Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Клиент-серверное приложение для обмена сообщениями

**Пояснительная записка**

RU. 02068048.502900.01.01

#### АННОТАЦИЯ

В данном программном документе представлен технический проект «Клиент-серверное приложение для обмена сообщениями» - предназначенный для обмена текстовыми сообщениями между пользователями. Текст программы реализован на языке C++ в интегрированной среде разработки Microsoft Visual Studio 2017.

Основной функцией проекта является обеспечение обмена текстовыми сообщениями между пользователями. Система состоит из двух частей: клиентской, устанавливаемой на машине пользователя и серверной, находящейся на специально отведенной машине заказчика. С помощью клиентской части, пользователи могут подключиться к серверу для дальнейшего обмена сообщениями с остальными пользователями. Программа защищена от динамического и статического анализа.

**СОДЕРЖАНИЕ**

1. ОПИСАНИЕ ПОСТАНОВКИ ЗАДАЧ…………......................................................4

1.1 Характеристики комплекса задач…………………………………..............4

1.2 Выходная информация…………………………………………………..….4

1.3 Входная информация……………………………………………………......4

2. ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ……………………………....5

2.1 Структура программного обеспечения…………………………………….5

2.2 Пояснения по работе основных функций серверной части приложения..5

2.3 Пояснения по работе основных функций клиентской части

приложения…………………………………………………………………………….10

2.4 Требования к операционной системе……………………………………...15

3. ПЕРЕЧЕНЬ ВЫХОДНЫХ СИГНАЛОВ………………………………………......16

3.1 Перечень выходных данных………………………………………………..16

4. АУДИТ БЕЗОПАСНОСТИ………………………………………………………....17

4.1 Анализ серверной части приложения……………………………………...17

4.2 Анализ клиентской части приложения…………………………………….20

4.3 Реализация программы для восстановления пароля……………………...22

Лист регистрации изменений………………………………………………………….24

Приложение А………………………………………………………………………….25

**1. ОПИСАНИЕ ПОСТАНОВКИ ЗАДАЧ**

**1.1 Характеристики комплекса задач.**

1. Назначение комплекса задач

Клиент-серверное приложение предназначено для решения задач по обмену текстовыми сообщениями между пользователями в одной локальной сети.

1. Периодичность решения задач

Задача решается по запросу, то есть каждый раз, как только пользователь успешно проходит аутентификацию и подключается к серверу.

1. Условия, при которых прекращается решение задач

Если клиент не успешно прошел аутентификацию, и введенный лицензионный ключ не верен, то решение задачи приостанавливается.

**1.2 Выходная информация.**

Выходной информацией для клиентской части являются сообщения, принятые от сервера.

Выходной информацией для серверной части приложения являются сообщения от клиентов, отправляемые каждому пользователю.

**1.3 Входная информация.**

Входной информацией для клиентской части являются сообщения, вводимые пользователями.

Входной информацией для серверной части является ip и port, с которых впоследствии и будут приходить сообщения.

**2. ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**2.1 Структура программного обеспечения.**

Система состоит из двух частей: приложение клиента, которое устанавливается непосредственно на машине пользователя, и приложение сервера, находящееся на специально отведенной машине. С помощью клиентской части пользователи подключаются к серверу для дальнейшего обмена сообщениями с другими уже подключенными пользователями.

Программа защищена от статического и динамического анализа, лицензионным ключом и защитой от отладчика соответственно. Для запуска приложения необходимо знать лицензионный ключ от клиентской части и от серверной.

**2.2 Пояснения по работе основных функций серверной части приложения.**

После запуска программы сервера запрашивается ввод пароля от пользователя:

std::cin >> s;

Пользователь вводит пароль, который скрывается программой, и пользователь не видит вводимых им символов.

HANDLE hStdin = GetStdHandle(STD\_INPUT\_HANDLE);

DWORD mode = 0;

GetConsoleMode(hStdin, &mode);

SetConsoleMode(hStdin, mode & (~ENABLE\_ECHO\_INPUT));

…

SetConsoleMode(hStdin, mode);

Затем программа для каждого символа пароля вычисляет его хэш и преобразует обратно в символ, тем самым создавая зашифрованную строку

char out[8];

for (unsigned int i = 0; i < 8; i++)

out[i] = (char)(pass[i] + 120) % 125 + 12;

В коде программы хранится «эталонная» строка ("Мимесис") , которая считывается из файла("countersign.txt"), с которой и происходит сравнение хэша пароля.

ifstream in("countersign.txt");

string str;

std::getline(in, str);

in.close();

std::cout << "Enter password\n";

string s;

getline(cin, s);

if (encode(s) != str)

{

std::cout << "Error\n";

return 0;

}

Если сравнение вернет не ноль, то программа прекращает свою работу и закрывается:

После чего программа определяет время ввода пароля, если пароль вводится более 10 секунд, выводится сообщение ("Время истекло"), сравнение возвращает 0 и программа закрывается:

auto begin = std::chrono::steady\_clock::now();

…

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count()/1000 << " s\n";

if (elapsed\_ms.count()/1000 >= 10)

{

std::cout << "Время истекло";

return 0;

}

Если введенный пароль удовлетворяет необходимым условиям, то далее происходит создание самого сервера.

Первым этапом является инициализация библиотеки для создания сокетов:

std::cout << "Creating server:\n";

WSAData ws;

WORD version = MAKEWORD(2, 2);

int MasterSocket = WSAStartup(version, &ws);

if (MasterSocket != 0)

{

return 0;

}

Если инициализация библиотеки прошла успешно, тогда происходит создание сокета и присвоение сокету локального адреса и вызов функции для ожидания подключений к сокету. Для хранения сокетов используется вектор, состоящий из Client – класс, представляющий каждого, подключенного к сети и состоящего из номера и сокета

class Client

{

private:

int key;

SOCKET Client\_Sock;

public:

Client(SOCKET \* in, int C) :Client\_Sock(\*in) { key = C; }

Client() {}

SOCKET getsock() { return Client\_Sock; }

int getkey() { return key; }

};

SOCKET Connect;

Std::vector<Client> Connection(64);

SOCKET Listen;

struct addrinfo hints;

struct addrinfo \* result;

ZeroMemory(&hints,sizeof(hints));

hints.ai\_family=AF\_INET;

hints.ai\_flags=AI\_PASSIVE;

hints.ai\_socktype=SOCK\_STREAM;

hints.ai\_protocol=IPPROTO\_TCP;

Пользователь задает ip и port

std::cout << "Введите ip:\n";

std::string iport; std::cin >> iport;

std::cout << "Введите port:\n";

std::string port; std::cin >> port;

getaddrinfo(iport.c\_str(), port.c\_str(), &hints, &result);

Listen=socket(result->ai\_family,result->ai\_socktype,result->ai\_protocol);

bind(Listen,result->ai\_addr, result->ai\_addrlen);

listen(Listen,SOMAXCONN);

freeaddrinfo(result);

После создается сокет для клиента и поток для его обслуживания.

std::cout << "Start" << std::endl;

char c\_connect[] = "Connect";

while (1)

{

//проверка на получение сигнала от кого-нибудь

if (Connect = accept(Listen, NULL, NULL))

{

std::cout << c\_connect << ' ' << Count << std::endl;

Connection[Count] = Client::Client(&Connect, Count);

Count++;

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)SendM, (LPVOID)(Count - 1), NULL, NULL);

}

Функция SendM предназначена для обслуживания очередного подключившегося клиента, отправки ему сообщений, полученных от других пользователей:

void SendM(int ID)

{

for (;; Sleep(75))

{

SENDBUFFER s;

int iResult = recv(Connection[ID].getsock(), (char\*)&s, sizeof(s), NULL);

std::cout << s.name << ' ' << s.typemessage << ' ' << Connection[ID].getkey() << std::endl;

if (s.typemessage == 3)

{

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

send(Connection[i].getsock(), (char\*)&s, sizeof(s), NULL);

closesocket(Connection[ID].getsock());

decompres(ID);

Count--;

break;

}

if (iResult>0)

{

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

send(Connection[i].getsock(), (char\*)&s, sizeof(s), NULL);

}

}

std::cout << ID << " off\n";

}

Каждое сообщение, проходящее через данную сеть, представляет из себя структуру, состоящую из четырех полей – ключ, тип сообщения, имя пользователя и само сообщения.

struct SENDBUFFER

{

SENDBUFFER()

{

typemessage = 0;

key = 0;

ZeroMemory(name, sizeof(char) \* 14);

ZeroMemory(buffer, sizeof(char) \* 202);

}

int key;

int typemessage;

char name[14];

char buffer[202];

};

Суть работы данной функции – она отслеживает приходящие от клиента сообщение и в зависимости от типа сообщения реагирует на них. Если тип сообщение «3» - это значит, что клиент хочет отключиться от сети, и, следовательно, его надо отключить, то есть удалить из вектора, тем самым освободив место для новых пользователей. В других же случаях надо разослать сообщение всем остальным клиентам.

