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 «Стеганография»

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:** изучить методы внедрения, извлечения и проверки некоторых данных в файле с изображением в формате BMP.

**Задание:** Разработать программу, выполняющую внедрение, извлечение и проверку наличия некоторых данных в файле с изображением (формат файла с изображением — BMP, не содержащий плитру 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.

**Описание работы программы:** были написаны 3 функции, для внедрения, извлечения и проверки наличия некоторых данных в файле. Размер секретного сообщения записывается в саму картинку. Далее побитно записываем сообщение в файл bmp. В процессе извлечения данных, считываем размер секретного сообщения, а после проходимся по всем битам картинки, записывая последний бит каждой компоненты в файл. В функции проверки наличия секретного сообщения сравниваем данные в области памяти с размером секретного сообщения, если там записан 0, то картинка не имеет в себе секретных данных.

**Результат работы программы:**

Команды:

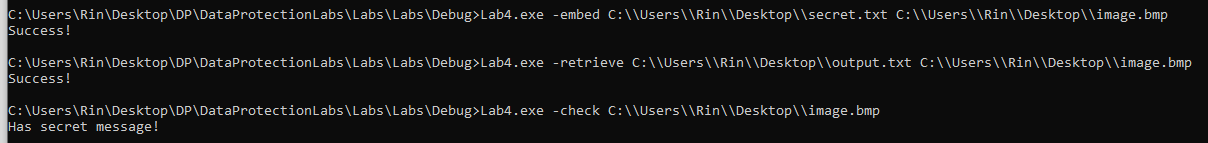


Рис 1. Работа в командной строке

Результат выполнения

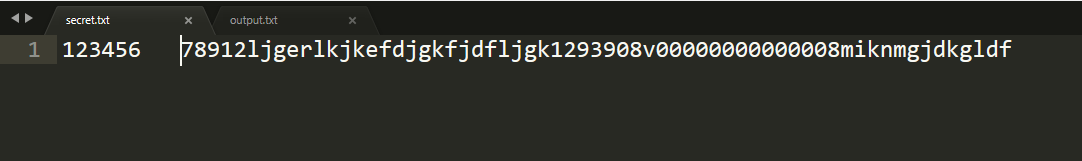


Рис 2. Файл с секретным содержимым

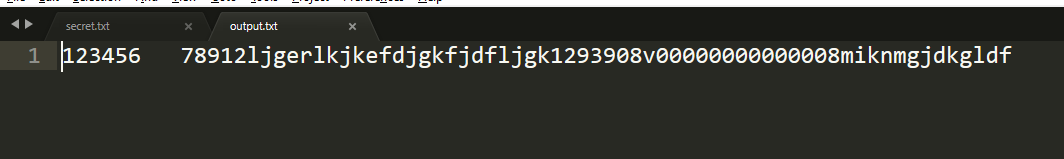


Рис 3. Файл с содержимым после извлечения

Листинг:

#include "pch.h"

#include <iostream>

#include <Windows.h>

#include <fstream>

#include <bitset>

#include <string>

#include <vector>

#include <queue>

#include <conio.h>

using namespace std;

const size\_t BITS\_IN\_BYTE = 8;

const size\_t WORD\_SIZE = 16;

const size\_t DWORD\_SIZE = 32;

const size\_t BUFFER\_SIZE = 3;

ostream& operator<<(ostream& os, const RGBTRIPLE &color)

{

os << static\_cast<int>(color.rgbtRed) <<

" " << static\_cast<int>(color.rgbtGreen) <<

" " << static\_cast<int>(color.rgbtBlue);

return os;

}

ostream& operator<<(ostream& os, const vector<bool> &vec)

{

for (int i = 0; i < vec.size(); i++)

{

os << vec[i];

}

return os;

}

ostream& operator<<(ostream& os, const BITMAPFILEHEADER bfh)

{

cout << "bfh bfType " << bfh.bfType << endl;

cout << "bfh bfOffBits " << bfh.bfOffBits << endl;

cout << "bfh bfReserved1 " << bfh.bfReserved1 << endl;

cout << "bfh bfReserved2 " << bfh.bfReserved2 << endl;

cout << "bfh bfSize " << bfh.bfSize << endl << endl;

return os;

}

ostream& operator<<(ostream& os, const BITMAPINFOHEADER bih)

{

cout << "bih biBitCount " << bih.biBitCount << endl;

cout << "bih biClrImportant " << bih.biClrImportant << endl;

cout << "bih biClrUsed " << bih.biClrUsed << endl;

cout << "bih biCompression " << bih.biCompression << endl;

cout << "bih biWidth " << bih.biWidth << endl;

cout << "bih biHeight " << bih.biHeight << endl;

cout << "bih biSize " << bih.biSize << endl;

cout << "bih biPlanes " << bih.biPlanes << endl;

cout << "bih biSizeImage " << bih.biSizeImage << endl << endl;

return os;

}

void check\_handle(HANDLE &handle)

