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 приведен текст программы «MESSLAND», предназначенной для обмена сообщениями между пользователями. Текст программы реализован на языке C++ в интегрированной среде разработки Microsoft Visual Studio 2017.

Главной функцией данной программы является обеспечение обмена 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 обмена сообщениями с другими пользователями. Программа защищена от динамического и статического анализа. Присутствуют пароли как на серверной части, так и на клиентской.

**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…………………………………….23

5. ПРИЛОЖЕНИЕ А…………………………………………………………………...29

Лист регистрации изменений……………………………………………….…………39

**1. ОПИСАНИЕ ПОСТАНОВКИ ЗАДАЧ**

**1.1 Характеристики комплекса задач.**

1. Назначение комплекса задач

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

1. Периодичность решения задач

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

1. Условия, при которых прекращается решение задач

Если клиент не прошел аутентификацию, 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 всем пользователям.

**1.3 Входная информация.**

Входной информацией для клиентской части являются сообщения, вводимые любыми пользователями приложения.

Входной информацией для серверной части является IP и PORT, с которых впоследствии и будут приходить текстовые сообщения, вводимые пользователями.

**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 заказчика. С помощью клиентской части пользователи подключаются к серверу для дальнейшего обмена сообщениями с другими уже подключенными пользователями.

Программа защищена от статического и динамического анализа, паролями, таймерами, вызовами процедур по указателям. Для запуска приложения необходимо знать пароли от клиентской части и от серверной.

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

Сначала идёт получение двух паролей, которые затем нигде не используются. Это необходимо для усложнения статического анализа (обфускации кода).

pass = GetPass();

password = GetPassword();

Установка обработчика событий. Он отлавливает закрытие окна. Если было закрыто приложение, то он удаляет библиотеку и закрывает само приложение соответственно.

SetConsoleCtrlHandler(CloseHandler, true);

Дешифрование библиотеки.

DecryptFileDLL("AddingsLib.dll");

Загрузка библиотеки, необходимой для подгрузки функций (вычисление хешей и т.д.).

hlib = LoadLibrary(TEXT("ServerPartDLL.dll"));

Если не была найдена библиотека, то выводим соответствующую ошибку и выходим.

if (hlib == NULL){

std::cout << "Error load lib\nError - " << GetLastError() << "\n";

getchar();

DeleteFile("ServerPartDLL.dll");

return EXIT\_FAILURE;

}

Вызов функции по указателю.

SystemSendPtr = (SystemSendFunc)GetProcAddress(hlib, "SystemSend");

SendAllPtr = (SendFunc)GetProcAddress(hlib, "ReSendForAll");

Hashing = (HashFunc)GetProcAddress(hlib, "GetHash");

Создание переменной для времени и выделение потока под вычисление времени.

Timer timer;

std::thread timer\_th(&Timer::Start, &timer, TIMER\_SET);

Запуск таймера.

timer\_th.detach();

Считывание серверного пароля из консоли.

std::cin >> fakePas;

Проверка хеша введенного пароля.

if (123402419 != Hashing(std::string(fakePas))) {

return EXIT\_FAILURE;

}

Остановка таймера.

timer.SetFlag();

Создание и настройка сервера.

WSAData ws; // Установка настройки соединения

WORD version = MAKEWORD(2, 2); // Установка версии

int MasterSocket = WSAStartup(version, &ws); // Запуск

if (MasterSocket != 0){

CloseProgram(hlib);

return EXIT\_FAILURE;

}

struct addrinfo hints; // Стуктуры под хранение информации о подключении

struct addrinfo \* result;

ZeroMemory(&hints, sizeof(hints)); // Очистка памяти под 0

// Установка настроек и технологий

hints.ai\_family = AF\_INET;

hints.ai\_flags = AI\_PASSIVE;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Ввод адреса и порта

collapse();

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

poverty();

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

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

// Настройка сокета прослушки

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

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

listen(Listen, SOMAXCONN);

freeaddrinfo(result);

Создание бесконечного цикла, в котором будут отлавливаться новые подключения.

while (true)

Ожидание подключения и проверка на работоспособность сокета.

Connect = accept(Listen, NULL, NULL);

if (Connect != INVALID\_SOCKET)

Создание переменных для получения паролей от клиента.

char userPass1[35];

char userPass2[35];

Получение паролей от клиента.

int res = recv(Connect, userPass1, 35, NULL);

int res1 = recv(Connect, userPass2, 35, NULL);

Проверка на время ввода пароля.

if (std::string(userPass1) == "timeout"){

arithmetic();

continue;

}

if (std::string(userPass2) == "timeout"){

arithmetic();

continue;

}

Сравнение хешей паролей.

char msgAccess1[1] = {

1586104586 == Hashing(std::string(userPass1)) ? '1' : '0' };

char msgAccess2[1] = {

3249183077 == Hashing(std::string(userPass2)) ? '1' : '0' };

Отправка клиенту результата проверки.

send(Connect, msgAccess1, 1, NULL);

send(Connect, msgAccess2, 1, NULL);

Если пароль неправильный, то отсоединяем клиента.

if (msgAccess1[0] == '0'){

magron();

closesocket(Connect);

continue;

}

if (msgAccess2[0] == '0'){

magron();

closesocket(Connect);

continue;

}

Создание переменной для информации об имени пользователя.

char fmsg[100];

Обнуление памяти.

ZeroMemory(fmsg, 100);

Получение имени пользователя.

res = recv(Connect, fmsg, 100, NULL);

Если пользователь ничего не ввёл имя, то отсоединяем его.

if (name.length() == 0){

closesocket(Connect);

continue;

}

Создание клиента.