Для удаления пользователя из сети существует функция decompress

void decompres(int n)

{

for (int i = n; i < Count - 1; i++)

Connection[i] = Connection[i + 1];

Connection[Count].~Client();

}

**2.3 Пояснение по работе основных функций клиентской части приложения.**

Принцип работы клиента в целом схож с работой сервера. Проверка лицензионного ключа отличается только самой эталонной строкой для сравнения "fdsssdf" которая делится на 2 части и после складывается, также присутствует защита от отладчика.

if (IsDebuggerPresent()) ExitProcess(0);

BOOL f=false;

CheckRemoteDebuggerPresent(GetCurrentProcess(), &f);

if (f==true) ExitProcess(0);

Полученная строка проходит через функцию кодировки и сравнивается с эталонной строкой

CString encode(CString s)

{

std::string out;

for (unsigned int i = 0;i < s.GetLength();i++)

out += (char)(s[i] + 120) % 125 + 12;

return CString(out.c\_str());

}

Сравнение с эталонной строкой:

CString S = "fds";

CString N = "ssdf";

if (encode(password) != S+N)

{

AfxMessageBox(\_T("Error password"));

return;

}

Далее начинается непосредственное создание клиента, создание самого сокета, который является классом, порожденным от класса CAsyncSocket – класс асинхронных сокетов библиотек MFC

class CSock : public CAsyncSocket

{

public:

CSock();

virtual ~CSock();

// События для возможности обработки

// сообщений в асинхронном режиме.

virtual void OnConnect(int nErrorCode);

virtual void OnClose(int nErrorCode);

virtual void OnReceive(int nErrorCode);

// Переменная для "связи" с родительским окном.

CWnd\* m\_pParent;

};

// Класс CAsyncSocket упрощает процедуру создания сокета,

// вкладывая в функцию Create() непосредственное создание

// сокета и связывание его с одним из IP адресом доступном на компьютере.

if(m\_mainSocket.Create() == TRUE)

{

CString strAddress;

m\_wndIPAddress.GetWindowText(strAddress);

CString strPort;

m\_wndPort.GetWindowText(strPort);

if(m\_mainSocket.Connect(strAddress, atoi(strPort)) == FALSE)

{

// В ассинхронном режиме код этой ошибки

// считается как ожидание события подключения,

// т.е. практически успешный возврат.

if(GetLastError() == WSAEWOULDBLOCK)

{

DisabledControl(false);

}

else

{

// Если какая-либо ошибка возникла,

// приводим приложение в первоначальное состояние,

// готовым к следующей попытке создания соединения.

StopChat();

}

}

}

Перегружаем встроенные в родительский класс процедуры подключения, закрытия и получения.

// Событие подключения на стороне клиентского приложения.

void CSock::OnConnect(int nErrorCode)

{

CChatCppDlg\* pDlg = (CChatCppDlg\*)m\_pParent;

nErrorCode == 0 ? pDlg->OnConnect(FALSE) : pDlg->OnConnect(TRUE);

CAsyncSocket::OnConnect(nErrorCode);

}

// Событие отключения от сети

void CSock::OnClose(int nErrorCode)

{

}

// Событие возможности получения сообщений.

void CSock::OnReceive(int nErrorCode)

{

CChatCppDlg\* pDlg = (CChatCppDlg\*)m\_pParent;

pDlg->OnReceive();

CAsyncSocket::OnReceive(nErrorCode);

}

// Событие подключения, происходит на стороне клиента.

void CChatCppDlg::OnConnect(BOOL Error)

{

if(Error == TRUE)

{

AfxMessageBox("Попытка подключения была отвергнута!\nВозможно сервер еще не создан!");

StopChat();

}

else

{

m\_ButtonSend.EnableWindow(TRUE);

SetWindowText("Сеть работает!");

TCHAR key\_out[1];

ZeroMemory(key\_out, sizeof(TCHAR));

m\_mainSocket.Receive(&key\_out, sizeof(key\_out));

CString str = key\_out;

key = atoi(str);

}

}

Сама функция принятия сообщений, которая выводит информацию в чат в зависимости от типа сообщения:

* «3» - пользователь покинул чат
* Иначе выводится сообщение

void CChatCppDlg::OnReceive(void)

{

SENDBUFFER sb;

if(m\_ButtonStartClient.GetCheck() == BST\_CHECKED)

{

m\_mainSocket.Receive(&sb, sizeof(SENDBUFFER));

}

// Обработка принятого сообщения на основе

// его типа.

switch(sb.typemessage)

{

case m\_TypeMessage::tmChat:

{

CString strChat;

m\_wndChat.GetWindowText(strChat);

strChat += " " + CString(sb.name) + ": " + CString(sb.buffer) + "\r\n";

m\_wndChat.SetWindowText(strChat);

int number\_line = m\_wndChat.GetLineCount();

m\_wndChat.LineScroll(number\_line);

}

break;

case m\_TypeMessage::tmDisconnect:

{

CString strChat;

m\_wndChat.GetWindowText(strChat);

// Если принято сообщение об остановки чата(отключении сервера),

// изменим содержимое сообщения.

strChat += " " + CString(sb.name) + " - покинул(а) чат!" + "\r\n";

m\_wndChat.SetWindowText(strChat);

int number\_line = m\_wndChat.GetLineCount();

m\_wndChat.LineScroll(number\_line);

}

break;

}

}

Отправка сообщений осуществляется при нажатии на кнопку, пустое сообщение отправить нельзя:

void CChatCppDlg::SendChat(CString strMessage)

{

SENDBUFFER sb;

int len = strMessage.GetLength();

//strMessage = encod(std::string((LPCTSTR)strMessage), len);

memcpy(sb.buffer, strMessage.GetBuffer(), sizeof(TCHAR)\*len);

m\_wndName.GetWindowText(strMessage);

len = strMessage.GetLength();

memcpy(sb.name, strMessage.GetBuffer(), sizeof(TCHAR)\*len);

sb.typemessage = m\_TypeMessage::tmChat;

SendBuffer(sb, true);

}

// Послать буфер подготовленного сообщения в сеть.

void CChatCppDlg::SendBuffer(SENDBUFFER sb, bool toserver)

{

// Если сокет не создан, нечего делать в этой функции.

if(m\_mainSocket.m\_hSocket == INVALID\_SOCKET) return;

if(m\_ButtonStartClient.GetCheck() == BST\_CHECKED)

{

int send = m\_mainSocket.Send(&sb, sizeof(SENDBUFFER));

if(send == sizeof(SENDBUFFER))

m\_wndSend.SetWindowText("");

}

}

Также стоит упомянуть процедуру создания сообщения об отключении

void CChatCppDlg::SendDisconnect(void)

{

SENDBUFFER sb;

CString s;

m\_wndName.GetWindowText(s);

int len = s.GetLength();

memcpy(sb.name, s.GetBuffer(), sizeof(TCHAR)\*len);

sb.typemessage = m\_TypeMessage::tmDisconnect;

SendBuffer(sb, true);

}

Графический интерфейс приложения клиента реализован на основе приложений MFC:

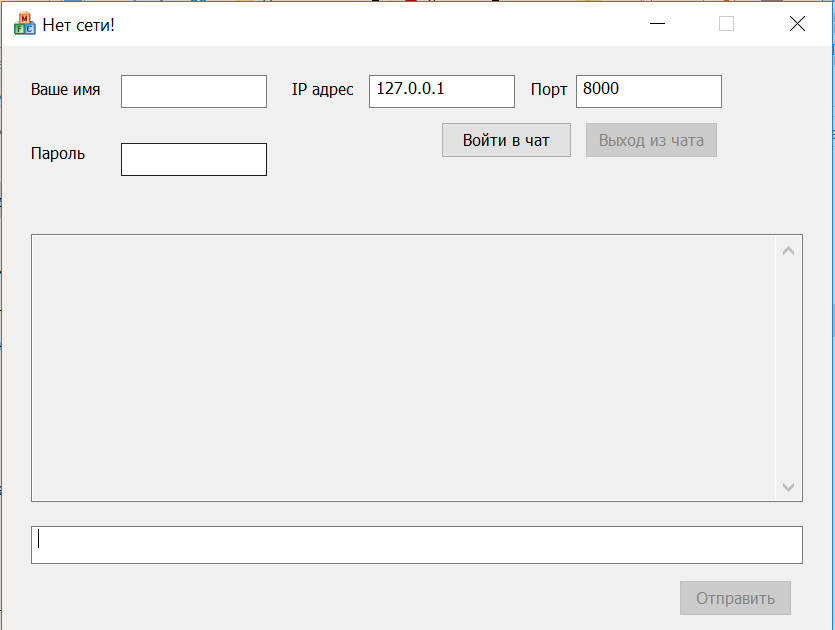


Рисунок. 1 – Реализация графической клиентской части

**2.4 Требования к операционной системе.**

Для выполнения программы необходима операционная система Windows 7 x64/86 или новее.

**3.** **ПЕРЕЧЕНЬ ВЫХОДНЫХ СИГНАЛОВ**

**3.1 Перечень выходных данных**

Выходными данными для клиентской части приложения являются сообщения, принятые с сервера.

Выходными данными для серверной части приложения являются сообщения клиентов, отправляемые каждому пользователю.

**4. АУДИТ БЕЗОПАСНОСТИ**

В данной работе передо мной поставлена задача о проведении аудита безопасности клиент-серверного приложения, проведение динамического и статического анализа, а также необходимость осуществления взлома приложения.

В качестве программы для проведения аудита была взята работа студента группы ИБ-116 Шахиной А.В.

Программный продукт (сервер/клиент) дизассемблировался, отлаживался и редактировался с помощью программы IDA Pro 7.0.

