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»

Выполнили  
студенты группы 18ВВ1:

Ручкин М.А.

Коновалова Д.А.

Дубинин А.В.

Приняли:

к.т.н., доцент Дубравин А.В.

к.т.н., доцент Карамышева Н.С.

2020

**Цель работы:** изучить атрибуты безопасности в ОС Windows.

**Задания:** Разработать программу, отображающую для раздела реестра, указанного в параметрах командной строки, информацию о пользователях и доступных для них правах доступа.

**Листинг**:

#include <iostream>

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#include <sddl.h>

#include <wchar.h>

#include <tchar.h>

#include <locale.h>

#include <string.h>

using namespace std;

const char \*sidToTextTok(PSID psid)

{

// S-rev- + SIA + subauthlen\*maxsubauth + terminator

static char buf[15 + 12 + 12 \* SID\_MAX\_SUB\_AUTHORITIES + 1];

char \*p = &buf[0];

PSID\_IDENTIFIER\_AUTHORITY psia;

DWORD numSubAuths, i;

// Validate the binary SID.

if (!IsValidSid(psid))

return FALSE;

psia = GetSidIdentifierAuthority(psid);

p = buf;

p += \_snprintf\_s(p, 15 + 12 + 12 \* SID\_MAX\_SUB\_AUTHORITIES + 1, &buf[sizeof buf] - p, "S-%lu-", 0x0f & \*((byte \*)psid));

if ((psia->Value[0] != 0) || (psia->Value[1] != 0))

p += \_snprintf\_s(p, 15 + 12 + 12 \* SID\_MAX\_SUB\_AUTHORITIES + 1, &buf[sizeof buf] - p, "0x\_hx\_hx\_hx\_hx\_hx\_hx",

(USHORT)psia->Value[0], (USHORT)psia->Value[1],

(USHORT)psia->Value[2], (USHORT)psia->Value[3],

(USHORT)psia->Value[4], (USHORT)psia->Value[5]);

else

p += \_snprintf\_s(p, 15 + 12 + 12 \* SID\_MAX\_SUB\_AUTHORITIES + 1, &buf[sizeof buf] - p, "%lu", (ULONG)(psia->Value[5]) +

(ULONG)(psia->Value[4] << 8) + (ULONG)(psia->Value[3] << 16) +

(ULONG)(psia->Value[2] << 24));

// Add SID subauthorities to the string.

numSubAuths = \*GetSidSubAuthorityCount(psid);

for (i = 0; i < numSubAuths; ++i)

p += \_snprintf\_s(p, 15 + 12 + 12 \* SID\_MAX\_SUB\_AUTHORITIES + 1, &buf[sizeof buf] - p, "-%lu", \*GetSidSubAuthority(psid, i));

return buf;

}

void PrintPermissionsTok(PACL DACL)

{

DWORD dwCount = 0;

ACCESS\_ALLOWED\_ACE \*ACE;

if (IsValidAcl(DACL) == TRUE) {

fprintf(stdout, "[i] |\n");

fprintf(stdout, "[i] +-+-> Default DACL for new objects created by this user\n");

// Now for each ACE in the DACL

for (dwCount = 0; dwCount < DACL->AceCount; dwCount++) {

if (GetAce(DACL, dwCount, (LPVOID\*)&ACE)) {

SID \*sSID = (SID\*)&(ACE->SidStart);

if (sSID != NULL)

{

DWORD dwSize = 2048;

char lpName[2048];

char lpDomain[2048];

SID\_NAME\_USE SNU;

switch (ACE->Header.AceType) {

// Allowed ACE

case ACCESS\_ALLOWED\_ACE\_TYPE:

// Lookup the account name and print it.

if (!LookupAccountSidA(NULL, sSID, lpName, &dwSize, lpDomain, &dwSize, &SNU)) {

DWORD dwResult = GetLastError();

if (dwResult == ERROR\_NONE\_MAPPED) {

fprintf(stdout, "[%d] |\n", dwCount);

fprintf(stdout, "[%d] +---+-> Allowed - NONMAPPED - SID %s\n", dwCount, sidToTextTok(sSID));

