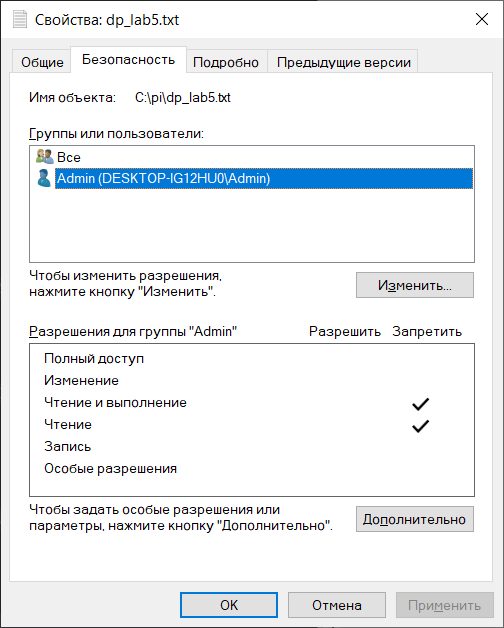
**Цель работы:** Разработать программу, выполняющую создание файла с атрибутами безопасности для группы пользователей «Все» и правами на чтение и исполнение, а для текущего пользователя следует запретить чтение и исполнение.

**Задание:**

|  |  |
| --- | --- |
| Номер варианта | Метод генерации случайных чисел |
| 2 | Разработать программу, выполняющую создание файла с атрибутами безопасности для группы пользователей «Все» и правами на чтение и исполнение, а для текущего пользователя следует запретить чтение и исполнение. |

**Описание работы программы:** создали атрибуты безопасности для группы пользователей «Все» и для текущего пользователя, после этого использовали их для создания файлов.

Результат работы программы:



Листинг:

#include <iostream>

#include <Windows.h>

#include <aclapi.h>

#include <Lmwksta.h>

#include <StrSafe.h>

PSID pCurrentUserSID;

BOOL GetCurrentUserAndDomain(PTSTR szUser, PDWORD pcchUser,

PTSTR szDomain, PDWORD pcchDomain)

{

BOOL fSuccess = FALSE;

HANDLE hToken = NULL;

PTOKEN\_USER ptiUser = NULL;

DWORD cbti = 0;

SID\_NAME\_USE snu;

\_\_try

{

// Get the calling thread's access token.

if (!OpenThreadToken(GetCurrentThread(), TOKEN\_QUERY, TRUE,

&hToken))

{

if (GetLastError() != ERROR\_NO\_TOKEN)

\_\_leave;

// Retry against process token if no thread token exists.

if (!OpenProcessToken(GetCurrentProcess(), TOKEN\_QUERY,

&hToken))

\_\_leave;

}

// Obtain the size of the user information in the token.

if (GetTokenInformation(hToken, TokenUser, NULL, 0, &cbti))

{

// Call should have failed due to zero-length buffer.

\_\_leave;

}

else

{

// Call should have failed due to zero-length buffer.

if (GetLastError() != ERROR\_INSUFFICIENT\_BUFFER)

\_\_leave;

}

// Allocate buffer for user information in the token.

ptiUser = (PTOKEN\_USER)HeapAlloc(GetProcessHeap(), 0, cbti);

if (!ptiUser)

\_\_leave;

// Retrieve the user information from the token.

if (!GetTokenInformation(hToken, TokenUser, ptiUser, cbti, &cbti))

\_\_leave;

// Retrieve user name and domain name based on user's SID.

if (!LookupAccountSid(NULL, ptiUser->User.Sid, szUser, pcchUser,

szDomain, pcchDomain, &snu))

\_\_leave;

pCurrentUserSID = ptiUser->User.Sid;

fSuccess = TRUE;

}

\_\_finally

{

// Free resources.

if (hToken)

CloseHandle(hToken);

if (ptiUser)

HeapFree(GetProcessHeap(), 0, ptiUser);

}

return fSuccess;

}

int main()

{

PSID pEveryoneSID = NULL, pAdminSID = NULL;

PACL pACL = NULL;

PSECURITY\_DESCRIPTOR pSecurityDescriptor = NULL;

EXPLICIT\_ACCESS ea[2] = { 0 };

SID\_IDENTIFIER\_AUTHORITY SIDAuthWorld = SECURITY\_WORLD\_SID\_AUTHORITY;

SID\_IDENTIFIER\_AUTHORITY SIDAuthLocal = SECURITY\_LOCAL\_SID\_AUTHORITY;

SID\_IDENTIFIER\_AUTHORITY SIDAuthNT = SECURITY\_NT\_AUTHORITY;

SID\_IDENTIFIER\_AUTHORITY SIDAuthCreator = SECURITY\_CREATOR\_SID\_AUTHORITY;

TCHAR user[1024], domain[1024];

DWORD chUser = sizeof(user), chDomain = sizeof(domain);

if (!GetCurrentUserAndDomain(user, &chUser, domain, &chDomain))

{

printf("GetCurrentUserSID is failed\n");

}

try

{

// создаем SID для всех пользователей

if (!AllocateAndInitializeSid(&SIDAuthWorld, 1, SECURITY\_WORLD\_RID,

0, 0, 0, 0, 0, 0, 0, &pEveryoneSID))

{

throw std::exception("Can't create SID for current user!\n");

}

//// для администраторов

//if (!AllocateAndInitializeSid(

// &SIDAuthNT,

// 2, SECURITY\_BUILTIN\_DOMAIN\_RID, DOMAIN\_ALIAS\_RID\_ADMINS,

// 0, 0, 0, 0, 0, 0,

// &pAdminSID))