**4.1 Анализ серверной части приложения.**

В начале рассмотрим список функций, используемых в программе. При изучении списка вызываемых функций (Рисунок 1) можно заметить функцию защиты от отладчика: IsDebuggerPresent, т.е. при попытке отладить программу, она закрывается и выдает ошибку. Таким образом, можно сделать очевидный вывод, что реализована защита от динамического анализа.

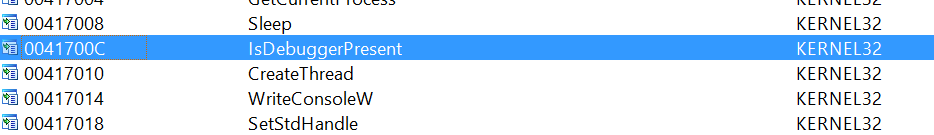


Рисунок. 1 - Список функций

Далее производим поиск переменных «pass» «hash», в которых могут храниться нужные нам данные. Поиск не дал результатов.

Но нам удается найти функцию проверки пароля, представляющую просто ветвление через if.

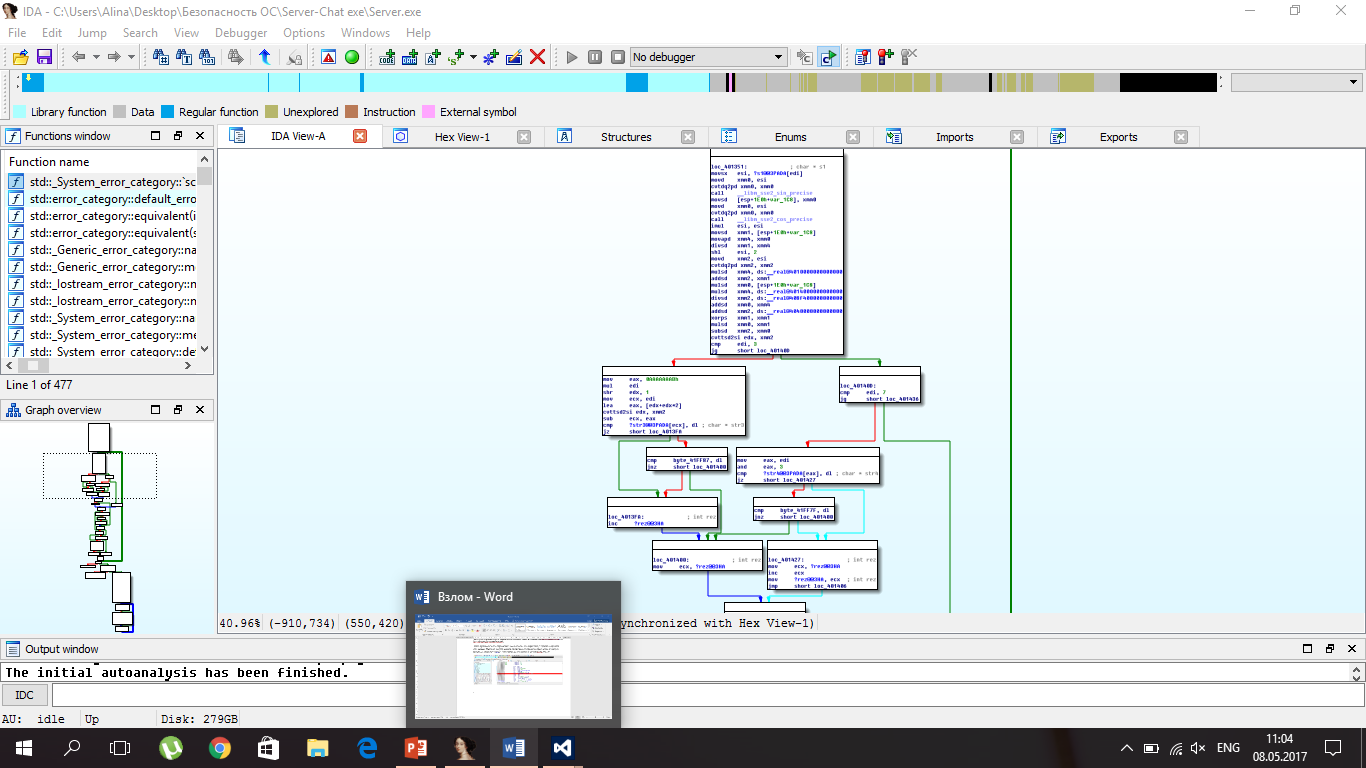


Рисунок. 2 – Функция проверки пароля

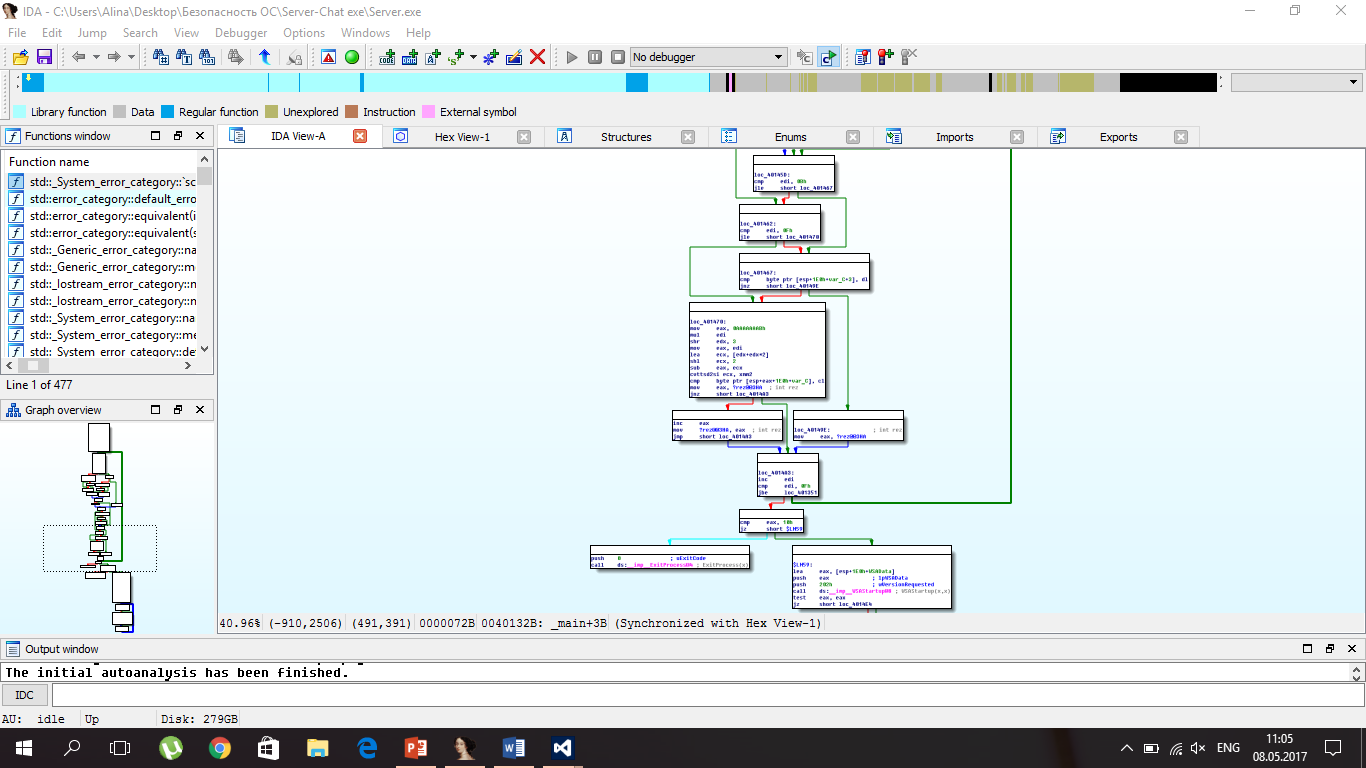


Рисунок. 3 – Функция проверки пароля

Можно сделать вывод, что проверка пароля осуществляется следующим образом:

В коде программы хранится какая-то эталонная строка, с которой сравнивается хэш пароля, высчитанный определенной формуле.

Проанализировав функцию, можно сделать вывод, что пароль делится на 4 части, первая часть сравнивается с str3, вторая - str4, третья - str2, четвертая - str1. Длина пароля равна 16.