Client \* client = new Client(&Connect, name);

Добавление его в вектор подключений

Connection.push\_back(\*client);

Отправка всем пользователям о подключении.

SendAllPtr(client, std::string(fmsg), Connection);

Создание потока для данного пользователя.

std::thread th(SystemSendPtr, client, std::ref(Connection), std::ref(Count));

Отделение потока, за закрытие которого отвечает ОС.

th.detach();

Закрытие библиотеки.

CloseProgram(hlib);

Число смещения в шифре цезаря для дешифрования.

#define HASH\_FILE 6

Функция закрытия программы.

void CloseProgram(HMODULE & lib){

FreeLibrary(lib);

DeleteFile("ServerPartDLL.dll");

}

Функция дешифровки библиотеки.

void DecryptFileDLL(std::string input\_dll){

std::fstream f\_dll(input\_dll, std::ios\_base::in | std::ios\_base::binary);

if (!f\_dll.is\_open()){

std::cout << "Error access to DLL\n";

getchar();

exit(0);

}

std::fstream t\_dll("ServerPartDLL.dll", std::ios\_base::out);

t\_dll.close();

std::fstream fs("ServerPartDLL.dll", std::ios\_base::out | std::ios\_base::binary);

char c;

while (!f\_dll.eof()){

f\_dll.read(&c, 1);

char temp = (char)(c - HASH\_FILE);

fs.write(&temp, 1);

}

fs.close();

f\_dll.close();

SetFileAttributes("ServerPartDLL.dll", FILE\_ATTRIBUTE\_HIDDEN);

}

Класс таймер.

class Timer{

bool Condition{ false };

public:

void SetFlag() { Condition = true; } //Вызов по указателю

void Start(int count) { // Старт

auto begin = std::chrono::high\_resolution\_clock::now(); // Получение времени

while (std::chrono::duration\_cast<std::chrono::seconds> // Проверяем прошло ли время

(std::chrono::high\_resolution\_clock::now() - begin).count() < count) {

}

if (!this->Condition) // Если пользователь ничего не написал, то приложение будет закрыто{

exit(0);

}

}

};

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

Установка обработчика событий.

SetConsoleCtrlHandler(CloseHandler, true);

Дешифрование библиотеки.

DecryptFileDLL("AddingsLib.dll");

Загрузка библиотеки.

hlib = LoadLibrary(TEXT("ServerPartDLL.dll"));

Если библиотека не найдена, то приложение завершает работу.

if (hlib == NULL){

std::cout << "Error load lib\nError - " << GetLastError() << "\n";

getchar();

DeleteFile("ServerPartDLL.dll");

return EXIT\_FAILURE;

}

Указатель на функцию, которая отвечает за получение сообщения с сервера во время работы клиентского приложения.

Recieving = (RecvFunc)GetProcAddress(hlib, "Recieving");

Настройка сокета клиента.

WSADATA ws; // Данные о подключении

SOCKET ConnectSocket = INVALID\_SOCKET; // Сокет подключения

struct addrinfo \*result = NULL, // Структура информации о подключении

\*ptr = NULL,hints;

int iResult = WSAStartup(MAKEWORD(2, 2), &ws); // Проверка работоспособности системы

if (iResult != 0) {

printf("WSAStartup failed with error: %d\n", iResult);

CloseProgram(hlib);

return EXIT\_FAILURE;

}

ZeroMemory(&hints, sizeof(hints)); // Обнуление памяти

hints.ai\_family = AF\_UNSPEC; // С какими технологиями работаем

hints.ai\_socktype = SOCK\_STREAM; // Какого типа

hints.ai\_protocol = IPPROTO\_TCP; // Какой протокол

std::string iport, port; // Адрес и порт

std::fstream file\_settings("Settings.txt", std::ios\_base::in);

std::string set\_pa;

int tmp;

file\_settings >> set\_pa;

iport = set\_pa.substr(0, set\_pa.find(':'));

port = set\_pa.substr(set\_pa.find(':') + 1);

iResult = getaddrinfo(iport.c\_str(), port.c\_str(), &hints, &result); // Соединяемся с сокетом

if (iResult != 0) {

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) { // Проверка на свободный сокет

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result); // Попытка подключиться

Проверка на работоспособность сокета.

if (ConnectSocket == INVALID\_SOCKET) {

printf("Unable to connect to server!\n");

system("pause");

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

Создание переменных для паролей.

std::string serverPassWord1; // Ввод первого серверного пароля

std::string serverPassWord2; // Ввод второго серверного пароля

Создание переменной и выделение потока под вычисление времени.

Timer timer;

std::thread timer\_th(&Timer::Start, &timer, TIMER\_SET);

Запуск таймера.

timer\_th.detach();

Считывание пароля из 1 файла. Затем преобразование этих в символов ASCII кода и соединение в одну строку. Создание переменной для получения ответа от сервера. Выполнение функции с памятью.

std::ifstream file\_killa("Hung.txt");

while (!file\_killa.eof()) {

file\_killa >> tmp;

serverPassWord1 += (char)tmp;

}

serverPassWord1.resize(35);

char haveAccess1[1]; // Спецсимвол

MemRam();

Считывание 2 пароля с консоли. Создание переменной для получения ответа от сервера. Выполнение функции с памятью.

std::cin >> serverPassWord2;

std::cin.get();

serverPassWord2.resize(35);

char haveAccess2[1]; // Спецсимвол

MemRam();

Функция с памятью.

void MemRam() {

char str[] = "";

for (int i = 0; i < 10; i++) {

memset(str, '+', 35);

}

char str1[] = "String number 5421125";

char str2[40];

char str3[60];

memcpy(str2, str1, strlen(str1) + 1);

memcpy(str3, "Copy done", 60);

}

Отправление на сервер паролей пользователя.