}

else if (dwResult != ERROR\_NONE\_MAPPED) {

fprintf(stderr, "[!] LookupAccountSid Error %u\n", GetLastError());

fprintf(stdout, "[%d] |\n", dwCount);

fprintf(stdout, "[%d] +---+-> Allowed - ERROR - SID %s\n", dwCount, sidToTextTok(sSID));

//return;

}

else {

continue;

}

}

else {

fprintf(stdout, "[%d] |\n", dwCount);

fprintf(stdout, "[%d] +---+-> Allowed - %s\\%s\n", dwCount, lpDomain, lpName);

char strUserFromPID[1024];

memset(strUserFromPID, 0x00, 1024);

if (!strcmp(lpDomain, "BUILTIN") == 0 && !strcmp(lpName, "OWNER RIGHTS") == 0 && !strcmp(lpDomain, "NT AUTHORITY") == 0 && !strcmp(lpDomain, "NT SERVICE") == 0) {

fprintf(stdout, "[%d] |\n", dwCount);

fprintf(stdout, "[%d] +-+-+-> Alert!\n", dwCount);

}

}

// print out the ACE mask

fprintf(stdout, "[%d] |\n", dwCount);

fprintf(stdout, "[%d] +-> Permissions - ", dwCount);

if (ACE->Mask & GENERIC\_ALL) fprintf(stdout, "| All");

if (ACE->Mask & GENERIC\_EXECUTE) fprintf(stdout, "| Execute");

if (ACE->Mask & GENERIC\_READ) fprintf(stdout, "| Read");

if (ACE->Mask & GENERIC\_WRITE) fprintf(stdout, "| Write");

if (ACE->Mask & DELETE) fprintf(stdout, "| Delete");

if (ACE->Mask & READ\_CONTROL) fprintf(stdout, "| Read control");

if (ACE->Mask & SYNCHRONIZE) fprintf(stdout, "| Sync");

if (ACE->Mask & WRITE\_DAC) fprintf(stdout, "| Modify DACL");

if (ACE->Mask & WRITE\_OWNER) fprintf(stdout, "| Write Owner");

if (ACE->Mask & STANDARD\_RIGHTS\_ALL) fprintf(stdout, "| All");

if (ACE->Mask & STANDARD\_RIGHTS\_EXECUTE) fprintf(stdout, "| Execute");

if (ACE->Mask & STANDARD\_RIGHTS\_READ) fprintf(stdout, "| Read Control");

if (ACE->Mask & STANDARD\_RIGHTS\_REQUIRED) fprintf(stdout, "| Write");

if (ACE->Mask & STANDARD\_RIGHTS\_WRITE) fprintf(stdout, "| Read Control");

fprintf(stdout, "\n");

break;

// Denied ACE

case ACCESS\_DENIED\_ACE\_TYPE:

break;

// Uh oh

default:

break;

}

}

}

else {

DWORD dwError = GetLastError();

fprintf(stderr, "[!] Error - %d - GetAce\n", dwError);

return;

}

}

}

else {

DWORD dwError = GetLastError();

fprintf(stderr, "[!] Error - %d - IsValidAcl\n", dwError);

return;

}

}

int getDacleString(HKEY root, LPCWSTR subkey)

{

SECURITY\_INFORMATION lable = DACL\_SECURITY\_INFORMATION;

HKEY hkey;

DWORD psdsize = 1;

REGSAM sam = KEY\_READ | KEY\_ENUMERATE\_SUB\_KEYS;

LONG err = RegOpenKeyEx(root, subkey, 0, sam, &hkey);

if (err != ERROR\_SUCCESS)

{

if (err == ERROR\_FILE\_NOT\_FOUND) {

printf("Key not found.\n");

return TRUE;

}

else {

printf("Error opening key.\n");

return FALSE;

}

}

if (err == 0 && hkey != 0) {

PSID group;

BOOL groupDefaulted = 0;

LPSTR groupSidString;

RegGetKeySecurity(hkey, lable, NULL, &psdsize);

PSECURITY\_DESCRIPTOR psd = LocalAlloc(LMEM\_FIXED, psdsize);

RegGetKeySecurity(hkey, lable, psd, &psdsize);