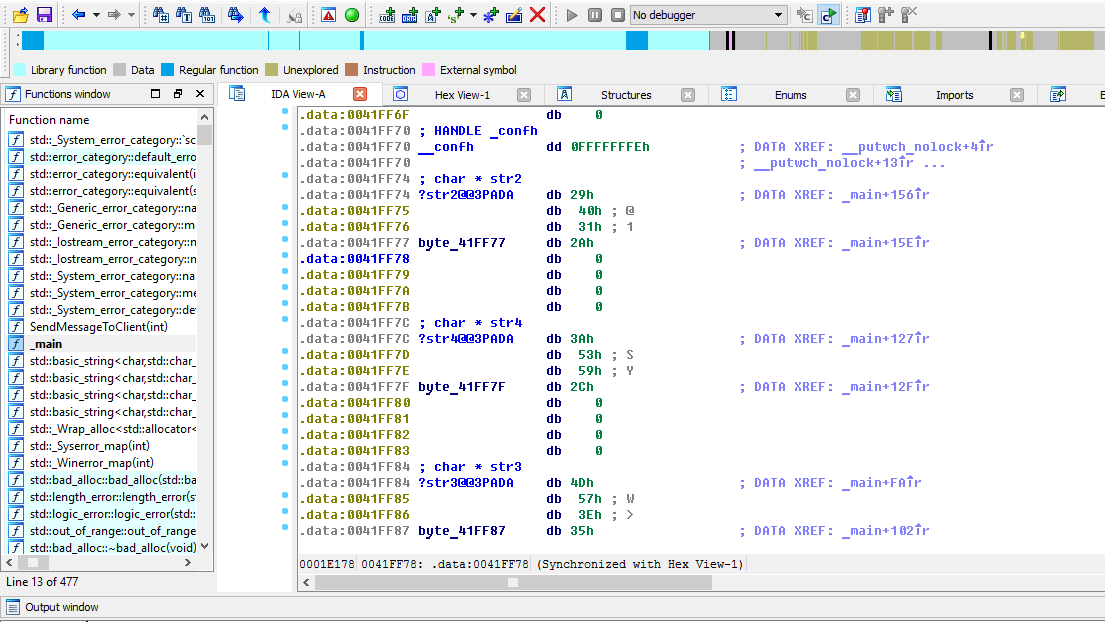


Рисунок. 4 – Функция проверки пароля

Так же мной была обнаружена переменная, в которой хранилось значение “+H<K”, что вполне может являться частью хэша пароля.

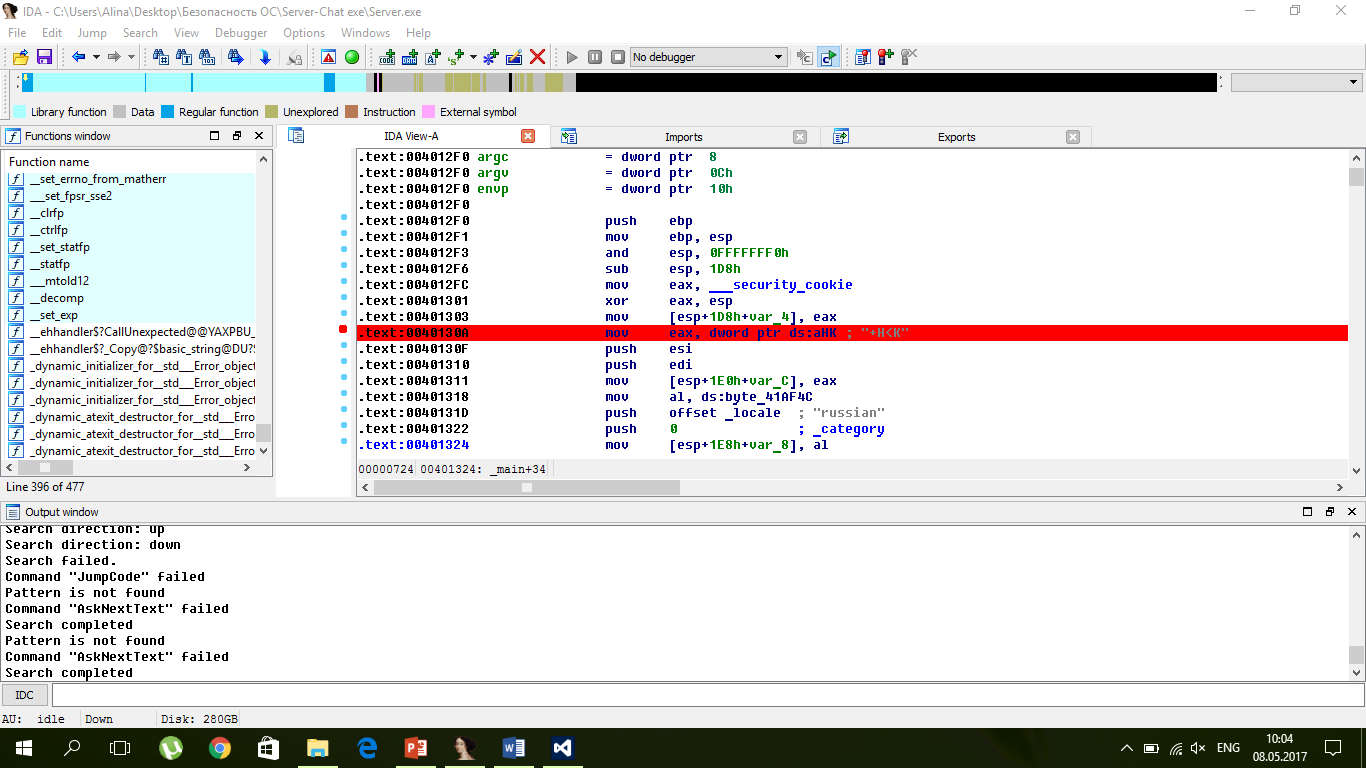


Рисунок. 5 – Переменная с частью хеша пароля

Таким образом, эталонная строка «MW>5:SY,)@1\*+H<K».

Посмотрим еще раз на функцию проверки соответствия пароля.

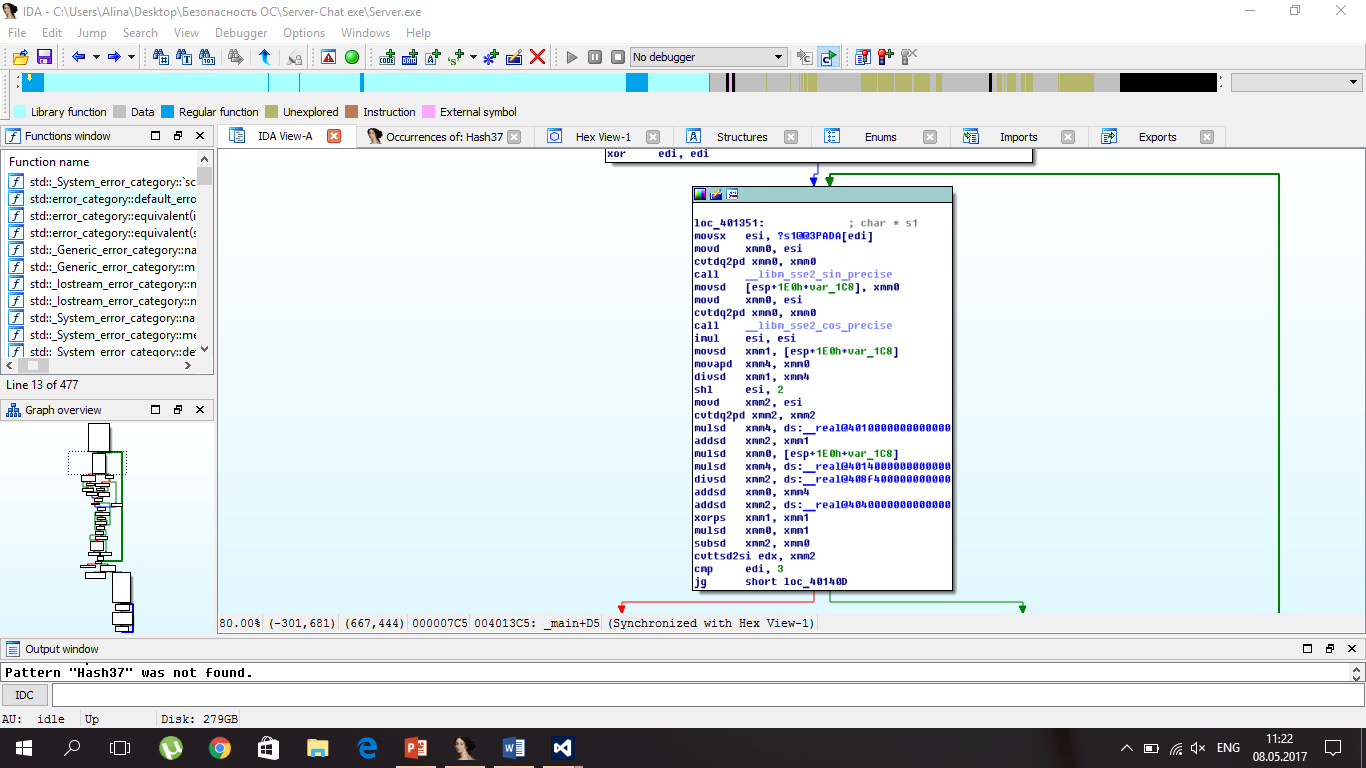


Рисунок. 6 – Функция проверки соотверствия пароля

Проведя анализ функции, удалось восстановить формулу, по которой высчитывается хэш.

(sin(x)/cos(x)+4\*x\*x)/1000+32-(sin(x)\*cos(x)+4\*cos(x)\*5)\*0 (1)

Заметим, что последний многочлен умножается на 0, следовательно, формулу можно упростить

(sin(x)/cos(x)+4\*x\*x)/1000+32 (2)

Из вышенаписанного можно сделать вывод, что пользователь вводит пароль, от каждого символа пароля вычисляется его код. Далее код изменяется с помощью формулы(2). После чего, код преобразуется в новый символ, формирующий хэш пароля, который сравнивается с эталонной строкой.