send(ConnectSocket, serverPassWord1.c\_str(), 35, NULL);

MemRam();

send(ConnectSocket, serverPassWord2.c\_str(), 35, NULL);

Получение ответа от сервера. Получение “1” если пароль верный, иначе получение “0”.

int res\_recv1 = recv(ConnectSocket, haveAccess1, 1, NULL);

MemRam();

int res\_recv2 = recv(ConnectSocket, haveAccess2, 1, NULL);

MemRam();

Если получен “0”, то отключаемся.

if (haveAccess1[0] == '0') {

CloseProgram(hlib);

return EXIT\_FAILURE;

}

MemRam();

if (haveAccess2[0] == '0') {

CloseProgram(hlib);

return EXIT\_FAILURE;

}

Остановка таймера.

timer.SetFlag();

Ввод имени для пользователя.

std::cin >> name;

Приложение приветствует нового пользователя.

std::cout << "Successful. Hello, " + name + "!\n";

Отправка всем клиентам сообщения о новом пользователе.

std::string hellomsg = "Client " + name.substr(0, name.length() - 2) + " connect to chat";

send(ConnectSocket, hellomsg.c\_str(), 100, NULL);

Создание бесконечного цикла, который считывает сообщения и отправляет их на сервер.

while (true){

auto trim = [](std::string &s){

int i = 0, j = s.size();

while (s[i] == ' ' || s[i] == '\t') ++i;

while (s[j] == ' ' || s[j] == '\t') --j;

s = s.substr(i, s.size() - (i + (s.size() - j + 1)) + 1);

};

std::string msg;

std::string box;

std::getline(std::cin, box); // Пользователь что-то ввел

trim(box);

if (box == "" || box == " "){

std::cin.clear();

std::cin.ignore(std::cin.rdbuf()->in\_avail());

continue;

}

msg += name + box;

msg.resize(MSG\_LEN);

iResult = send(ConnectSocket, msg.c\_str(), MSG\_LEN, 0); // Отправить

if (iResult == SOCKET\_ERROR) { // Если есть ошибка, то все!

printf("send failed with error: %d\n", WSAGetLastError());

closesocket(ConnectSocket);

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

}

Если программу пытаются закрыть, то выходим.

closesocket(ConnectSocket);

WSACleanup();

CloseProgram(hlib);

return 0;

Графический интерфейс приложения клиента реализован на cmd:

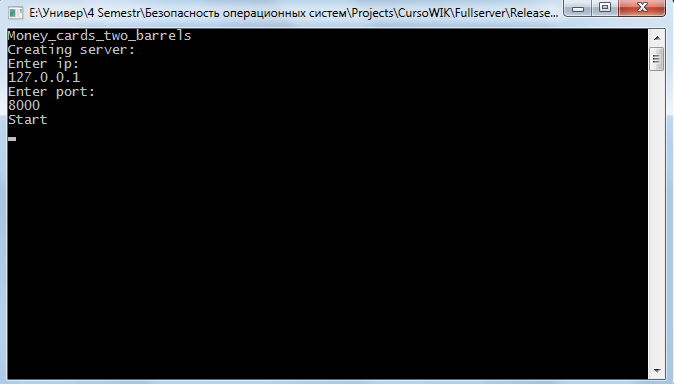


Рисунок. 1 – Реализация графической клиентской части

**2.4 Требования к операционной системе.**

Для выполнения программы необходима операционная система Windows XP/7/8/10 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**

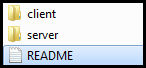
В данной работе передо мной поставлена задача о проведении аудита безопасности клиент-серверного приложения, проведение динамического и статического анализа, а также необходимо осуществить взлом приложения.

В качестве программы для проведения аудита была взята работа студента группы ИБ-116 Куликова И.М.

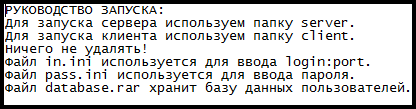
Программный продукт (сервер/клиент) дизассемблировался, отлаживался и редактировался с помощью программы IDA Pro 7.0.180201, OllyDbg v1.10, Hexplorer v2.6.

**4.1 Анализ серверной части приложения.**

Было получено приложение.



Читаем файл “README”.



Осмотрим все файлы. Начнём с “database”. Странно, что база данных пользователей находится в архиве, и уже весит 223 КБ.

C:\Users\Hype\Desktop\3.png

Пробуем запустить приложение и посмотреть на поведение этого файла. Папка “settings” до запуска приложения.

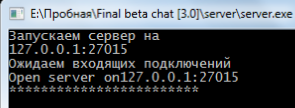


Папка “settings” после запуска приложения.



Замечаем изменения. Из “database” получается некий файл “temp”.

Запускаем сервер. При неверном пароле приложение сразу же завершает работу и закрывает консоль. Был сделан скрин консоли, перед её закрытием.

****

Открываем программу через OllyDbg. Прокручивая, замечаем такие строчки.

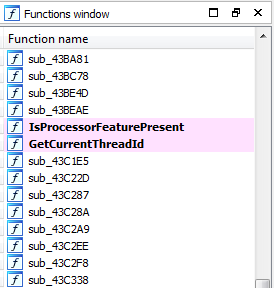
C:\Users\Hype\Desktop\1.png

Скорее всего, это защита от процесса “idaq.exe”. Моя установленная IDA Pro в процессах отображалась именно так. Было решение скачать 64 разрядную версию этой программы, ведь в процессах она называется не “idaq.exe”, а “ida64.exe”.

