Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Защита информации в сети Internet»

на тему «Cтеганография»

Выполнили:  
студенты группы 18ВВ1:  
Федулов В.

Пчелинцев А.

Приняли:

к.т.н. Дубравин А.В.

Пенза 2020

**Цель работы**: разработка программы шифрования/дешифрирования текста с использованием алгоритма стеганографии.

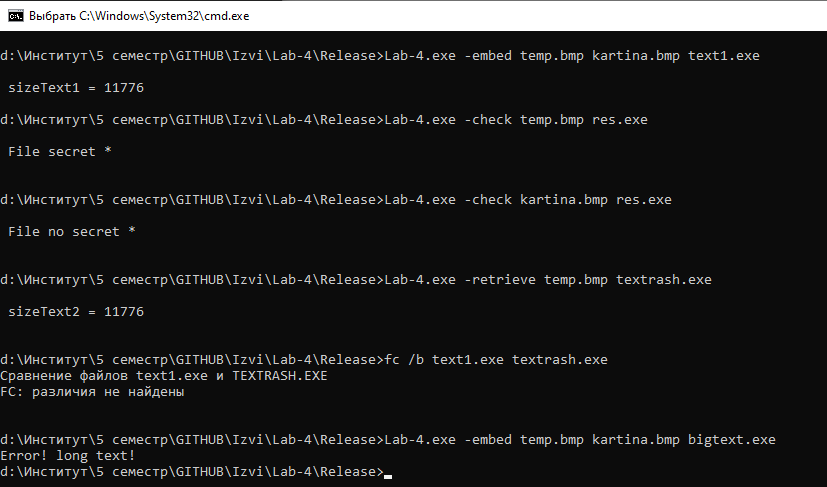
**Задание:** разработать программу, выполняющую внедрение, извлечение и проверку наличия некоторых данных в файле с изображением (формат файла с изображением — BMP, не содержащий палитру и имеющий значение biBitCount равное 24).

Тип операции (внедрение, извлечение и проверка), а также имена файлов, участвующих в каждой операции следует передавать в программу через командную строку.

**Последовательность перестановки:**

|  |  |
| --- | --- |
| Командная строка | Описание действия |
| steg.exe -embed secret.txt picture.bmp | Помещает «секретное» содержимое файла secret.txt в файл picture.bmp |
| steg.exe -retrieve picture.bmp secret.txt | Извлекает «секретное» содержимое из файла picture.bmp и помещает его в файл secret.txt |
| steg.exe -check picture.bmp | Проверяет наличие «секретного» содержимого в файле picture.bmp |

**Результат работы программы:**



**Листинг:**

#include <stdio.h>

#include <conio.h>

#include <windows.h>

bool CreateBMP(LPTSTR szFileName, LPTSTR szFileNameOld, LPTSTR szFileNameText, bool newFile)

{

int sizeText = 0;

RGBTRIPLE color;

BYTE\* h[3] = { &color.rgbtBlue, &color.rgbtGreen, &color.rgbtRed };

BITMAPFILEHEADER bfh;

BITMAPINFOHEADER bih;

DWORD RW;

HANDLE hFile, hFile2 = NULL;

BYTE Palette[1024];

memset(Palette, 0, 1024);

if (newFile) {

int Width = 50;

int Height = 50;

memset(&bfh, 0, sizeof(bfh));

bfh.bfType = 0x4D42;

bfh.bfOffBits = sizeof(bfh) + sizeof(bih) /\*+ 1024\*/;

bfh.bfSize = bfh.bfOffBits + sizeof(color) \* Width \* Height + Height \* ((sizeof(color) \* Width) % 4); // Ïîñ÷èòàå ðàçìåð êîíå÷íîãî ôàéë

memset(&bih, 0, sizeof(bih));

bih.biSize = sizeof(bih);

bih.biBitCount = 24;

bih.biClrUsed = 0;

bih.biCompression = BI\_RGB;

bih.biHeight = Height;

bih.biWidth = Width;

bih.biPlanes = 1;

bih.biSizeImage = bfh.bfSize - bfh.bfOffBits;

hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, 0, NULL);

if (hFile == INVALID\_HANDLE\_VALUE)

return false;