LPBOOL lpbDaclPresent;

PACL pAcl;

LPVOID pAclInformation;

DWORD cbAclInfo;

ACL\_INFORMATION\_CLASS dwAclInfoClass;

PSID pOwnerSID = NULL;

PSID pGroupSID = NULL;

PACL pDACL = NULL;

PACL pSACL = NULL;

BOOL bOwnerDefaulted = FALSE;

BOOL bGroupDefaulted = FALSE;

BOOL bDaclPresent = FALSE;

BOOL bDaclDefaulted = FALSE;

BOOL bSaclPresent = FALSE;

BOOL bSaclDefaulted = FALSE;

if (psd) {

GetSecurityDescriptorOwner(psd, &pOwnerSID, &bOwnerDefaulted);

GetSecurityDescriptorGroup(psd, &pGroupSID, &bGroupDefaulted);

GetSecurityDescriptorDacl(psd, &bDaclPresent, &pDACL, &bDaclDefaulted);

GetSecurityDescriptorSacl(psd, &bSaclPresent, &pSACL, &bSaclDefaulted);

}

PrintPermissionsTok(pDACL);

}

else return NULL;

}

static HKEY Translate(char\* key)

{

if (strcmp(key, "HKEY\_LOCAL\_MACHINE"))

return HKEY\_LOCAL\_MACHINE;

else if (strcmp(key, "HKEY\_CLASSES\_ROOT"))

return HKEY\_CLASSES\_ROOT;

else if (strcmp(key, "HKEY\_USERS"))

return HKEY\_USERS;

else if (strcmp(key, "HKEY\_CURRENT\_CONFIG"))

return HKEY\_CURRENT\_CONFIG;

else if (strcmp(key, "HKEY\_CURRENT\_USER"))

return HKEY\_CURRENT\_USER;

return NULL;

}

wchar\_t \*convertCharArrayToLPCWSTR(const char\* charArray)

{

wchar\_t\* wString = new wchar\_t[4096];

MultiByteToWideChar(CP\_ACP, 0, charArray, -1, wString, 4096);

return wString;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Rus");

if (argc == 3)

{

printf("\n1\n");

getDacleString(Translate(argv[1]), convertCharArrayToLPCWSTR(argv[2]));

}

else if (argc == 2)

{

printf("\n2\n");

getDacleString(Translate(argv[1]), NULL);