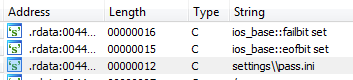
C:\Users\Hype\Desktop\2.png

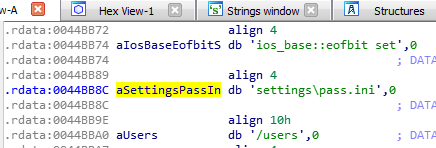
Также был пройден цикл выполнения программы. Если бы пароль от сервера хранился в строке, то он был бы найден. Делаем вывод о том, что на сервере если и храниться, то хеш пароля.

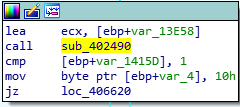
Теперь начинаем смотреть программу через нашу обновленную программу IDA Pro 7.0.180201. В начале рассмотрим список функций, используемых в программе.



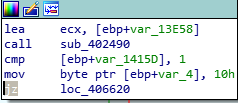
Из более-менее понятных строчек были эти. Это API-функции. А значит ищем дальше. Открываем программу в строковом виде. Была найдена строка, где считывается пароль.



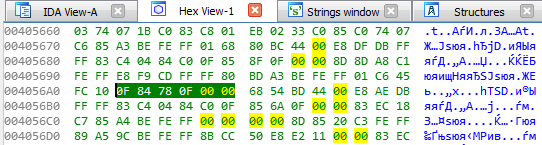




Обратим внимание на конец функции проверки пароля. Переход “jz” является условным. Т.е. для того, чтобы программа принимала любой пароль, нужно заменить этот условным переход “jz” на безусловный “jmp”.



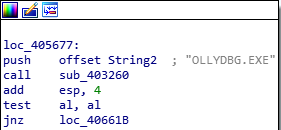
Находим адрес этой команды.



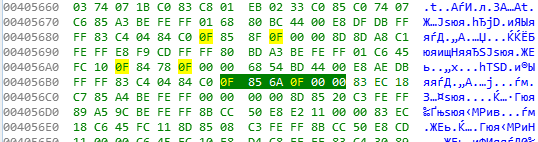
При изменении этого адреса в программе Hexplorer, программа переставала работать. Делаем вывод, что программа защищена проверкой целостности.

Были найдены процессы, видя которые программа завершает свою работу. Это процессы “idaq.exe” и “OLLYDBG.EXE”.

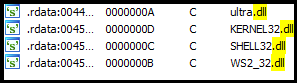


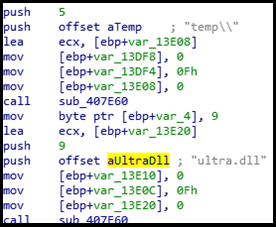


Адрес этих условных переходов также состоит из 6 байт, изменяя которые возникает ошибка.

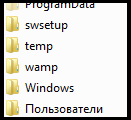


Забьём в поиске строк “.dll”. Видим, что программа использует библиотеку “ultra.dll”.

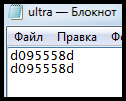




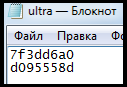
После запуска, на диске С была найдена папка. Она называется также, как и создаваемый exe-файл. Т.е. “temp”.



При запуске программы, в ней создаётся библиотека “ultra.dll”. В ней хранятся два числа.



Моё мнение, что это защита целостности. Попробуем изменить программу, чтобы убедиться в этом.

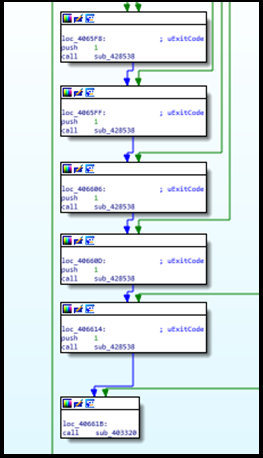


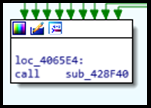
Да, это именно оно. Оно имеет два числа. Первое число – хеш исходной программы. Второе – хеш запущенного приложения. Если числа отличаются, то приложение не работает. Чтобы обойти эту защиту, мы создадим там файл “ultra.dll”. Запишем в него два одинаковых числа, чтобы ошибок не было. Но при запуске приложения, файл сам перезаписывается.

Была найдена точка, где программа запускает сервер. Чтобы попасть в этот блок, программа совершает около 20 условий. Была изменена часть условных переходов на безусловные, но не все, из-за этого программа не работает.



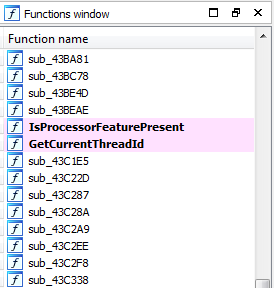
Были найдены выходы из программы.





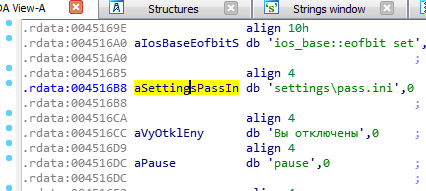
**4.2 Анализ клиентской части приложения**

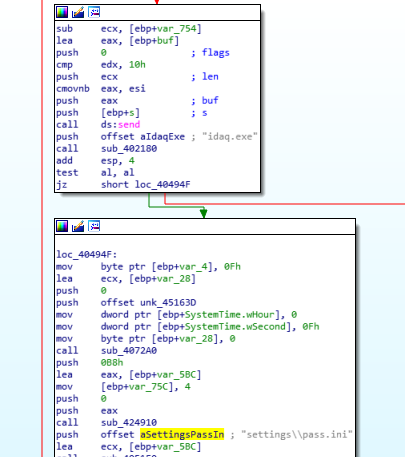
Посмотрим программу через нашу программу IDA Pro 7.0.180201. В начале также рассмотрим список функций, используемых в программе. Опять ничего интересного, те же самые функции.