WriteFile(hFile, &bfh, sizeof(bfh), &RW, NULL);

WriteFile(hFile, &bih, sizeof(bih), &RW, NULL);

}

else

{

hFile2 = CreateFile(szFileNameOld, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0 /\*FILE\_FLAG\_SEQUENTIAL\_SCAN\*/, NULL);

if (NULL == hFile2)

return false;

hFile = CreateFile(szFileName, GENERIC\_WRITE, FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, 0 /\*FILE\_FLAG\_SEQUENTIAL\_SCAN\*/, NULL);

if (!ReadFile(hFile2, &bfh, sizeof(BITMAPFILEHEADER), &RW, NULL)

|| !ReadFile(hFile2, &bih, sizeof(BITMAPINFOHEADER), &RW, NULL)

|| bfh.bfType != 'MB'

|| bih.biCompression != BI\_RGB

|| bih.biBitCount != 24)

{

CloseHandle(hFile);

CloseHandle(hFile2);

return false;

}

WriteFile(hFile, &bfh, sizeof(bfh), &RW, NULL);

WriteFile(hFile, &bih, sizeof(bih), &RW, NULL);

}

int temp, sizeText2 = (bih.biHeight \* bih.biWidth \* 3 / 8) - 6, pos = bfh.bfOffBits;

for (int i = 0; i < bih.biHeight; ++i)

{

for (int j = 0; j < bih.biWidth; ++j)

{

if (newFile) {

color.rgbtRed = rand() % 254;

color.rgbtGreen = rand() % 254;

color.rgbtBlue = rand() % 254;

}

else {

ReadFile(hFile2, &color, sizeof(color), &RW, NULL);

}

WriteFile(hFile, &color, sizeof(color), &RW, NULL);

}

WriteFile(hFile, &color, (sizeof(color) \* bih.biWidth) % 4, &RW, NULL);

}

SetFilePointer(hFile2, bfh.bfOffBits, NULL, FILE\_BEGIN);

char buff[3];

HANDLE hFile3 = CreateFile(szFileNameText, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_ALWAYS, 0, NULL);

DWORD s[2] = { GetFileSize(hFile3, &s[1]) };

sizeText = (s[1] << 16) + s[0];

if (sizeText2 < sizeText) {

printf("Error! long text!");

CloseHandle(hFile);

CloseHandle(hFile2);

CloseHandle(hFile3);

return false;

}

for (char i = 0; i < 2; i++) {

temp = sizeText;

for (char j = 0; j < 8; j++) {

if (newFile) {

color.rgbtRed = rand() % 254;

color.rgbtGreen = rand() % 254;

color.rgbtBlue = rand() % 254;

}

else {

ReadFile(hFile2, &color, sizeof(color), &RW, NULL);

}

for (char k = 0; k < 3; k++)

{

if (temp & (1 << (23 - k)))

\*h[k] |= (1 << 0);

else

\*h[k] &= ~(1 << 0);

}

temp = temp << 3;

SetFilePointer(hFile, pos, NULL, FILE\_BEGIN);

WriteFile(hFile, &color, sizeof(color), &RW, NULL);

pos += 3;

}

}

printf("\n sizeText1 = %d \n %", sizeText);

for (int j = 0, p = 3, g = 24; j < sizeText; j += 3) {

if (j + 3 > sizeText) {

p = sizeText % 3;

g = p \* 8;

}

ReadFile(hFile3, &buff, p, &RW, NULL);

for (char k = 0; k < p; k++) {

temp = temp + ((unsigned char)buff[k] << 8 \* (2 - k));

}

for (char k = 0; k < 8 && p == 3 || p != 3 && g > 0; k++)

{

if (newFile) {

color.rgbtRed = rand() % 254;

color.rgbtGreen = rand() % 254;

color.rgbtBlue = rand() % 254;

}

else {

ReadFile(hFile2, &color, sizeof(color), &RW, NULL);

}

for (char t = 0; t < 3; t++)

{

if (g > 0 || p == 3) {

if (p != 3)

g--;

if (temp & (1 << (23 - t)))

\*h[t] |= (1 << 0);

else

\*h[t] &= ~(1 << 0);