}

else

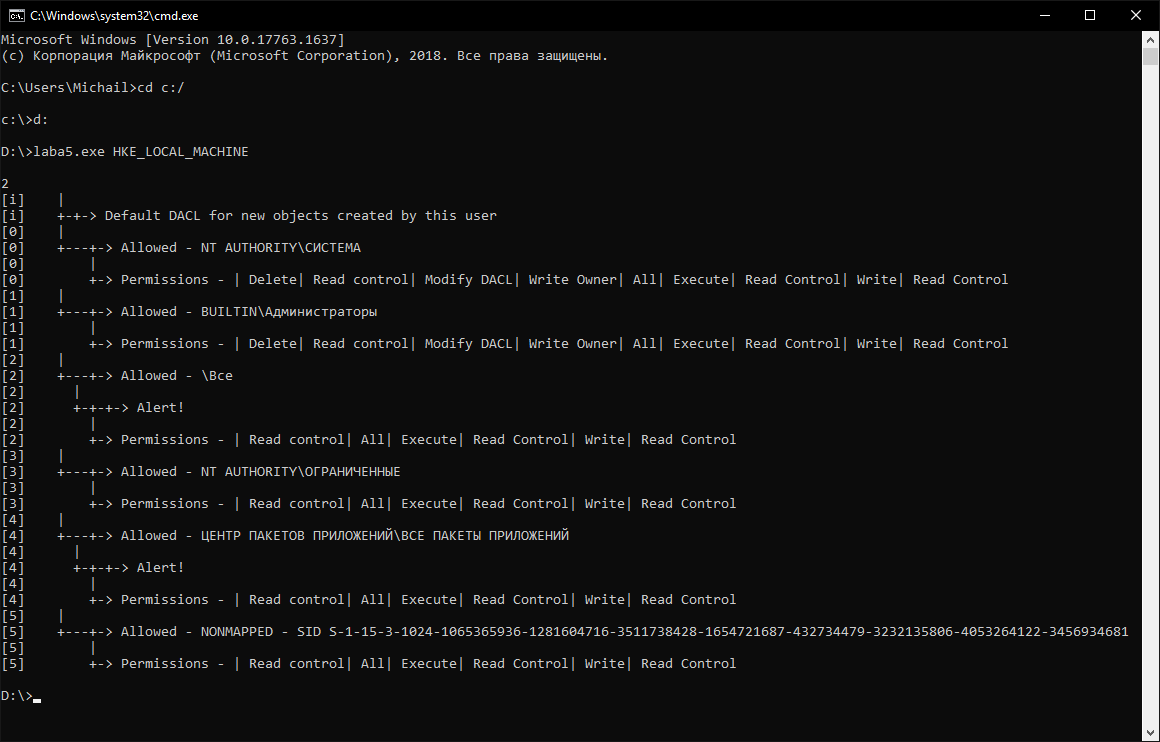
{

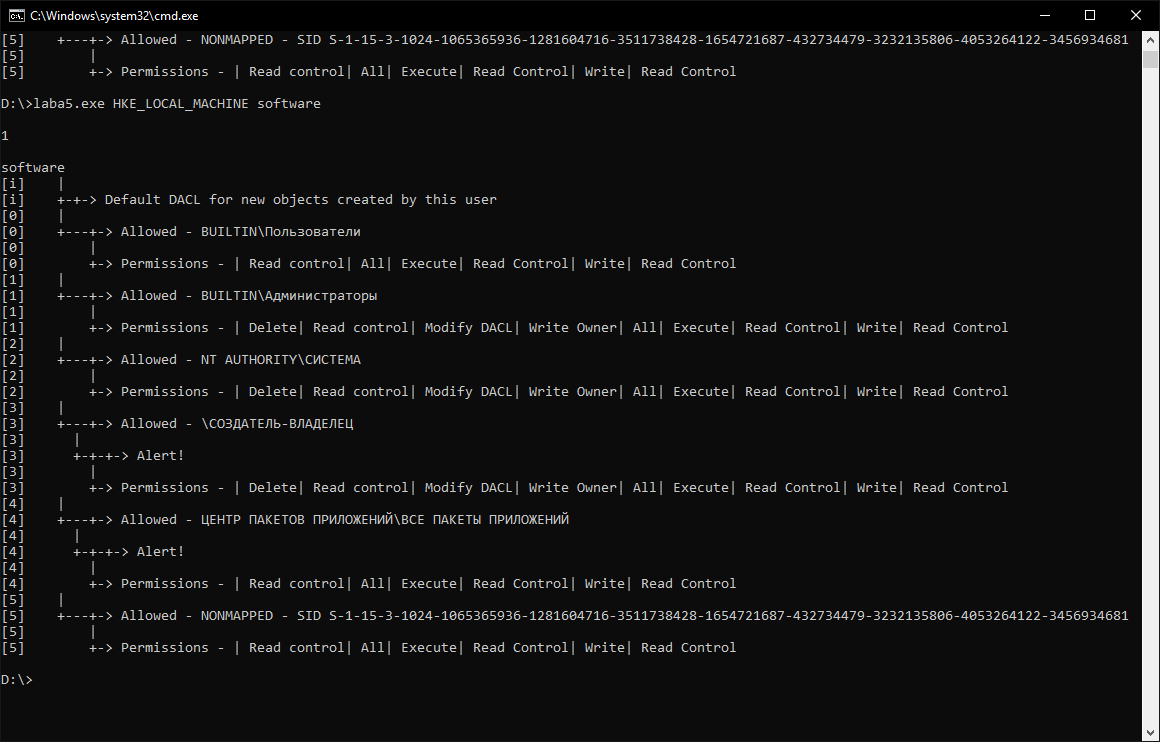
printf("\nNot arguments or too many arguments\n");

}

}

**Вывод программы**:





**Вывод:** в ходе лабораторной работы изучены атрибуты безопасности в ОС Windows.