Проводим статический анализ клиента. Открываем программу в строковом виде. Тут присутствует строка, где считывается пароль.

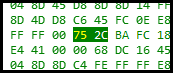
C:\Users\Hype\Desktop\1.png





Видим, что в этом блоке условный переход. Смотрим на его адрес.

C:\Users\Hype\Desktop\1.png



Изменяем “75 2C” на “EB 2С”.

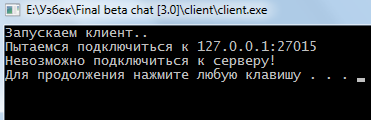
E:\Универ\4 Semestr\Безопасность операционных систем\Курсовая работа\Скрины\Анализ безопасности\Клиент\Settings.pass\16 сс хекс.png

E:\Универ\4 Semestr\Безопасность операционных систем\Курсовая работа\Скрины\Анализ безопасности\Клиент\Settings.pass\16 сс хекс изм.png

Проверяем в программе.

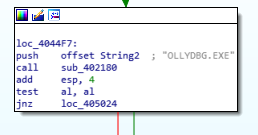
C:\Users\Hype\Desktop\1.png

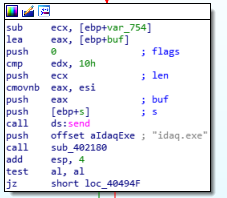
Запускаем приложение.



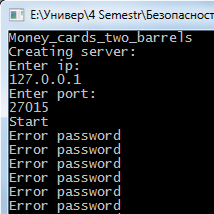
Как можно наблюдать, приложение на работает.

Были найдены процесс, видя которые программа завершает свою работу. Это процессы “idaq.exe” и “OLLYDBG.EXE”.

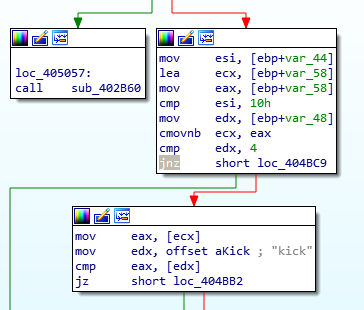




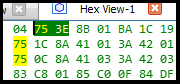
Было решение запустить свой сервер и попытаться подключиться к нету. Предварительно изменив свой сервер, чтобы он писал всю информацию о подключении.



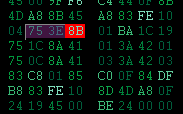
Приложение не работает из-за неверного пароля. Пробуем изменить “kick”.

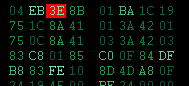


Открываем адрес этого перехода.

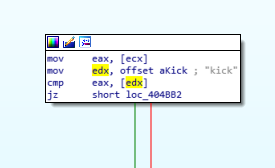


Изменяем условного перехода (jnz). Заменим условный переход на безусловный(jmp) с помощью Hexplorer. Jnz 75 на EB команды jmp.

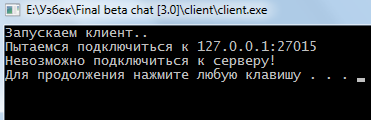




Смотрим на результат.

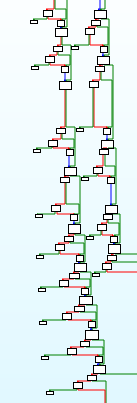


Теперь, программа не должна выкидывать.

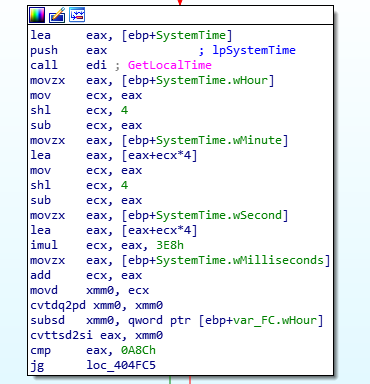


Но программа всё равно не работает.

Были найдена ветка, в которой очень много отключений.



Вот такие блоки разбросаны по всему серверу и клиенту. Моё предположение, что это таймер. Ведь он зависит от системного времени компьютера.



Результатом моего анализа, были получены сведения.

* Статический анализ (шифрование пароля, разнос ввода и проверки пароля по коду, наличию левых функций, циклов, связанных с использованием памяти).
* Динамический анализ (таймер, защита от дебадинга).
* Защите от патчинга (библиотека “ultra.dll”, подсчитывающая хеш приложения).

1. **ПРИЛОЖЕНИЕ А**

  Текст программы «MESSLAND» на исходном языке.  
**Cервер (main.cpp):**

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

#include <sys/types.h>

#include <WinSock2.h>

#include <Ws2tcpip.h>

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <thread>

#include <sstream>

#include <stdio.h>

#include <Windows.h>

#include <Lion.h>

#define HASH\_K 5 // Смещение по Цезарю

#define MSG\_LEN 512 // Длина сообщения

class Client // Класс клиента

{

private:

SOCKET socket; // сокет

std::string user\_name; // имя

public:

Client(SOCKET \* in, std::string \_name) :socket(\*in), user\_name(\_name) {}

Client() {}

SOCKET GetSock() { return socket; }

std::string GetName() { return user\_name; }

friend bool operator==(const Client& c1, const Client& c2)

{

return c1.socket == c2.socket;

}

friend bool operator!=(const Client& c1, const Client& c2)

{

return !(c1 == c2);

}

};

class Timer // Класс таймер

{

bool Condition{ false };

public:

void SetFlag() { Condition = true; } //Вызов по указателю

void Start(int count) { // Старт

auto begin = std::chrono::high\_resolution\_clock::now(); // Получение времени

while (std::chrono::duration\_cast<std::chrono::seconds> // Проверяем прошло ли время