}

else {

\*h[t] &= ~(1 << 0);

}

}

temp = temp << 3;

SetFilePointer(hFile, pos, NULL, FILE\_BEGIN);

WriteFile(hFile, &color, sizeof(color), &RW, NULL);

pos += 3;

}

}

CloseHandle(hFile);

CloseHandle(hFile2);

CloseHandle(hFile3);

}

bool OpenBMP(LPTSTR szFileName, LPTSTR szFileNameText, bool testFile)

{

RGBTRIPLE color;

BYTE\* h[3] = { &color.rgbtBlue, &color.rgbtGreen, &color.rgbtRed };

BITMAPFILEHEADER bfh;

BITMAPINFOHEADER bih;

DWORD RW;

HANDLE hFile = CreateFile(szFileName, GENERIC\_READ, FILE\_SHARE\_READ || FILE\_SHARE\_WRITE, NULL, OPEN\_ALWAYS, FILE\_FLAG\_SEQUENTIAL\_SCAN, NULL);

if (NULL == hFile)

return false;

if (!ReadFile(hFile, &bfh, sizeof(BITMAPFILEHEADER), &RW, NULL)

|| !ReadFile(hFile, &bih, sizeof(BITMAPINFOHEADER), &RW, NULL)

|| bfh.bfType != 'MB'

|| bih.biCompression != BI\_RGB

|| bih.biBitCount != 24)

{

CloseHandle(hFile);

return false;

}

SetFilePointer(hFile, bfh.bfOffBits, NULL, FILE\_BEGIN);

int sizeText = (bih.biWidth \* abs(bih.biHeight) \* 3) / 8 - 6, temp, key;

HANDLE hFile3 = CreateFile(szFileNameText, GENERIC\_WRITE, FILE\_SHARE\_WRITE, NULL, CREATE\_ALWAYS, 0, NULL);

for (char i = 0; i < 2; i++) {

temp = 0;

for (char j = 0; j < 8; j++) {

ReadFile(hFile, &color, sizeof(color), &RW, NULL);

temp = temp << 3;

for (char k = 0; k < 3; k++) {

if (\*h[2 - k] & (1 << 0))

temp |= (1 << k);

else

temp &= ~(1 << k);

}

}

if (i == 0)

sizeText = temp;

else

key = temp;

}

if (key != sizeText)

return false;

if (key == sizeText && testFile) {

CloseHandle(hFile);

CloseHandle(hFile3);

return true;

}

printf("\n sizeText2 = %d \n\n", sizeText);

char buff[3];

for (int j = 0, p = 3, g = 24; j < sizeText; j += 3) {

if (j + 3 > sizeText) {

p = sizeText % 3;

g = p \* 8;

}

temp = 0;

for (char k = 0; k < 8 && p == 3 || p != 3 && g > 0; k++)

{

ReadFile(hFile, &color, sizeof(color), &RW, NULL);

temp = temp << 3;

for (char t = 0; t < 3; t++)

{

if (g > 0 || p == 3) {

if (p != 3)

g--;

if (\*h[t] & (1 << 0))

temp |= (1 << (2 - t));

else

temp &= ~(1 << (2 - t));

}

else {

temp = temp << ((3 - p) \* 8 - p);

break;

}

}

}

for (char k = 0; k < 3; k++) {

buff[k] = temp >> (8 \* (2 - k));

}

WriteFile(hFile3, &buff, p, &RW, NULL);

}

CloseHandle(hFile);

CloseHandle(hFile3);

return true;

}

int wmain(int argc, wchar\_t\* argv[])

{

if (wcscmp(argv[1], L"-embed") == 0) {

CreateBMP(argv[2], argv[3], argv[4], false);

}

else if (wcscmp(argv[1], L"-retrieve") == 0) {

OpenBMP(argv[2], argv[3], false);

}

else if (wcscmp(argv[1], L"-check") == 0) {

if (OpenBMP(argv[2], argv[3], true) && true)

printf("\n File secret \*\n\n");

else if (true)

printf("\n File no secret \*\n\n");

}

\_getch();

return 0;

}

**Вывод:** в ходе данной лабораторной работы, мы создали программу шифрования/дешифрования текста с использованием алгоритма стегранографии.