//{

// throw std::exception("Can't create SID for admins!\n");

//}

// заполняем структуры EXPLICIT\_ACCESS

ZeroMemory(&ea, 2 \* sizeof(EXPLICIT\_ACCESS));

// 0 - для текущего

ea[0].grfAccessPermissions = GENERIC\_READ | GENERIC\_EXECUTE; // разрешение - ставим для чтения & выполненения

ea[0].grfAccessMode = DENY\_ACCESS; // для установки доступа

ea[0].grfInheritance = NO\_INHERITANCE; // без разрешения наследования

ea[0].Trustee.TrusteeForm = TRUSTEE\_IS\_SID; // форма структуры

ea[0].Trustee.TrusteeType = TRUSTEE\_IS\_USER; // группа, с которой мы будем работать

ea[0].Trustee.ptstrName = (LPWSTR)pCurrentUserSID; // указатель на название

// 1 - для всех

ea[1].grfAccessPermissions = GENERIC\_READ | GENERIC\_EXECUTE; // разрешение - ставим полное разрешение(чтение, запись)

ea[1].grfAccessMode = SET\_ACCESS; // для установки доступа

ea[1].grfInheritance = NO\_INHERITANCE; // без разрешения наследования

ea[1].Trustee.TrusteeForm = TRUSTEE\_IS\_SID; // форма структуры

ea[1].Trustee.TrusteeType = TRUSTEE\_IS\_WELL\_KNOWN\_GROUP;// группа, с которой мы будем работать

ea[1].Trustee.ptstrName = (LPWSTR)pEveryoneSID; // указатель на название

// создаем Acess control list, содержащий выше проинициализированные записи

if (SetEntriesInAcl(

2, // количество ea

ea,

NULL, // старый ACL

&pACL) // новый ACL

!= ERROR\_SUCCESS)

{

int a = GetLastError();

throw std::exception("Can't create ACL!\n ,%d", a);

}

// Инициализируем SECURITY\_DESCRIPTOR

pSecurityDescriptor = (PSECURITY\_DESCRIPTOR)LocalAlloc(LPTR, SECURITY\_DESCRIPTOR\_MIN\_LENGTH);

if (!pSecurityDescriptor)

{

throw std::exception("Can't create pointer to security descriptor!\n");

}

if (!InitializeSecurityDescriptor(pSecurityDescriptor, SECURITY\_DESCRIPTOR\_REVISION))

{

throw std::exception("can't create security descriptor!\n");

}

// Добавляем ACL к SECURITY\_DESCRIPTOR

if (!SetSecurityDescriptorDacl(

pSecurityDescriptor, // указатель на SECURITY\_DESCRPITOR

TRUE, // Есть ли ACL

pACL, // Сам ACL

FALSE) // ACL по умолчанию

)

{

throw std::exception("Can't add ACL to security descriptor!\n");

}

// создаем структуры SECURITY\_ATTRIBUTES

SECURITY\_ATTRIBUTES securityAttributes = { 0 };

securityAttributes.nLength = sizeof(SECURITY\_ATTRIBUTES);

securityAttributes.lpSecurityDescriptor = pSecurityDescriptor;

securityAttributes.bInheritHandle = FALSE;

// теперь создаем файлы с использованием атрибутов безопасности

HANDLE hFile1 = CreateFile(L"C:\\pi\\dp\_lab5.txt", GENERIC\_ALL, FILE\_SHARE\_READ | FILE\_SHARE\_DELETE,

&securityAttributes, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_ARCHIVE, NULL);

CloseHandle(hFile1);

}

catch (const std::exception& e)

{

printf("Error %s\n", e.what());

}

// очищаем все поля

if (pEveryoneSID)

FreeSid(pEveryoneSID);

if (pAdminSID)

FreeSid(pAdminSID);

if (pACL)

LocalFree(pACL);

if (pSecurityDescriptor)

LocalFree(pSecurityDescriptor);

//const size\_t INFO\_BUFFER\_SIZE = 2560;

//DWORD bufCharCount = INFO\_BUFFER\_SIZE;

//TCHAR infoBuf[INFO\_BUFFER\_SIZE];

// if (!GetUserName(infoBuf, &bufCharCount))

// printf("GetUserName %d", GetLastError());

// printf("Username is %s", infoBuf);

//

// bufCharCount = INFO\_BUFFER\_SIZE;

// if (!GetComputerName(infoBuf, &bufCharCount))

// printf("GetComputerName");

// printf("\nComputer name: %s", infoBuf);

//

// if (!GetSystemDirectory(infoBuf, INFO\_BUFFER\_SIZE))

// printf("GetSystemDirectory");

// printf("\nSystem Directory: %s", infoBuf);

//

// // Get and display the Windows directory.

// if (!GetWindowsDirectory(infoBuf, INFO\_BUFFER\_SIZE))

// printf("GetWindowsDirectory");

// printf("\nWindows Directory: %s", infoBuf);

//

// DWORD bufSize = MAX\_PATH;

// TCHAR domainNameBuf[MAX\_PATH];

// GetComputerNameEx(ComputerNameDnsDomain, domainNameBuf, &bufSize);

// printf("ComputerNameEx [%s]", domainNameBuf);

return 0;

}**Вывод:** в ходе работы была разработана программа, выполняющая создание файла с атрибутами безопасности для группы пользователей «Все» и правами на чтение и исполнение, а для текущего пользователя было запрещено чтение и исполнение.