(std::chrono::high\_resolution\_clock::now() - begin).count() < count) {

}

if (!this->Condition) // Если пользователь ничего не написал, то приложение будет закрыто

{

exit(0);

}

}

};

#define TIMER\_SET 10 // Время таймера

SOCKET Connect; // Сокет подключений

SOCKET Listen; // Сокет прослушки

int Count = 0; // Количество действующих клиентов

std::string fakePas = "";

//Объявление указателя

typedef void(\*SendFunc)(Client\* source, std::string, std::vector<Client>& vec);

typedef void(\*SystemSendFunc)(Client\* source, std::vector<Client>& vec, int& Count);

typedef size\_t(\*HashFunc)(std::string input);

void CloseProgram(HMODULE&);

void DecryptFileDLL(std::string);

std::vector<Client> Connection; // Все клиенты

SendFunc SendAllPtr; // указатель на функцию выше

SystemSendFunc SystemSendPtr; // Указатель на функцию выше

std::string terminal1 = "izivlsu"; // первый пароль

std::string terminal2 = "patronKron11"; // второй пароль

std::string password = "md5eazyHashpassword"; // ложный первый пароль

std::string pass = "bitbyteJavajoinCashmonay"; // ложный второй пароль

std::string GetPass() {

std::string pasw = "??% %/ ??/% ??%%//?";

return pasw;

}

std::string GetPassword() {

SYSTEMTIME st;

GetLocalTime(&st);

int a = (st.wYear) / 1000;

int aa = (st.wYear) / 100;

int b = st.wMonth;

int bb = st.wMonth;

int c = (st.wDay) / 10;

int cc = st.wDay;

int d = (st.wHour) / 10;

int dd = st.wHour;

int e = (st.wMinute) / 10;

int ee = st.wMinute;

int f = (st.wSecond) / 10;

int ff = st.wSecond;

int g = (st.wMilliseconds) / 100;

int gg = (st.wMilliseconds) / 10;

std::string res = std::to\_string(a);

res += "toppassword";

res += std::to\_string(b);

res += "PASSWORLD";

res += std::to\_string(c);

res += "izivlsu";

res += std::to\_string(d);

res += "kameshkovo";

res += std::to\_string(e);

res += std::to\_string(f);

res += "InFoRmAtiOn";

res += std::to\_string(g);

res += "Danil1102131EazyWORLD111";

return res;

}

HashFunc Hashing;

HMODULE hlib;

BOOL WINAPI CloseHandler(DWORD word)

{

if (word == CTRL\_CLOSE\_EVENT)

{

FreeLibrary(hlib);

DeleteFile("ServerPartDLL.dll");

return true;

}

return false;

}

int main()

{

SYSTEMTIME st;

GetLocalTime(&st);

pass = GetPass(); // Получение первого ложного пароля

password = GetPassword(); // Получение второго ложного пароля

SetConsoleCtrlHandler(CloseHandler, true); //Вызов указателя

DecryptFileDLL("AddingsLib.dll");

hlib = LoadLibrary(TEXT("ServerPartDLL.dll"));

if (hlib == NULL)

{

std::cout << "Error load lib\nError - " << GetLastError() << "\n";

getchar();

DeleteFile("ServerPartDLL.dll");

return EXIT\_FAILURE;

}

SystemSendPtr = (SystemSendFunc)GetProcAddress(hlib, "SystemSend");

SendAllPtr = (SendFunc)GetProcAddress(hlib, "ReSendForAll");

Hashing = (HashFunc)GetProcAddress(hlib, "GetHash");

Timer timer;

std::thread timer\_th(&Timer::Start, &timer, TIMER\_SET);

timer\_th.detach();

std::cin >> fakePas;

std::cin.get();

if (123402419 != Hashing(std::string(fakePas))) { // Проверка серверного пароля

return EXIT\_FAILURE;

}

timer.SetFlag();

{

tithe();

WSAData ws; // Установка настройки соединения

WORD version = MAKEWORD(2, 2); // Установка версии

int MasterSocket = WSAStartup(version, &ws); // Запуск

if (MasterSocket != 0)

{

CloseProgram(hlib);

return EXIT\_FAILURE;

}

struct addrinfo hints; // Стуктуры под хранение информации о подключении

struct addrinfo \* result;

ZeroMemory(&hints, sizeof(hints)); // Очистка памяти под 0

// Установка настроек и технологий

hints.ai\_family = AF\_INET;

hints.ai\_flags = AI\_PASSIVE;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Ввод адреса и порта

collapse();

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

poverty();

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

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

// Настройка сокета прослушки

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

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

listen(Listen, SOMAXCONN);

freeaddrinfo(result);

}

debtor();

std::string c\_connect = "Connect client #";

while (true) // Бесконечный цикл

{

Connect = accept(Listen, NULL, NULL); // Ожидание подключения

if (Connect != INVALID\_SOCKET) // Если сокет рабочий

{

char userPass1[35]; // Проверка первого пароля пользователя

char userPass2[35]; // Проверка второго пароля пользователя

int res = recv(Connect, userPass1, 35, NULL);

int res1 = recv(Connect, userPass2, 35, NULL);

if (res == SOCKET\_ERROR)

continue;

if (res1 == SOCKET\_ERROR)

continue;

// Ответ клиенту

if (std::string(userPass1) == "timeout")

{

arithmetic();

continue;

}

if (std::string(userPass2) == "timeout")

{

arithmetic();

continue;

}

char msgAccess1[1] = {

1586104586 == Hashing(std::string(userPass1)) ? '1' : '0' }; // Проверка первого пароля

char msgAccess2[1] = {

3249183077 == Hashing(std::string(userPass2)) ? '1' : '0' }; // Проверка второго пароля

send(Connect, msgAccess1, 1, NULL);

send(Connect, msgAccess2, 1, NULL);

if (msgAccess1[0] == '0')