**4.2 Анализ клиентской части.**

При просмотре и анализе всех функций удалось выявить такие функции, как IsDebuggerPresent() и CheckRemoteDebuggerPresent(). Так же были найдены функции Hash37 и toHash37, проанализировав которые, мне удалось выяснить, что функция Hash37 получает ASCII-код символа, а функция toHash37 обратная ей.

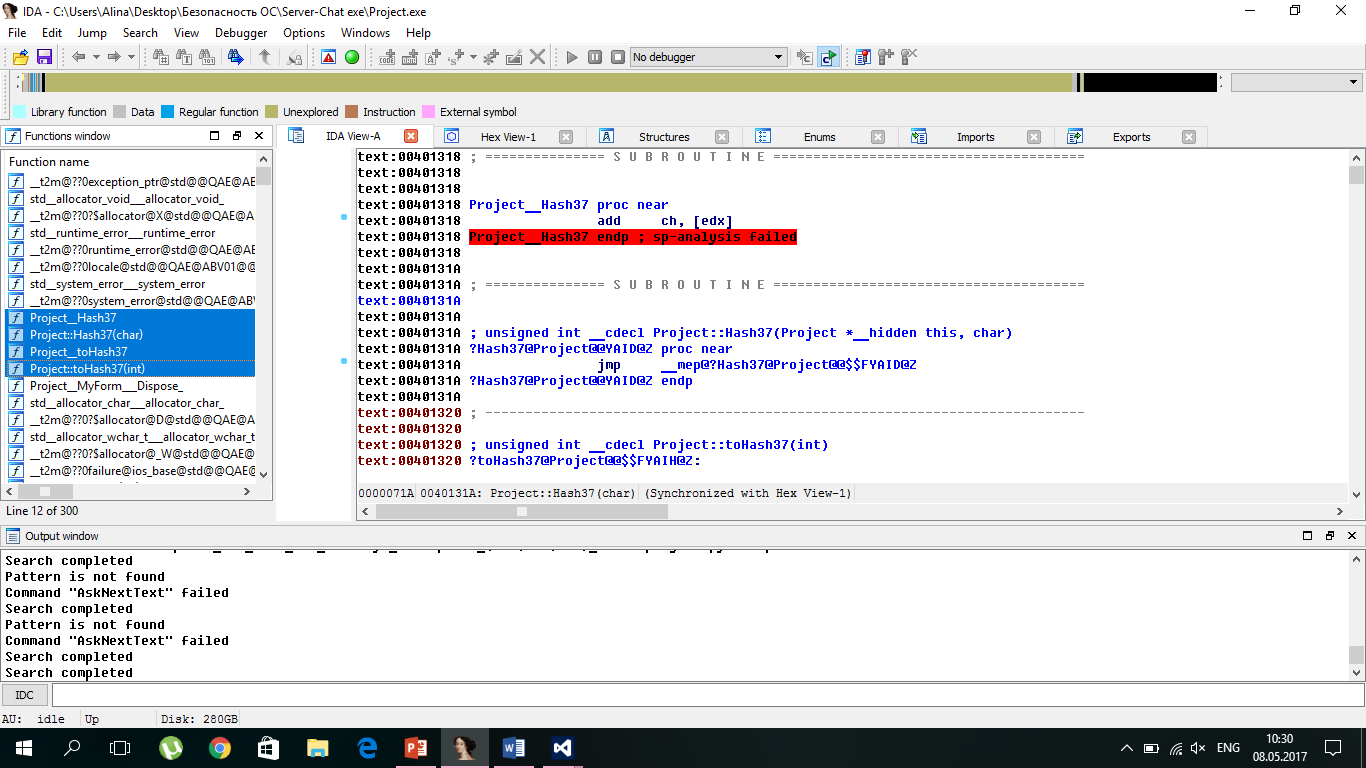


Рисунок. 7 – Наличие функций Hash37 и toHash37

Далее произвела поиск переменных str1, str2, str3, str4, в них, как и в приложении сервера, хранятся значения похожие на части хэша.