{

if (INVALID\_HANDLE\_VALUE == handle)

{

cout << "Invalid handle value " << GetLastError() << endl;

exit(1);

}

}

void str\_to\_bits(string &str, queue<bool> &bits)

{

for (int i = 0; i < str.length(); i++)

{

bitset<BITS\_IN\_BYTE> sym\_bits(str[i]);

for (int j = 0; j < BITS\_IN\_BYTE; j++)

{

bits.push(sym\_bits[j]);

}

}

}

void change\_pixel(RGBTRIPLE &pixel, queue<bool> &bits)

{

bitset<BITS\_IN\_BYTE> rgb\_bs[3]{

bitset<BITS\_IN\_BYTE>(pixel.rgbtRed),

bitset<BITS\_IN\_BYTE>(pixel.rgbtGreen),

bitset<BITS\_IN\_BYTE>(pixel.rgbtBlue)

};

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

for (int i = 0; i < 3 && !bits.empty(); i++)

{

rgb\_bs[i][0] = bits.front(); // change last bit

\*(color\_byte[i]) = (BYTE)(rgb\_bs[i].to\_ulong());

bits.pop();

}

}

void get\_secret\_bits(RGBTRIPLE &pixel, vector<bool> &secret\_bits)

{

bitset<BITS\_IN\_BYTE> rgb\_bs[3]{

bitset<BITS\_IN\_BYTE>(pixel.rgbtRed),

bitset<BITS\_IN\_BYTE>(pixel.rgbtGreen),

bitset<BITS\_IN\_BYTE>(pixel.rgbtBlue)

};

for (int i = 0; i < 3; i++)

{

secret\_bits.push\_back(rgb\_bs[i][0]); // last bit

}

}

FILE\* open\_file(const char\* filename, const char\* mode)

{

FILE \*file = fopen(filename, mode);

if (!file)

{

printf("Can't open file %s!", filename);

exit(1);

}

return file;

}

void add\_byte(queue<bool> &bits, byte \*buffer)

{

bitset<BITS\_IN\_BYTE> bs(\*buffer);

for (int i = 0; i < BITS\_IN\_BYTE; i++)

{

bits.push(bs[i]);

}

}

queue<bool> get\_str\_from\_file(string &filename)

{

FILE\* file = open\_file(filename.c\_str(), "rb");

queue<bool> bits;

byte \*buffer = new byte[BUFFER\_SIZE \* sizeof(byte)];

while (!feof(file))

{

size\_t element\_count = fread\_s(buffer, sizeof(byte) \* BUFFER\_SIZE, sizeof(byte), BUFFER\_SIZE, file);

if (element\_count == 0) break;

for (size\_t i = 0; i < element\_count; i++)

{

add\_byte(bits, &(buffer[i]));

}

}

free(buffer);

fclose(file);

return bits;

}

bool is\_bmp(BITMAPFILEHEADER bfh)

{

return (bfh.bfType == 0x4D42);

}

void add\_secret\_msg(string secret\_fname, string image\_fname)

{

queue<bool> secret\_bits = get\_str\_from\_file(secret\_fname);

size\_t msg\_len = secret\_bits.size() / 8; // length in bytes

if (msg\_len > UINT\_MAX)

{

cout << "Message is too big!\n";

exit(1);

}

HANDLE hFile = CreateFileA(image\_fname.c\_str(), GENERIC\_READ | GENERIC\_WRITE, 0, NULL,

OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

check\_handle(hFile);

OVERLAPPED olf = { 0 };

DWORD bytes = 0;

// read structure BITMAPFILEHEADER

BITMAPFILEHEADER bfh;

ReadFile(hFile, &bfh, sizeof(bfh), &bytes, &olf);

if (!is\_bmp(bfh))

{

cout << "Not BMP image!\n";

CloseHandle(hFile);

exit(1);

}

bitset<DWORD\_SIZE> msg\_len\_bs(msg\_len);

bitset<WORD\_SIZE> bs\_high(msg\_len\_bs.to\_string().substr(0, WORD\_SIZE));

bitset<WORD\_SIZE> bs\_low(msg\_len\_bs.to\_string().substr(WORD\_SIZE, WORD\_SIZE));

bfh.bfReserved1 = (WORD)bs\_high.to\_ulong();

bfh.bfReserved2 = (WORD)bs\_low.to\_ulong();

WriteFile(hFile, &bfh, sizeof(bfh), &bytes, &olf);

olf.Offset += bytes;

// read structure BITMAPINFOHEADER

BITMAPINFOHEADER bih;

ReadFile(hFile, &bih, sizeof(bih), &bytes, &olf);

// check messag size

LARGE\_INTEGER image\_max\_bits{ 0 };

image\_max\_bits.QuadPart = bih.biWidth;

image\_max\_bits.QuadPart \*= bih.biHeight;

image\_max\_bits.QuadPart \*= 3;

if (image\_max\_bits.QuadPart <= secret\_bits.size())

{

cout << "Size of message is too big!\n";

olf.Offset = 0;

bfh.bfReserved1 = 0;

bfh.bfReserved2 = 0;

WriteFile(hFile, &bfh, sizeof(bfh), &bytes, &olf);

CloseHandle(hFile);

exit(1);

}

olf.Offset += bytes;

RGBTRIPLE pixel;

while (!secret\_bits.empty())

{

ReadFile(hFile, &pixel, sizeof(pixel), &bytes, &olf);

change\_pixel(pixel, secret\_bits);

WriteFile(hFile, &pixel, sizeof(pixel), &bytes, &olf);

olf.Offset += bytes;

}

cout << "Success!\n";

CloseHandle(hFile);

}

bool has\_secret\_msg(string image\_fname)

{

HANDLE hFile = CreateFileA(image\_fname.c\_str(), GENERIC\_READ, 0, NULL,

OPEN\_EXISTING, 0, NULL);

check\_handle(hFile);

DWORD bytes = 0;

BITMAPFILEHEADER bfh;

ReadFile(hFile, &bfh, sizeof(bfh), &bytes, NULL);

if (!is\_bmp(bfh))

{

cout << "Not BMP image!\n";

exit(1);

}

CloseHandle(hFile);

return (bfh.bfReserved1 != 0 || bfh.bfReserved2 != 0);

}

void get\_secret\_msg(string output, string image\_fname)

{

if (!has\_secret\_msg(image\_fname))

{

cout << "No secret message!\n";

exit(1);

}

vector<bool> secret\_bits;

HANDLE hFile = CreateFileA(image\_fname.c\_str(), GENERIC\_READ, 0, NULL,

OPEN\_EXISTING, 0, NULL);

check\_handle(hFile);

DWORD bytes = 0;

// read structure BITMAPFILEHEADER

BITMAPFILEHEADER bfh;

ReadFile(hFile, &bfh, sizeof(bfh), &bytes, NULL);

// get size of secret msg

WORD high = bfh.bfReserved1;

WORD low = bfh.bfReserved2;

bitset<DWORD\_SIZE> bs;

bitset<WORD\_SIZE> high\_bs(high);

bitset<WORD\_SIZE> low\_bs(low);

for (int i = 0; i < WORD\_SIZE; i++)

{

bs[WORD\_SIZE + i] = high\_bs[i];

bs[i] = low\_bs[i];

}

size\_t msg\_bits\_len = (size\_t)bs.to\_ulong() \* BITS\_IN\_BYTE;

// read structure BITMAPINFOHEADER

BITMAPINFOHEADER bih;

ReadFile(hFile, &bih, sizeof(bih), &bytes, NULL);

RGBTRIPLE pixel;

// read secret message

for (int i = 0; i < msg\_bits\_len; i += 3)

{

ReadFile(hFile, &pixel, sizeof(pixel), &bytes, NULL);

get\_secret\_bits(pixel, secret\_bits);

}

ofstream secret(output);

// write secret msg to file

size\_t len = secret\_bits.size() / BITS\_IN\_BYTE;

for (int i = 0; i < len; i++)

{

bitset<BITS\_IN\_BYTE> sym\_bits;

for (int j = 0; j < BITS\_IN\_BYTE; j++)

{

sym\_bits[j] = secret\_bits[i\*BITS\_IN\_BYTE + j];

}

//cout << "Get symbits: " << (char)sym\_bits.to\_ulong() << endl;

secret << (char)sym\_bits.to\_ulong();

}

cout << "Success!\n";

CloseHandle(hFile);

secret.close();

}

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

{

if (argc == 1)

{

cout << "No arguments\n";

exit(1);

}

if (strcmp(argv[1], "-embed") == 0)

{

if (argc == 4) add\_secret\_msg(argv[2], argv[3]);

else cout << "Wrong arguments!\n";

}

else if (strcmp(argv[1], "-retrieve") == 0)

{

if (argc == 4) get\_secret\_msg(argv[2], argv[3]);

else cout << "Wrong arguments!\n";

}

else if (strcmp(argv[1], "-check") == 0)

{

if (argc == 3)

{

if (has\_secret\_msg(argv[2])) cout << "Has secret message!\n";

else cout << "No secret message!\n";

}

else cout << "Wrong arguments! " << argc << "!\n";

}

else

{

cout << "Wrong command!\n";

exit(1);

}

/\*add\_secret\_msg(

"C:\\Users\\Rin\\Desktop\\secret.txt",

"C:\\Users\\Rin\\Desktop\\image.bmp"

);

get\_secret\_msg(

"C:\\Users\\Rin\\Desktop\\output.txt",

"C:\\Users\\Rin\\Desktop\\image.bmp"

);

if (has\_secret\_msg("C:\\Users\\Rin\\Desktop\\image.bmp")) cout << "Secret\n";\*/

return 0;

}

**Вывод:** изучили методы внедрения, извлечения и проверки некоторых данных в файле с изображением в формате BMP.