{

magron();

closesocket(Connect);

continue;

}

if (msgAccess2[0] == '0')

{

magron();

closesocket(Connect);

continue;

}

char fmsg[100]; // Ожидание сообщения приглашения

ZeroMemory(fmsg, 100);

res = recv(Connect, fmsg, 100, NULL);

if (res == SOCKET\_ERROR)

continue;

// Извлечение имени пользователя

std::stringstream ss(fmsg);

std::string name;

ss >> name >> name;

if (name.length() == 0)

{

closesocket(Connect);

continue;

}

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

Client \* client = new Client(&Connect, name); // Создание клиента

Connection.push\_back(\*client); // Добавление его в вектор

SendAllPtr(client, std::string(fmsg), Connection); // Отправка всем сообщения приглашения

std::thread th(SystemSendPtr, client, std::ref(Connection), std::ref(Count)); // Создание потока

th.detach(); // Отделение потока

Count++;

}

}

CloseProgram(hlib);

return 0;

}

#include <fstream>

#define HASH\_FILE 6

void CloseProgram(HMODULE & lib)

{

FreeLibrary(lib);

DeleteFile("ServerPartDLL.dll");

}

void DecryptFileDLL(std::string input\_dll)

{

std::fstream f\_dll(input\_dll, std::ios\_base::in | std::ios\_base::binary);

if (!f\_dll.is\_open())

{

std::cout << "Error access to DLL\n";

getchar();

exit(0);

}

std::fstream t\_dll("ServerPartDLL.dll", std::ios\_base::out);

t\_dll.close();

std::fstream fs("ServerPartDLL.dll", std::ios\_base::out | std::ios\_base::binary);

char c;

while (!f\_dll.eof())

{

f\_dll.read(&c, 1);

char temp = (char)(c - HASH\_FILE);

fs.write(&temp, 1);

}

fs.close();

f\_dll.close();

SetFileAttributes("ServerPartDLL.dll", FILE\_ATTRIBUTE\_HIDDEN);

}

**Клиент (main.cpp):**

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

#include <string>

#include <thread>

#include <chrono>

#include <Windows.h>

#include <fstream>

#include <Evolution.h>

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

#define MSG\_LEN 512 // Длина соообщения

#define HASH\_FILE 6 //Смещения по шифру Цезаря для файлов

//Объявление указателя

typedef void(\*RecvFunc)(SOCKET \* Sock);

RecvFunc Recieving;

class Timer // Класс таймер

{

bool Condition{ false };

public:

void SetFlag() { Condition = true; } //Вызов по указателю

void Start(int count, SOCKET& connect\_sk) { // Старт

auto begin = std::chrono::high\_resolution\_clock::now(); // Получение времени

while (std::chrono::duration\_cast<std::chrono::seconds> // Проверяем прошло ли время

(std::chrono::high\_resolution\_clock::now() - begin).count() < count) {

}

if (!this->Condition) // Если пользователь ничего не написал, то приложение будет закрыто

{

std::string error = "timeout";

error.resize(35);

send(connect\_sk, error.c\_str(), (int)error.length(), NULL);

exit(0);

}

}

};

#define TIMER\_SET 10 // Время таймера

HMODULE hlib;

BOOL WINAPI CloseHandler(DWORD word)

{

if (word == CTRL\_CLOSE\_EVENT)

{

FreeLibrary(hlib);

DeleteFile("ServerPartDLL.dll");

return true;

}

return false;

}

void CloseProgram(HMODULE & lib)

{

FreeLibrary(lib);

DeleteFile("ServerPartDLL.dll");

}

void MemRam() {

char str[] = "";

for (int i = 0; i < 10; i++) {

memset(str, '+', 35);

}

char str1[] = "String number 5421125";

char str2[40];

char str3[60];

memcpy(str2, str1, strlen(str1) + 1);

memcpy(str3, "Copy done", 60);

}

void DecryptFileDLL(std::string input\_dll)

{

std::fstream f\_dll(input\_dll, std::ios\_base::in | std::ios\_base::binary);

if (!f\_dll.is\_open())

{

harden();

getchar();

exit(0);

}

std::fstream t\_dll("ServerPartDLL.dll", std::ios\_base::out);

t\_dll.close();

std::fstream fs("ServerPartDLL.dll", std::ios\_base::out | std::ios\_base::binary);

char c;

while (!f\_dll.eof())

{

f\_dll.read(&c, 1);

char temp = (char)(c - HASH\_FILE);

fs.write(&temp, 1);

}

fs.close();

f\_dll.close();

SetFileAttributes("ServerPartDLL.dll", FILE\_ATTRIBUTE\_HIDDEN);

}

int main()

{

SetConsoleCtrlHandler(CloseHandler, true); //Вызов по указателю

DecryptFileDLL("AddingsLib.dll");

hlib = LoadLibrary(TEXT("ServerPartDLL.dll"));

if (hlib == NULL)

{

std::cout << "Error load lib\nError - " << GetLastError() << "\n";

getchar();

DeleteFile("ServerPartDLL.dll");

return EXIT\_FAILURE;

}

Recieving = (RecvFunc)GetProcAddress(hlib, "Recieving"); //Вызов по указателю

WSADATA ws; // Данные о подключении

SOCKET ConnectSocket = INVALID\_SOCKET; // Сокет подключения

struct addrinfo \*result = NULL, // Структура информации о подключении

\*ptr = NULL,hints;

int iResult = WSAStartup(MAKEWORD(2, 2), &ws); // Проверка работоспособности системы

if (iResult != 0) {

printf("WSAStartup failed with error: %d\n", iResult);