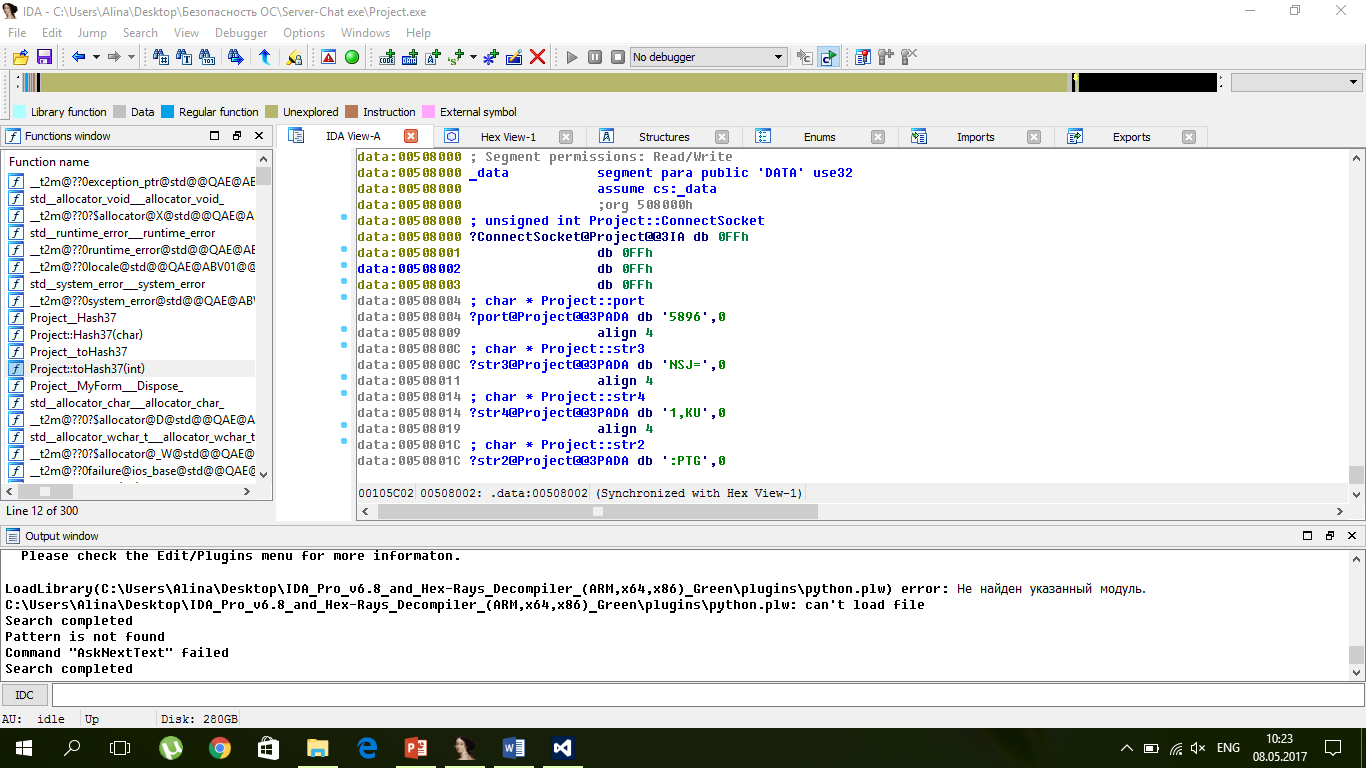


Рисунок. 8 – Поиск переменных

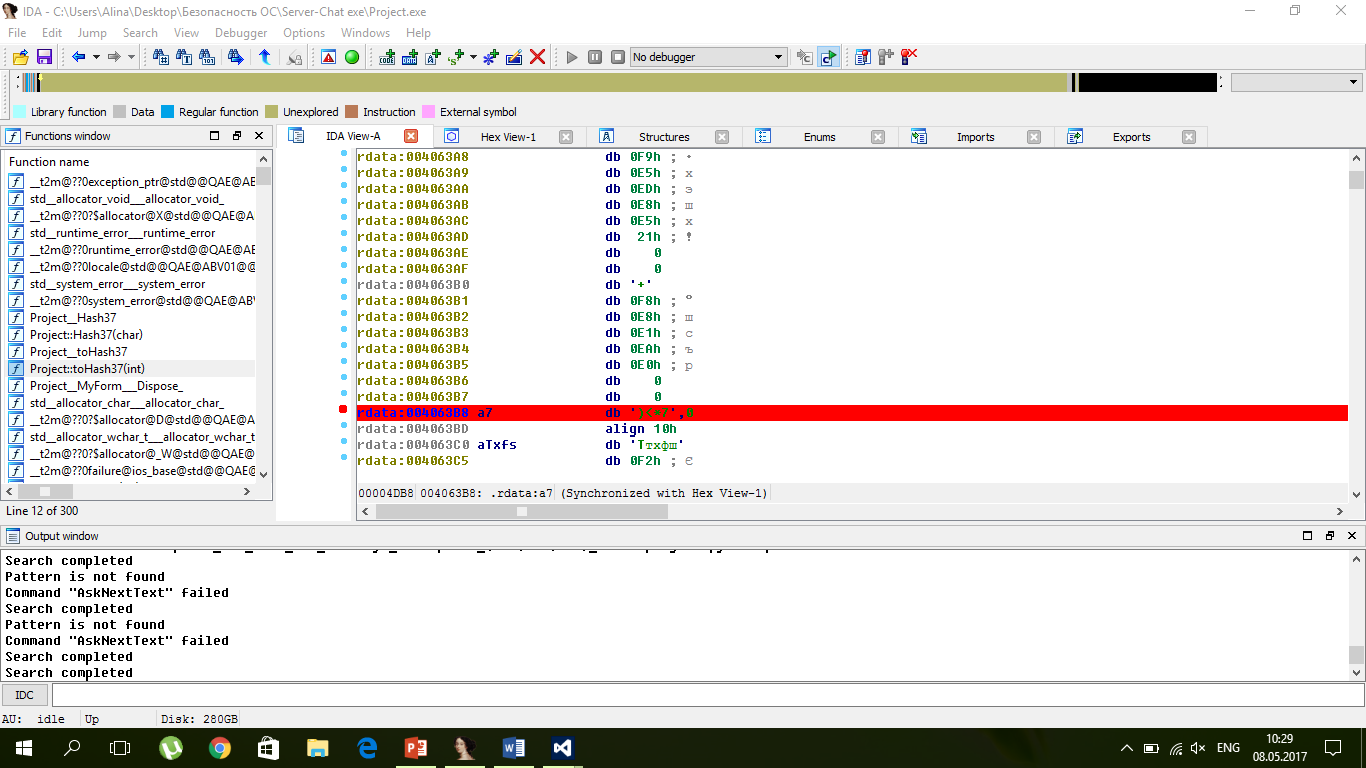


Рисунок. 9 - Переменная с частью хеша пароля

Мной была обнаружена функция function, которая содержит формулу вычисления хэша.

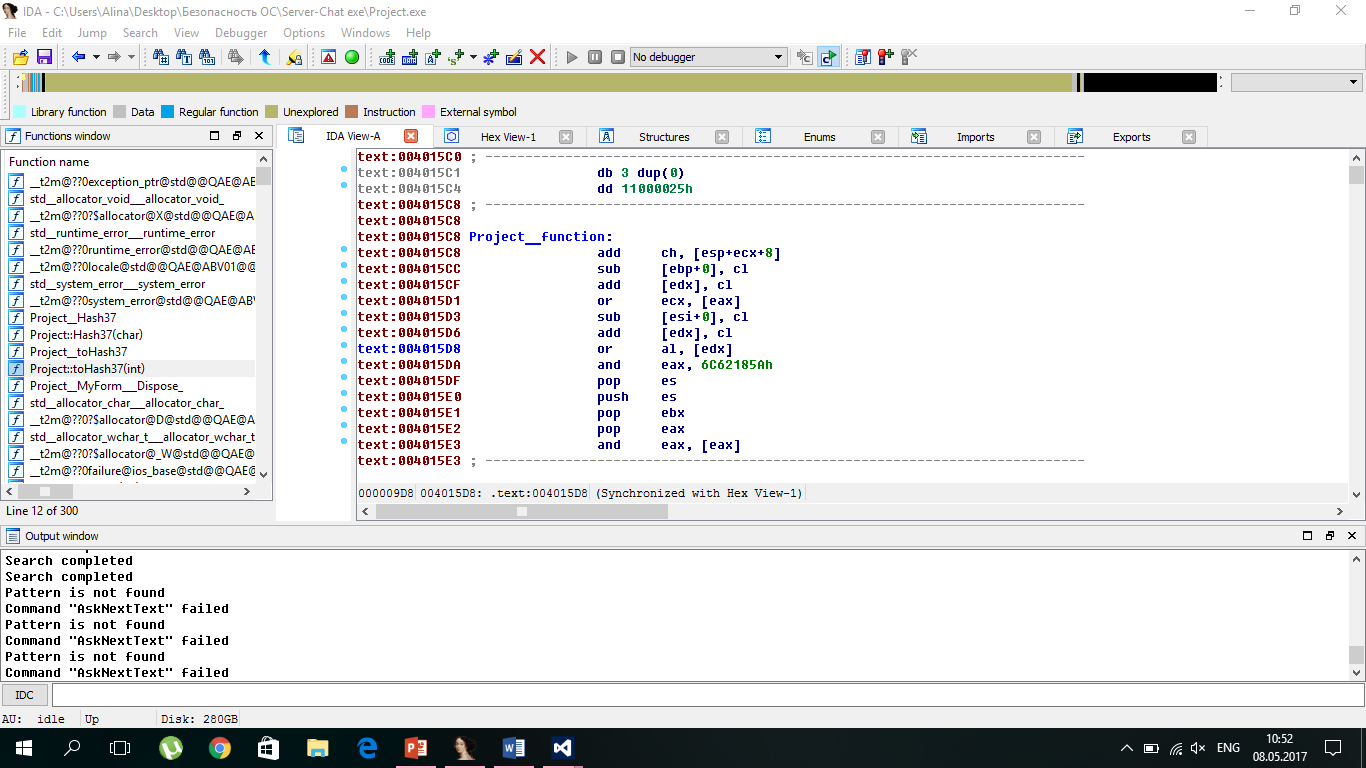


Рисунок. 10 - Функция хэширующая пароль

Для приложения клиента эталонная строка равна «NSJ=1,KU:PTG)<\*7»

Вероятнее всего, хэш пароля в приложении клиента вычисляется также, как хэш пароля в приложении сервера.

**4.3 Реализация программы для восстановления пароля.**

Реализую программу на языке программирования С++, которая сможет восстановить исходный пароль

#include<iostream>

#include <cmath>

using namespace std;

int main()

{

char hash[] ="MW>5:SY,)@1\*+H<K";

char pass[16];

for (int i = 0; i < sizeof(hash); i++)

{

int p=char(hash[i]);

int q=0;

int flag=0;

while (flag!=1)

{

int m=(sin(q)/cos(q)+4\*q\*q)/1000+32;

if (m==p)

{

pass[i]=char(q);

flag=1;

}

else

q=q+1;

}

printf("%c",pass[i]);

}

}

Результатом работы написанной программы оказались значения:

kvWIRrx71ZC35eUh для сервера

lqgVB7htRnsd0T3M для клиента

Результат работы:

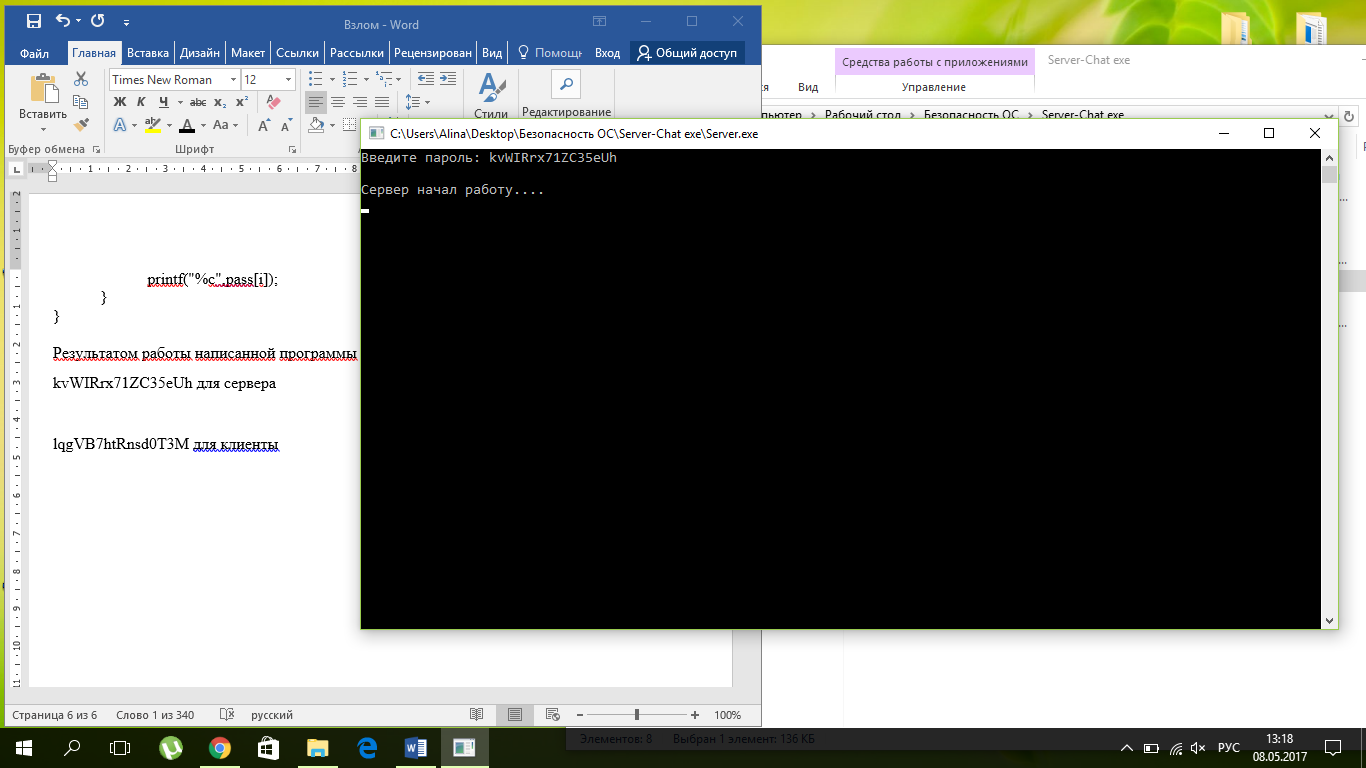


Рисунок. 11 – Результат работы сервера

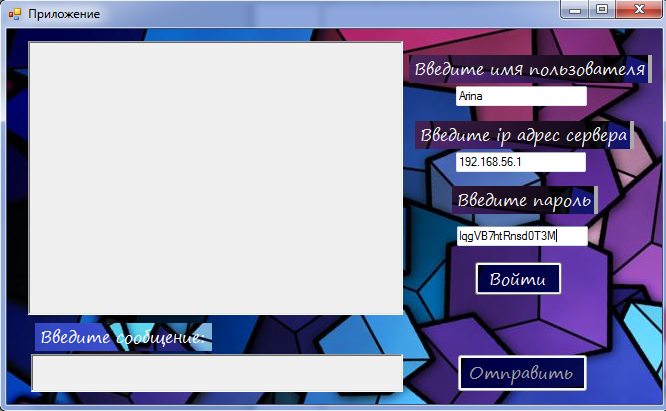


Рисунок. 12 – Результат работы клиента

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего  листов  (страниц)  в докум | №  документа | Входящий  № сопрово  дительного  документа  и дата | Подп. | Дата |
| Изм | изменен  ных | заме  ненных | новых | Анулиро  ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

**ПРИЛОЖЕНИЕ А**

Текст программы на исходном языке.

**Клиентский модуль «Client.exe»:**

**ChatCpp.cpp:**

// ChatCppDlg.cpp : implementation file

//

#include "stdafx.h"

#include "ChatCppDlg.h"

#include ".\chatcppdlg.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CAboutDlg dialog used for App About

bool access = true;

CChatCppDlg::CChatCppDlg(CWnd\* pParent /\*=NULL\*/)

: CDialog(CChatCppDlg::IDD, pParent)

, password(\_T(""))

{

m\_hIcon = AfxGetApp()->LoadIcon(IDR\_MAINFRAME);

}

void CChatCppDlg::DoDataExchange(CDataExchange\* pDX)

{

CDialog::DoDataExchange(pDX);

DDX\_Control(pDX, IDC\_RADIO\_CLIENT, m\_ButtonStartClient);

DDX\_Control(pDX, IDC\_EDIT\_NAME, m\_wndName);

DDX\_Control(pDX, IDC\_EDIT\_IP, m\_wndIPAddress);

DDX\_Control(pDX, IDC\_EDIT\_PORT, m\_wndPort);

DDX\_Control(pDX, IDC\_EDIT\_SEND, m\_wndSend);

DDX\_Control(pDX, IDC\_BUTTON\_SEND, m\_ButtonSend);

DDX\_Control(pDX, IDC\_BUTTON\_STOPCHAT, m\_ButtonStopChat);

DDX\_Control(pDX, IDC\_EDIT\_CHAT, m\_wndChat);

DDX\_Text(pDX, IDC\_EDIT1, password);

}

BEGIN\_MESSAGE\_MAP(CChatCppDlg, CDialog)

ON\_WM\_SYSCOMMAND()

ON\_WM\_PAINT()

ON\_WM\_QUERYDRAGICON()

ON\_BN\_CLICKED(IDC\_RADIO\_CLIENT, OnBnClickedRadioClient)

ON\_BN\_CLICKED(IDC\_BUTTON\_STOPCHAT, OnBnClickedButtonStopchat)

ON\_BN\_CLICKED(IDC\_BUTTON\_SEND, OnBnClickedButtonSend)

ON\_WM\_CLOSE()

END\_MESSAGE\_MAP()

int key = 0;

BOOL CChatCppDlg::OnInitDialog()

{

CDialog::OnInitDialog();

// Add "About..." menu item to system menu.

// IDM\_ABOUTBOX must be in the system command range.

ASSERT((IDM\_ABOUTBOX & 0xFFF0) == IDM\_ABOUTBOX);

ASSERT(IDM\_ABOUTBOX < 0xF000);

CMenu\* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

CString strAboutMenu;

strAboutMenu.LoadString(IDS\_ABOUTBOX);

if (!strAboutMenu.IsEmpty())

{

pSysMenu->AppendMenu(MF\_SEPARATOR);

pSysMenu->AppendMenu(MF\_STRING, IDM\_ABOUTBOX, strAboutMenu);

}

}

// Set the icon for this dialog. The framework does this automatically

// when the application's main window is not a dialog

SetIcon(m\_hIcon, TRUE); // Set big icon

SetIcon(m\_hIcon, FALSE); // Set small icon

//////////////////////////////////////////////////////////////////////////////

// Необходимая инициализация

SetWindowText("Нет сети!");

m\_wndName.SetLimitText(12); // Ограничение количества вводимых символов.

m\_wndSend.SetLimitText(200); // Ограничение количества вводимых символов.

m\_wndIPAddress.SetWindowText("127.0.0.1"); // Адрес по умолчанию для тестирования.

m\_wndPort.SetWindowText("8000");

m\_ButtonStopChat.SetWindowText(g\_strExitFromChat);

m\_mainSocket.m\_pParent = this;

return TRUE; // return TRUE unless you set the focus to a control

}

void CChatCppDlg::OnPaint()

{

if (IsIconic())

{

CPaintDC dc(this); // device context for painting

SendMessage(WM\_ICONERASEBKGND, reinterpret\_cast<WPARAM>(dc.GetSafeHdc()), 0);

// Center icon in client rectangle

int cxIcon = GetSystemMetrics(SM\_CXICON);

int cyIcon = GetSystemMetrics(SM\_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

// Draw the icon

dc.DrawIcon(x, y, m\_hIcon);

}

else

{

CDialog::OnPaint();

}

}

CString encode(CString s)

{

std::string s1((LPCSTR)s);

std::string out;

for (unsigned int i = 0;i < s1.length();i++)

out += (char)(s[i] + 120) % 125 + 12;

return CString(out.c\_str());

}

// Запуск клиента

void CChatCppDlg::OnBnClickedRadioClient()

{

UpdateData(true);

CString S = "fds";

CString N = "ssdf";

if (encode(password) != S+N)

{

AfxMessageBox(\_T("Error password"));

return;

}

// Контроль несанкционированного запуска клиента:

// Если кнопка не в состоянии нажатой,

// если сокет в работе (т.е. только с нулевым сокетом можно начинать работать),

// очень неудобно если в чате все будут под одинаковыми именами..

if(m\_ButtonStartClient.GetCheck() != BST\_CHECKED) return;

if(m\_mainSocket.m\_hSocket != INVALID\_SOCKET) return;

if(QueryName() == false)

{

AfxMessageBox("Введите свое имя для чата!");

StopChat();

return;

}

// Класс CAsyncSocket упрощает процедуру создания сокета,

// вкладывая в функцию Create() непосредственное создание

// сокета и связывание его с одним из IP адресом доступном на компьютере.

if(m\_mainSocket.Create() == TRUE)

{

CString strAddress;

m\_wndIPAddress.GetWindowText(strAddress);

CString strPort;

m\_wndPort.GetWindowText(strPort);

if(m\_mainSocket.Connect(strAddress, atoi(strPort)) == FALSE)

{

// В ассинхронном режиме код этой ошибки

// считается как ожидание события подключения,

// т.е. практически успешный возврат.

if(GetLastError() == WSAEWOULDBLOCK)

{

DisabledControl(false);

}

else

{

// Если какая-либо ошибка возникла,

// приводим приложение в первоначальное состояние,

// готовым к следующей попытке создания соединения.

StopChat();

}

}

}

}

// Нажали кнопку "Выйти из чата".

void CChatCppDlg::OnBnClickedButtonStopchat()

{

StopChat();

}

// Запрещает доступ к управлениям при работе

// приложения в режиме сервера или клиента.

// Цель запрета - избежать исключения от

// случайного нажатия "неправильных" кнопок.

void CChatCppDlg::DisabledControl(bool server)

{

// Запреты.

m\_wndIPAddress.EnableWindow(FALSE);

m\_wndPort.EnableWindow(FALSE);

m\_ButtonSend.EnableWindow(FALSE);

if(server == true)

{

m\_ButtonStopChat.SetWindowText(g\_strStopChat);

m\_ButtonStartClient.EnableWindow(TRUE);

}

else

{

m\_ButtonStopChat.SetWindowText(g\_strExitFromChat);

//m\_ButtonStartServer.EnableWindow(FALSE);

}

// Разрешения.

// Разрешить возможность выхода из чата.

m\_ButtonStopChat.EnableWindow(TRUE);

}

// Разрешить доступ к управлениям после закрытия сокетов.

// Цель запрета - избежать исключения от

// случайного нажатия "неправильных" кнопок.

void CChatCppDlg::EnabledControl(void)

{

// Разрешения.

m\_wndIPAddress.EnableWindow(TRUE);

m\_wndPort.EnableWindow(TRUE);

m\_ButtonStartClient.EnableWindow(TRUE);

//m\_ButtonStartServer.EnableWindow(TRUE);

// Запреты.

m\_ButtonStopChat.EnableWindow(FALSE);

m\_ButtonSend.EnableWindow(FALSE);

}

// Выход из чата,

// если это сработало на стороне сервера,

// то это полная остановка чата.

// Для более надежной работы чата, во всех

// приложениях должны быть запущеня дублирующие серверы...

void CChatCppDlg::StopChat(void)

{

// Отсылаем сигнал об отключении от чата.

SendDisconnect();

m\_mainSocket.Close();

// Очистим вектор от ненужных элементов.

m\_vecSockets.clear();

//m\_ButtonStartServer.SetCheck(BST\_UNCHECKED);

m\_ButtonStartClient.SetCheck(BST\_UNCHECKED);

// Разрешим доступ к управлению для

// повторных попыток.

EnabledControl();

// В чате нет никого.

SetWindowText("Нет сети!");

}

// Отправка подготовленного сообщения.

void CChatCppDlg::OnBnClickedButtonSend()

{

CString strChat;

m\_wndSend.GetWindowText(strChat);

SendChat(strChat);

}

// Извлечение сообщений из сети чата.

void CChatCppDlg::OnReceive(void)

{

SENDBUFFER sb;

if(m\_ButtonStartClient.GetCheck() == BST\_CHECKED)

{

m\_mainSocket.Receive(&sb, sizeof(SENDBUFFER));

}

// Обработка принятого сообщения на основе

// его типа.

switch(sb.typemessage)

{

case m\_TypeMessage::tmChat:

{

CString strChat;

m\_wndChat.GetWindowText(strChat);

strChat += " " + CString(sb.name) + ": " + CString(sb.buffer) + "\r\n";

m\_wndChat.SetWindowText(strChat);

int number\_line = m\_wndChat.GetLineCount();

m\_wndChat.LineScroll(number\_line);

}

break;

case m\_TypeMessage::tmDisconnect:

{

CString strChat;

m\_wndChat.GetWindowText(strChat);

// Если принято сообщение об остановки чата(отключении сервера),

// изменим содержимое сообщения.

strChat += " " + CString(sb.name) + " - покинул(а) чат!" + "\r\n";

m\_wndChat.SetWindowText(strChat);

int number\_line = m\_wndChat.GetLineCount();

m\_wndChat.LineScroll(number\_line);

}

break;

}

}

// При закрытии приложения отправим в чат

// информацию об отключении чатующего.

void CChatCppDlg::OnClose()

{

StopChat();

CDialog::OnClose();

}

// Послать строку-сообщение в чат.

void CChatCppDlg::SendChat(CString strMessage)

{

SENDBUFFER sb;

int len = strMessage.GetLength();

//strMessage = encod(std::string((LPCTSTR)strMessage), len);

memcpy(sb.buffer, strMessage.GetBuffer(), sizeof(TCHAR)\*len);

m\_wndName.GetWindowText(strMessage);

len = strMessage.GetLength();

memcpy(sb.name, strMessage.GetBuffer(), sizeof(TCHAR)\*len);

sb.typemessage = m\_TypeMessage::tmChat;

SendBuffer(sb, true);

}

// Послать буфер подготовленного сообщения в сеть.

void CChatCppDlg::SendBuffer(SENDBUFFER sb, bool toserver)

{

// Если сокет не создан, нечего делать в этой функции.

if(m\_mainSocket.m\_hSocket == INVALID\_SOCKET) return;

if(m\_ButtonStartClient.GetCheck() == BST\_CHECKED)

{

int send = m\_mainSocket.Send(&sb, sizeof(SENDBUFFER));

if(send == sizeof(SENDBUFFER))

m\_wndSend.SetWindowText("");

}

}

// Формируем и отправляем сообщение об отключении от сети.

void CChatCppDlg::SendDisconnect(void)

{

SENDBUFFER sb;

CString s;

m\_wndName.GetWindowText(s);

int len = s.GetLength();

memcpy(sb.name, s.GetBuffer(), sizeof(TCHAR)\*len);

sb.typemessage = m\_TypeMessage::tmDisconnect;

SendBuffer(sb, true);

}

// Событие подключения, происходит на стороне клиента.

void CChatCppDlg::OnConnect(BOOL Error)

{

if(Error == TRUE)

{

AfxMessageBox("Попытка подключения была отвергнута!\nВозможно сервер еще не создан!");

StopChat();

}

else

{

m\_ButtonSend.EnableWindow(TRUE);

SetWindowText("Сеть работает!");

TCHAR key\_out[1];

ZeroMemory(key\_out, sizeof(TCHAR));

m\_mainSocket.Receive(&key\_out, sizeof(key\_out));

CString str = key\_out;

key = atoi(str);

}

}

// Запрос имени чатующего перед созданием сокета.

bool CChatCppDlg::QueryName(void)

{

CString strName;

m\_wndName.GetWindowText(strName);

if(strName == g\_EmptyName || strName.IsEmpty() == true)

return false;

return true;

}

**Серверный модуль «Server.exe»:**

**Source.cpp:**

// ConsoleApplication12.cpp: определяет точку входа для консольного приложения.

#pragma comment(lib,"Ws2\_32.lib")

#include <sys/types.h>

#include <WinSock2.h>

#include <Ws2tcpip.h>

#include <ctime>

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <chrono>

#include <windows.h>

#define STRICT 1

using namespace std;

class Client

{

private:

int key;

SOCKET Client\_Sock;

public:

Client(SOCKET \* in, int C) :Client\_Sock(\*in) { key = C; }

Client() {}

SOCKET getsock() { return Client\_Sock; }

int getkey() { return key; }

};

struct SENDBUFFER

{

SENDBUFFER()

{

typemessage = 0;

key = 0;

ZeroMemory(name, sizeof(char) \* 14);

ZeroMemory(buffer, sizeof(char) \* 202);

}

int key;

int typemessage;

char name[14];

char buffer[202];

};

//Сокет для подключения, для хранения подключенных и для "прослушка" + подсчет количества подключенных

SOCKET Connect;

SOCKET Listen;

int Count = 0;

std::vector<Client> Connection(64);

//Функция для отправки-приема сообщений... Принимает сообщение - рассылает всем подключенным

void decompres(int n)

{

for (int i = n; i < Count - 1; i++)

Connection[i] = Connection[i + 1];

Connection[Count].~Client();

}

void SendM(int ID)

{

for (;; Sleep(75))

{

SENDBUFFER s;

int iResult = recv(Connection[ID].getsock(), (char\*)&s, sizeof(s), NULL);

std::cout << s.name << ' ' << s.typemessage << ' ' << Connection[ID].getkey() << std::endl;

if (s.typemessage == 3)

{

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

send(Connection[i].getsock(), (char\*)&s, sizeof(s), NULL);

closesocket(Connection[ID].getsock());

decompres(ID);

Count--;

break;

}

if (iResult>0)

{

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

send(Connection[i].getsock(), (char\*)&s, sizeof(s), NULL);

}

}

std::cout << ID << " off\n";

}

std::string encode(std::string s)

{

std::string out;

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

out += (char)(s[i] + 120) % 125 + 12;

return out;

}

int main()

{

auto begin = std::chrono::steady\_clock::now(); //начало таймера

if (IsDebuggerPresent())ExitProcess(0);

BOOL f = false;

CheckRemoteDebuggerPresent(GetCurrentProcess(), &f);

if (f == true) ExitProcess(0);

setlocale(LC\_ALL, "russian");

ifstream in("countersign.txt");

string str;

std::getline(in, str);

in.close();

std::cout << "Enter password\n";

//скрывает вводимые символы

HANDLE hStdin = GetStdHandle(STD\_INPUT\_HANDLE);

DWORD mode = 0;

GetConsoleMode(hStdin, &mode);

SetConsoleMode(hStdin, mode & (~ENABLE\_ECHO\_INPUT));

//ввод пароля

string s;

getline(cin, s);

//таймер заканчивается

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count()/1000 << " s\n";

if (elapsed\_ms.count()/1000 >= 10)

{

std::cout << "Время истекло";

return 0;

}

if (encode(s) != str)

{

std::cout << "Error\n";

return 0;

}

SetConsoleMode(hStdin, mode);

std::cout << "Creating server:\n";

WSAData ws;

WORD version = MAKEWORD(2, 2);

int MasterSocket = WSAStartup(version, &ws);

if (MasterSocket != 0)

{

return 0;

}

struct addrinfo hints;

struct addrinfo \* result;

ZeroMemory(&hints, sizeof(hints));

//Задание сокетов

hints.ai\_family = AF\_INET;

hints.ai\_flags = AI\_PASSIVE;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

//установка ip и порта

std::cout << "Введите ip:\n";

std::string iport; std::cin >> iport;

std::cout << "Введите port:\n";

std::string port; std::cin >> port;

getaddrinfo(iport.c\_str(), port.c\_str(), &hints, &result);

//Заполнение сокета listen

Listen = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

bind(Listen, result->ai\_addr, result->ai\_addrlen);

listen(Listen, SOMAXCONN);

freeaddrinfo(result);

//Начало работы сервера

std::cout << "Start" << std::endl;

char c\_connect[] = "Connect";

while (1)

{

//проверка на получение сигнала от кого-нибудь

if (Connect = accept(Listen, NULL, NULL))

{

std::cout << c\_connect << ' ' << Count << std::endl;

Connection[Count] = Client::Client(&Connect, Count);

Count++;

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)SendM, (LPVOID)(Count - 1), NULL, NULL);

}

}

return 0;

}