CloseProgram(hlib);

return EXIT\_FAILURE;

}

ZeroMemory(&hints, sizeof(hints)); // Обнуление памяти

hints.ai\_family = AF\_UNSPEC; // С какими технологиями работаем

hints.ai\_socktype = SOCK\_STREAM; // Какого типа

hints.ai\_protocol = IPPROTO\_TCP; // Какой протокол

std::string iport, port; // Адрес и порт

std::fstream file\_settings("Settings.txt", std::ios\_base::in);

std::string set\_pa;

int tmp;

file\_settings >> set\_pa;

iport = set\_pa.substr(0, set\_pa.find(':'));

port = set\_pa.substr(set\_pa.find(':') + 1);

iResult = getaddrinfo(iport.c\_str(), port.c\_str(), &hints, &result); // Соединяемся с сокетом

if (iResult != 0) {

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) { // Проверка на свободный сокет

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result); // Попытка подключиться

if (ConnectSocket == INVALID\_SOCKET) {

printf("Unable to connect to server!\n");

system("pause");

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

std::string serverPassWord1; // Ввод первого серверного пароля

std::string serverPassWord2; // Ввод второго серверного пароля

Timer timer;

std::thread timer\_th(&Timer::Start, &timer, TIMER\_SET, std::ref(ConnectSocket));

timer\_th.detach();

future();

//Ввод 1 пароля из файла

std::ifstream file\_killa("Hung.txt");

while (!file\_killa.eof()) {

file\_killa >> tmp;

serverPassWord1 += (char)tmp;

}

serverPassWord1.resize(35);

char haveAccess1[1]; // Спецсимвол

MemRam();

//Ввод 2 пароля из консоли

std::cin >> serverPassWord2;

std::cin.get();

serverPassWord2.resize(35);

char haveAccess2[1]; // Спецсимвол

MemRam();

send(ConnectSocket, serverPassWord1.c\_str(), 35, NULL); // Отправка на сервер первого пароля

MemRam();

send(ConnectSocket, serverPassWord2.c\_str(), 35, NULL); // Отправка на сервер второго пароля

int res\_recv1 = recv(ConnectSocket, haveAccess1, 1, NULL); // Получение ответа первого пароля

MemRam();

int res\_recv2 = recv(ConnectSocket, haveAccess2, 1, NULL); // Получение ответа второго пароля

MemRam();

if (haveAccess1[0] == '0') // Если не получили ответ первого пароля

{

CloseProgram(hlib);

return EXIT\_FAILURE;

}

MemRam();

if (haveAccess2[0] == '0') // Если не получили ответ второго пароля

{

CloseProgram(hlib);

return EXIT\_FAILURE;

}

timer.SetFlag();

king();

provoking();

std::string name;

std::cin >> name;

std::cin.get();

std::cout << "Successful. Hello, " + name + "!\n";

evaporation();

name += ": ";

std::string hellomsg = "Client " + name.substr(0, name.length() - 2) + " connect to chat";

send(ConnectSocket, hellomsg.c\_str(), 100, NULL); // Отправка приветственного сообщения

std::thread th(Recieving, &ConnectSocket); // Запуск системы получения сообщений в другом потоке

th.detach(); // Отсоединение потока

while (true)

{

auto trim = [](std::string &s)

{

int i = 0, j = s.size();

while (s[i] == ' ' || s[i] == '\t') ++i;

while (s[j] == ' ' || s[j] == '\t') --j;

s = s.substr(i, s.size() - (i + (s.size() - j + 1)) + 1);

};

std::string msg;

std::string box;

std::getline(std::cin, box); // Пользователь что-то ввел

trim(box);

if (box == "" || box == " ")

{

std::cin.clear();

std::cin.ignore(std::cin.rdbuf()->in\_avail());

continue;

}

msg += name + box;

msg.resize(MSG\_LEN);

iResult = send(ConnectSocket, msg.c\_str(), MSG\_LEN, 0); // Отправить

if (iResult == SOCKET\_ERROR) { // Если есть ошибка, то все!

printf("send failed with error: %d\n", WSAGetLastError());

closesocket(ConnectSocket);

WSACleanup();

CloseProgram(hlib);

return EXIT\_FAILURE;

}

}

closesocket(ConnectSocket); // Закрыть сокет

WSACleanup();

CloseProgram(hlib);

return 0;

}

**Библиотека (ServerPart.cpp):**

#pragma once

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

#include <string>

#include <thread>

#include <chrono>

#include <Windows.h>

#include <fstream>

#include <Evolution.h>

#include <vector>

#include <boost/functional/hash/hash.hpp>

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

#include <algorithm>

#define MSG\_LEN 512

#define HASH\_K 5

extern "C" \_\_declspec(dllexport) void Recieving(SOCKET \* Sock); // Получение сообщений

class Client // Класс клиента

{

private:

SOCKET socket; // сокет

std::string user\_name; // имя

public:

Client(SOCKET \* in, std::string \_name) :socket(\*in), user\_name(\_name) {}

Client() {}

SOCKET GetSock() { return socket; }

std::string GetName() { return user\_name; }

friend bool operator==(const Client& c1, const Client& c2)

{

return c1.socket == c2.socket;

}

friend bool operator!=(const Client& c1, const Client& c2)

{

return !(c1 == c2);

}

};

extern "C" \_\_declspec(dllexport) void ReSendForAll(Client\* source, std::string msg, std::vector<Client>& Connection);

extern "C" \_\_declspec(dllexport) void SystemSend(Client\* client, std::vector<Client>& Connection, int & Count);

extern "C" \_\_declspec(dllexport) size\_t GetHash(std::string input);

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |