Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №4

на тему

**ЗАЩИТА АТАКИ ПРИ УСТАНОВКИ TCP-СОЕДИНЕНИЯ И ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ**

Выполнил: студент гр.253504 Лавренова А.С.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

Введение3

[1  [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2  Теоретические сведения 5](#_Toc178067890)

[3  Описание и пример выполнения программы 7](#_Toc178067891)

[3.1  Реализация серверного приложения 7](#_Toc178067891)

[3.2  Реализация клиентского приложения 8](#_Toc178067891)

[3.3  Реализация атаки 8](#_Toc178067891)

[3.4  Пример выполнения программы 10](#_Toc178067891)

Заключение [13](#_Toc178067899)

[Список использованных источников 14](#_Toc178067900)

[Приложение А (обязательное) 15](#_Toc178067901)

# ВВЕДЕНИЕ

Сетевые приложения, использующие протокол TCP, подвержены различным атакам, связанным с установкой соединений. Одной из наиболее распространенных угроз является атака типа SYN-flood, при которой злоумышленник отправляет множество запросов на установление соединения, но не завершает процесс рукопожатия. Это приводит к исчерпанию ресурсов сервера и отказу в обслуживании легитимных клиентов. Важно учитывать эти угрозы при разработке серверных приложений и применять механизмы защиты.

В рамках данной работы рассматривается процесс установления TCP-соединений и разработка серверного приложения, устойчивого к подобным атакам. Для этого используется среда программирования и отладки языков высокого уровня, что позволяет реализовать сервер с защитными мерами. Важным этапом является применение сетевых утилит операционной системы, которые помогают отслеживать активные подключения и анализировать состояние сервера в реальном времени.

Для проверки работы сервера и исследования сетевого взаимодействия применяется анализ сетевого трафика. В частности, утилита WireShark используется для мониторинга TCP-пакетов, что позволяет изучить трехстороннее рукопожатие, зафиксировать возможные попытки атаки и убедиться в корректности работы реализованных механизмов защиты. Это позволяет детально проанализировать сетевую активность и оценить эффективность примененных решений.

Таким образом, данная лабораторная работа направлена на углубление наших знаний в области сетевой безопасности, а также на разработку эффективных методов защиты приложений.

1 ЦЕЛЬ РАБОТЫ

Цель лабораторной работы заключается в разработке и реализации серверного приложения, защищенного от различных типов атак при установке TCP-соединения, а также в проектировании и внедрении прикладного протокола для безопасного обмена данными между клиентом и сервером. В рамках данной работы будет проведено исследование методов защиты от атак на этапах установления TCP-соединения, таких как SYN-флуд, атаки на уязвимости протокола TCP, а также других угроз, направленных на нарушению нормального функционирования сетевых сервисов. Особое внимание будет уделено защите сервера от несанкционированного доступа и обеспечению его устойчивости к различным видам сетевых угроз.

Для реализации серверной части будет использована среда программирования на языке высокого уровня (например, C++), а также различные средства отладки и разработки, позволяющие эффективно тестировать и анализировать функциональность приложения. Важным аспектом работы является использование сетевых утилит операционной системы для защиты и фильтрации трафика, а также средств анализа сетевого трафика, таких как Wireshark, для мониторинга, исследования и выявления потенциальных уязвимостей в процессе установления соединений и передачи данных.

Задача также включает в себя разработку прикладного протокола для безопасного обмена данными, включая шифрование, аутентификацию и другие методы защиты данных на уровне приложения. В рамках работы будет необходимо протестировать устойчивость серверного приложения к атакам на уровне TCP и приложений, а также провести анализ трафика для оценки надежности системы и ее защиты от различных видов сетевых угроз.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

TCP (Transmission Control Protocol) — это один из основных протоколов интернета, который обеспечивает надежную передачу данных между устройствами. Он работает поверх IP (Internet Protocol) и отвечает за установление соединения, передачу данных и контроль их целостности. TCP широко используется в различных приложениях, таких как веб-серфинг, электронная почта и обмен файлами. Этот протокол был разработан для обеспечения надежной и последовательной передачи данных, что особенно важно в условиях нестабильных сетей и больших расстояний между устройствами. [1]

TCP является частью стека протоколов TCP/IP, который является основой для передачи данных в интернете. В отличие от других протоколов, таких как UDP (User Datagram Protocol), TCP гарантирует доставку данных в правильном порядке и без потерь. Это достигается за счет использования различных механизмов, таких как контроль ошибок, управление потоком и контроль перегрузки. Благодаря этим характеристикам, TCP стал стандартом для многих интернет-приложений, требующих надежной передачи данных.

Любое TCP-соединение [начинается с установления соединения](https://hpbn.co/building-blocks-of-tcp/#slow-start). Этот процесс называется рукопожатием (handshake). Перед тем как соединение между клиентом и сервером будет установлено, они должны согласовать стартовую последовательность чисел, с которой начинается передача данных. Последовательность чисел уникальна для каждого соединения. В TCP рукопожатие проходит в три этапа:

1. Отправитель генерирует случайное число *x* и передаёт его получателю в пакете SYN (от Sequence number). В этом же пакете могут указываться какие-то специальные флаги для настройки соединения.
2. Получатель прибавляет к числу *x* единицу и генерирует своё число *y*. После этого получатель посылает отправителю числа *x + 1* и *y* в пакете SYN ACK (от Acknowledgment Number).
3. Завершающим этапом является передача пакета ACK отправителем с числами *x + 1* и *y + 1*.

После рукопожатия отправитель посылает поток данных получателю. Отправитель может посылать данные сразу после посылки ACK пакета, получатель же должен подождать прихода этого пакета перед приёмом потока данных. Такой механизм рукопожатия и отправки данных обязателен для каждого соединения при общении по протоколу TCP. ПакетTCP — это блок данных, который передаётся в потоке. Порядок пакетов строго определён и не может быть нарушен. [2]

TCP обладает несколькими ключевыми характеристиками, которые делают его незаменимым для надежной передачи данных:

1. Надежность. TCP гарантирует доставку данных в правильном порядке и без потерь. Это достигается за счет использования механизмов квитирования (ACK) и повторной передачи потерянных сегментов. Если сегмент данных не был получен, отправитель повторно отправляет его до тех пор, пока не получит подтверждение.
2. Потоковая передача. Данные передаются в виде потока байтов, что позволяет передавать большие объемы информации. Это особенно важно для приложений, таких как видеостриминг и онлайн-игры, где требуется передача больших объемов данных в реальном времени.
3. Контроль перегрузки. TCP регулирует скорость передачи данных, чтобы избежать перегрузки сети. Это достигается за счет использования алгоритмов, таких как алгоритм медленного старта и алгоритм избегания перегрузки. Эти алгоритмы позволяют динамически регулировать скорость передачи данных в зависимости от состояния сети.
4. Контроль ошибок. Используются контрольные суммы для проверки целостности данных. Каждое сообщение TCP содержит контрольную сумму, которая проверяется получателем. Если контрольная сумма не совпадает, данные считаются поврежденными и запрашивается их повторная передача.

Wireshark – это широко распространённый инструмент для захвата и анализа сетевого трафика, который активно используется как для образовательных целей, так и для устранения неполадок на компьютере или в сети. Wireshark работает практически со всеми протоколами модели OSI, обладает понятным для обычного пользователя интерфейсом и удобной системой фильтрации данных. Помимо всего этого, программа является кроссплатформенной и поддерживает следующие операционные системы: Windows, Linux, Mac OS X, Solaris, FreeBSD, NetBSD, OpenBSD. [3]

В заключение, можно отметить, что протокол TCP является основой для надежной передачи данных в интернете, обеспечивая гарантированную доставку информации в правильном порядке и без потерь. Его надежность обеспечивается через механизмы подтверждений (ACK), повторной передачи потерянных сегментов и использование контрольных сумм для проверки целостности данных. Контроль перегрузки и потоковая передача делают TCP идеальным выбором для приложений, требующих стабильности и высокоскоростной передачи данных, таких как видеостриминг и онлайн-игры.

Кроме того, инструменты, такие как Wireshark, предоставляют мощные возможности для анализа и устранения проблем в сетевом трафике, позволяя отслеживать пакеты и диагностировать неполадки. Совмещение теоретических знаний о TCP и практических инструментов анализа трафика предоставляет ценные ресурсы для разработки и оптимизации сетевых приложений. За счет этого TCP продолжает оставаться стандартом для сетевых коммуникаций, а применение таких инструментов, как Wireshark, позволяет более глубоко понять работу сетевых протоколов и повысить безопасность и эффективность серверных приложений.

**3 ОПИСАНИЕ И ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ**

В данной лабораторной работе рассматривается реализация серверного и клиентского приложений с использованием сокетов для обмена данными через TCP-соединение. Сервер обрабатывает подключения клиентов, проводит аутентификацию с использованием пароля и осуществляет обмен сообщениями между клиентами. Клиенты могут отправлять сообщения, которые будут расшифрованы, если это необходимо, или сохранены в базе данных с последующим рассыланием хеша другим клиентам. В этой главе будет подробно описано выполнение программы, с добавлением скриншотов работы серверной и клиентской части.

В данной лабораторной работе сервер и клиент используют TCP-соединение для передачи сообщений и хешей. Рассмотрим более подробно, как именно TCP-соединение используется на стороне сервера и клиента.

**3.1 Реализация серверного приложения**

Серверное приложение создает и настраивает TCP-сокет, который будет ожидать входящие подключения от клиентов, а затем взаимодействовать с ними через это соединение. Описание ключевых шагов:

1. Инициализация Winsock. Для работы с сокетами на платформе Windows необходимо выполнить инициализацию библиотеки Winsock с помощью функции WSAStartup, которая загружает библиотеку и готовит среду для работы с сетевыми протоколами. Это обязательный шаг, так как без него невозможно использовать сетевые функции Windows, такие как создание и использование сокетов.
2. Создание сокета. Для установления TCP-соединения серверу необходимо создать сокет с использованием функции socket, которая принимает параметры AF\_INET — семейство адресов (IPv4), SOCK\_STREAM — тип сокета, который используется для TCP-соединений и протокол 0 указывает на использование протокола, связанного с типом сокета, то есть TCP. Если сокет не был создан, программа завершится с ошибкой.
3. Привязка сокета к порту. Для того чтобы сервер мог прослушивать соединения, его сокет должен быть привязан к определенному порту на машине. Для этого используется функция bind, которая принимает структуру sockaddr\_in, в которой указаны IP-адрес сервера и порт для прослушивания (в данном случае 5000). Этот шаг необходим для того, чтобы сервер мог принимать подключения от клиентов, присоединяющихся к его IP-адресу и порту.
4. Прослушивание подключений. После привязки сокета к порту сервер начинает прослушивание с помощью функции listen. Эта функция сообщает системе, что сокет будет использоваться для принятия входящих соединений.
5. Принятие подключения. Когда клиент подключается к серверу, сервер принимает соединение с помощью функции accept. Эта функция блокирует выполнение программы, пока не будет установлено подключение. Когда соединение установлено, сервер получает новый сокет для общения с этим клиентом.
6. Обработка клиента. После того как сервер принял подключение, он создает отдельный поток для обработки запросов клиента, который включает запрос пароля для аутентификации, обработку и отправку сообщений, сохранение хешированных сообщений в базе данных, рассылку сообщений всем другим подключенным клиентам. Этот процесс реализуется в функции HandleClient, где для каждого клиента выделяется отдельный поток.

**3.2 Реализация клиентского приложения**

Клиентское приложение выполняет подключение к серверу, отправляет и принимает сообщения. Основные этапы взаимодействия с сервером через TCP-соединение:

1. Создание сокета. Для установления TCP-соединения клиент также создает сокет с помощью той же функции socket.
2. Подключение к серверу. С помощью функции connect клиент подключается к серверу, указывая его IP-адрес и порт. Важно, чтобы сервер был запущен и прослушивал подключения на этом порту.
3. Если подключение невозможно (например, сервер не доступен или неверно указан порт), клиент завершит выполнение с ошибкой.
4. Обмен сообщениями с сервером. После успешного подключения клиент может отправлять сообщения на сервер и получать ответы. Для этого используется функция send для отправки данных и recv для получения.

**3.3 Реализация атаки**

Атака на сервер заключается в многократных попытках установки TCP-соединения за короткий промежуток времени. Данная атака может привести к исчерпанию доступных соединений на сервере, создавая отказ в обслуживании (DoS).

Программа-атакующий создает 10 попыток подключения к серверу, используя функцию connect(). После установления соединения сокет сразу закрывается, но сам сервер может не успеть обработать такие соединения, что потенциально может привести к перегрузке. Изображение невозможности подключения к серверу изображено на рисунке 3.1.

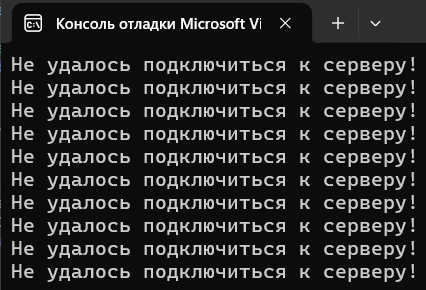


Рисунок 3.1 – Невозможность подключиться к серверу

Для защиты сервера была внедрена система ограничения подключений с одного IP-адреса за определенное время. Это реализуется с помощью хранения временных меток подключений и фильтрации IP-адресов, которые делают слишком много подключений в заданный промежуток времени.

Основные механизмы защиты:

1. Ограничение частоты подключений. Ведется учет подключений каждого IP-адреса. Если количество подключений за 10 секунд превышает 3, новые подключения от этого IP блокируются.
2. Очистка старых записей. При каждом новом подключении удаляются записи, время которых превышает заданный порог (10 секунд).
3. Вывод предупреждения. Если IP-адрес превышает лимит, клиенту отправляется уведомление и соединение закрывается.

Блокировка подключения изображена на рисунке 3.2.

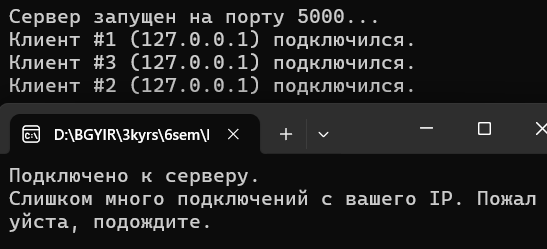


Рисунок 3.2 – Ответ сервера другим клиентам при атаке

Рассмотрим работу Wireshark во время атаки (рисунок 3.3).

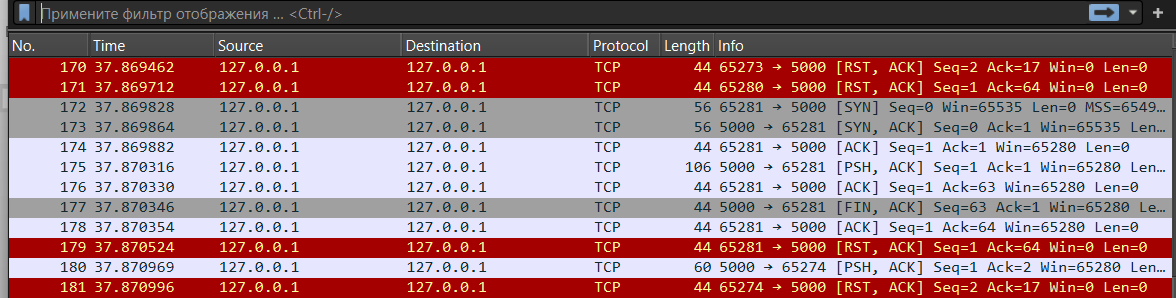


Рисунок 3.3 – Работа Wireshark во время атаки

В Wireshark видны все попытки подключения к серверу. Каждая строка представляет отдельный пакет данных, который содержит информацию о TCP-соединениях.

Столбцы:

* Time: показывает время, когда был захвачен пакет;
* Source: IP-адрес атакующего клиента, который пытается установить соединение;
* Destination: IP-адрес сервера, к которому производится подключение;
* Protocol: указывает, что используется TCP;
* Length: длина пакета в байтах;
* Info: содержит информацию о типе пакета, например, [SYN] для начала установки соединения.

На рисунке можно увидеть, как атакующий отправляет множество SYN-запросов к серверу. Каждый из этих запросов пытается инициировать новое TCP-соединение. В ответ на каждый SYN-запрос сервер отправляет SYN-ACK, подтверждая получение запроса. Однако, если атака продолжается слишком долго, сервер может не успеть обработать все запросы, что может привести к перегрузке. Многие из установленных соединений закрываются сразу после их создания, что также отображается в Wireshark, где можно наблюдать FIN-пакеты.

Реализованная защита позволяет снизить вероятность успешного проведения атаки, ограничивая число подключений с одного IP в определенное время. Такой механизм эффективен против простых DoS-атак и перегрузки сервера за счет избыточных соединений.

**3.4 Пример работы программы**

После запуска серверного приложения, оно слушает входящие подключения от клиентов. На рисунке 3.4 будет видно сообщение о запуске сервера.

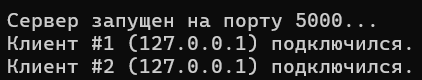


Рисунок 3.4 – Запуск серверного приложения и успешное подключение клиента

После подключения клиента сервер запрашивает ввод пароля. Если пароль введен правильно, клиент получает доступ, иначе — отключается (рисунок 3.5).

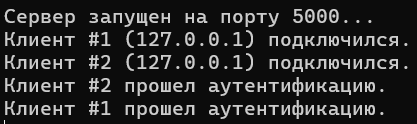


Рисунок 3.5 – Вид успешной аутентификация клиента на стороне сервера

После успешной аутентификации клиенты могут отправлять сообщения. Сервер рассылает сообщения всем подключенным клиентам (рисунок 3.6).

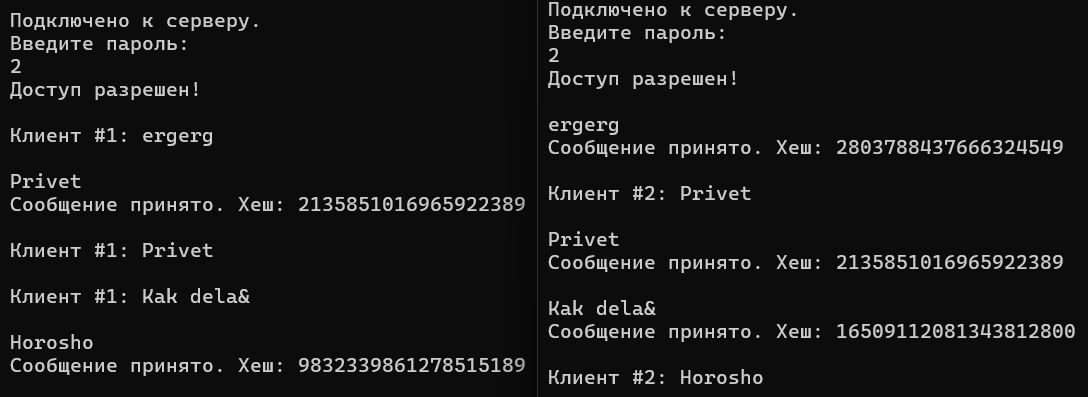


Рисунок 3.6 – Обмен сообщениями между клиентами через сервер

Когда клиент отправляет сообщение, сервер сохраняет его хеш в базе данных. Если клиент отправляет хеш, сервер расшифровывает его, если это возможно (рисунок 3.7).

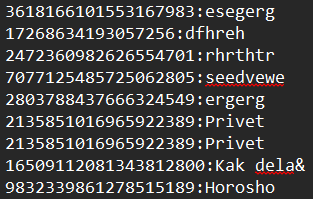


Рисунок 3.7 – Хранение хешей

Реализованная защита позволяет снизить вероятность успешного проведения атаки, ограничивая число подключений с одного IP в определенное время. Такой механизм эффективен против простых DoS-атак и перегрузки сервера за счет избыточных соединений.

В данной главе была рассмотрена реализация серверного и клиентского приложений для обмена данными через TCP-соединение. Были подробно описаны основные этапы работы обеих частей системы, включая инициализацию, установление соединения, обработку сообщений и защиту сервера от атак.

Особое внимание уделено механизму защиты от частых подключений с одного IP-адреса, что позволило существенно снизить вероятность DoS-атак. Введенные ограничения на частоту соединений с сервером обеспечивают стабильную работу и предотвращают перегрузку.

Примеры выполнения программы, сопровождаемые скриншотами, демонстрируют корректную работу системы: успешное подключение клиентов, прохождение аутентификации, обмен сообщениями и сохранение хешированных данных. Реализация механизма блокировки IP-адресов атакующего также подтверждена на практике.

Таким образом, разработанное приложение обеспечивает надежную передачу данных между клиентами и защищает сервер от потенциальных угроз, повышая общую устойчивость системы.

# ЗАКЛЮЧЕНИЕ

В ходе работы была реализована серверная часть сетевого приложения с поддержкой TCP-соединений и встроенной защитой от атак, связанных с установкой множества незавершенных соединений. Исследование механизма трехстороннего рукопожатия TCP позволило лучше понять процесс установления соединения и его потенциальные уязвимости.

Для повышения безопасности была внедрена аутентификация клиентов, требующая ввода пароля перед началом обмена данными. Это позволило предотвратить несанкционированный доступ и ограничить взаимодействие только авторизованными пользователями. Дополнительно была реализована система хранения и передачи сообщений в хешированном виде, что позволило минимизировать риски утечки данных.

Благодаря механизму широковещательной рассылки сообщений сервер обеспечил эффективное взаимодействие между клиентами. Разграничение доступа позволило неаутентифицированным пользователям видеть только хеши сообщений, в то время как авторизованные клиенты могли получать полный текст переданных данных. Это повысило уровень защиты передаваемой информации и предотвратило возможные утечки.

Использование многопоточной обработки подключений обеспечило высокую производительность сервера, позволив ему одновременно обслуживать несколько клиентов. Организация локального хранилища хешированных сообщений обеспечила возможность восстановления переданных данных, что увеличило удобство работы с системой.

Для анализа сетевого взаимодействия был использован Wireshark, что позволило детально изучить обмен пакетами между клиентом и сервером, проверить корректность установления TCP-соединений и выявить потенциальные уязвимости. Это помогло оценить надежность разработанного решения и убедиться в правильности его функционирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Системы и сети передачи информации. Е. А. Черенцова [Электронный ресурс]. – Режим доступа: http://elib.rshu.ru/files\_books/pdf/img-504180741.pdf.

[2] Протокол TCP. Ю. А. Семенов [Электронный ресурс]. – Режим доступа: https://www.protokols.ru/WP/wp-content/uploads/1994/05/rfc1613.pdf.

[3] Wireshark Network Security. Piyush Verma [Электронный ресурс]. – Режим доступа: https://www.cyberthai.com/images/Book\_pdf/Wireshark%20net work%20Analysis.pdf253.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <unordered\_map>

#include <fstream>

#include <atomic>

#include <string>

#include <algorithm>

#include <sstream>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

#define SERVER\_PORT 5000

#define PASSWORD "2"

struct ClientInfo {

SOCKET socket;

int id;

string ip;

bool authenticated;

};

atomic<int> clientCounter(1);

vector<ClientInfo> clients;

mutex clients\_mutex, log\_mutex, messages\_mutex;

void LogMessage(const string& msg) {

lock\_guard<mutex> lock(log\_mutex);

cout << msg << endl;

}

void WriteToDatabase(size\_t hashValue, const string& originalMessage) {

lock\_guard<mutex> lock(messages\_mutex);

ofstream dbFile("hashes.txt", ios::app);

if (dbFile.is\_open()) {

dbFile << hashValue << ":" << originalMessage << "\n";

}

else {

LogMessage("Ошибка при открытии файла базы данных.");

}

}

string ReadFromDatabase(size\_t hashValue) {

lock\_guard<mutex> lock(messages\_mutex);

ifstream dbFile("hashes.txt");

string line;

while (getline(dbFile, line)) {

size\_t delimiterPos = line.find(':');

if (delimiterPos != string::npos) {

size\_t storedHash = stoull(line.substr(0, delimiterPos));

if (storedHash == hashValue) {

return line.substr(delimiterPos + 1);

}

}

}

return "";

}

void BroadcastMessage(const string& originalMsg, size\_t hashValue, int senderId) {

lock\_guard<mutex> lock(clients\_mutex);

for (const auto& client : clients) {

// Если клиент не тот, кто отправил сообщение

if (client.id != senderId) {

string msg;

if (client.authenticated) {

msg = "Клиент #" + to\_string(senderId) + ": " + originalMsg + "\n";

}

else {

msg = "Клиент #" + to\_string(senderId) + " [Хеш]: " + to\_string(hashValue) + "\n";

}

send(client.socket, msg.c\_str(), msg.size(), 0);

}

}

}

void HandleClient(SOCKET clientSocket, string clientIP, int clientId) {

char buffer[1024];

int bytesReceived;

{

lock\_guard<mutex> lock(clients\_mutex);

clients.push\_back({ clientSocket, clientId, clientIP, false });

}

LogMessage("Клиент #" + to\_string(clientId) + " (" + clientIP + ") подключился.");

send(clientSocket, "Введите пароль: ", 16, 0);

bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);

if (bytesReceived <= 0) {

closesocket(clientSocket);

return;

}

buffer[bytesReceived] = '\0';

string receivedPassword(buffer);

receivedPassword.erase(remove(receivedPassword.begin(), receivedPassword.end(), '\r'), receivedPassword.end());

receivedPassword.erase(remove(receivedPassword.begin(), receivedPassword.end(), '\n'), receivedPassword.end());

if (receivedPassword != PASSWORD) {

send(clientSocket, "Неверный пароль! Отключение.\n", 30, 0);

closesocket(clientSocket);

return;

}

{

lock\_guard<mutex> lock(clients\_mutex);

for (auto& client : clients) {

if (client.id == clientId) {

client.authenticated = true;

break;

}

}

}

send(clientSocket, "Доступ разрешен!\n", 18, 0);

LogMessage("Клиент #" + to\_string(clientId) + " прошел аутентификацию.");

while ((bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0)) > 0) {

buffer[bytesReceived] = '\0';

string message(buffer);

message.erase(remove(message.begin(), message.end(), '\r'), message.end());

message.erase(remove(message.begin(), message.end(), '\n'), message.end());

if (message.empty()) continue;

size\_t hashValue;

bool isNumeric = false;

try {

hashValue = stoull(message);

isNumeric = true;

}

catch (...) {

isNumeric = false;

}

if (isNumeric) {

bool isAuthenticated = false;

{

lock\_guard<mutex> lock(clients\_mutex);

for (const auto& client : clients) {

if (client.id == clientId) {

isAuthenticated = client.authenticated;

break;

}

}

}

if (!isAuthenticated) {

string response = "Ошибка: требуется аутентификация.\n";

send(clientSocket, response.c\_str(), response.size(), 0);

continue;

}

string originalMessage = ReadFromDatabase(hashValue);

string response;

if (!originalMessage.empty()) {

response = "[Расшифровано] " + originalMessage + "\n";

}

else {

response = "Неизвестный хеш.\n";

}

send(clientSocket, response.c\_str(), response.size(), 0);

}

else {

hash<string> hasher;

size\_t computedHash = hasher(message);

WriteToDatabase(computedHash, message);

string confirmation = "Сообщение принято. Хеш: " + to\_string(computedHash) + "\n";

send(clientSocket, confirmation.c\_str(), confirmation.size(), 0);

BroadcastMessage(message, computedHash, clientId);

}

}

{

lock\_guard<mutex> lock(clients\_mutex);

auto it = remove\_if(clients.begin(), clients.end(), [clientId](const ClientInfo& info) {

return info.id == clientId;

});

clients.erase(it, clients.end());

}

LogMessage("Клиент #" + to\_string(clientId) + " отключился.");

closesocket(clientSocket);

}

int main() {

setlocale(LC\_ALL, "Russian");

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

cerr << "Ошибка инициализации Winsock." << endl;

return 1;

}

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (serverSocket == INVALID\_SOCKET) {

cerr << "Ошибка создания сокета." << endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddr{};

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(SERVER\_PORT);

if (bind(serverSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

cerr << "Ошибка привязки сокета." << endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

cerr << "Ошибка прослушивания." << endl;

closesocket(serverSocket);

WSACleanup();

return 1;

}

LogMessage("Сервер запущен на порту " + to\_string(SERVER\_PORT) + "...");

vector<thread> clientThreads;

while (true) {

sockaddr\_in clientAddr;

int clientAddrSize = sizeof(clientAddr);

SOCKET clientSocket = accept(serverSocket, (sockaddr\*)&clientAddr, &clientAddrSize);

if (clientSocket == INVALID\_SOCKET) {

cerr << "Ошибка принятия подключения." << endl;

continue;

}

char clientIP[INET\_ADDRSTRLEN];

inet\_ntop(AF\_INET, &clientAddr.sin\_addr, clientIP, INET\_ADDRSTRLEN);

int clientId = clientCounter.fetch\_add(1);

clientThreads.emplace\_back(HandleClient, clientSocket, string(clientIP), clientId);

}

for (auto& t : clientThreads) {

if (t.joinable())

t.join();

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iostream>

#include <thread>

#include <string>

#include <algorithm>

#pragma comment(lib, "ws2\_32.lib")

using namespace std;

#define SERVER\_IP "127.0.0.1"

#define SERVER\_PORT 5000

void ReceiveMessages(SOCKET clientSocket) {

char buffer[1024];

int bytesRead;

while ((bytesRead = recv(clientSocket, buffer, sizeof(buffer) - 1, 0)) != SOCKET\_ERROR) {

if (bytesRead == 0) break; // Connection closed

buffer[bytesRead] = '\0';

string receivedMessage(buffer);

receivedMessage.erase(remove(receivedMessage.begin(), receivedMessage.end(), '\r'), receivedMessage.end());

cout << receivedMessage << endl;

}

}

void SendMessages(SOCKET clientSocket) {

string message;

while (true) {

getline(cin, message);

if (message.empty()) continue;

send(clientSocket, message.c\_str(), message.size(), 0);

}

}

int main() {

setlocale(LC\_ALL, "Russian");

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

cerr << "Ошибка инициализации Winsock." << endl;

return 1;

}

SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (clientSocket == INVALID\_SOCKET) {

cerr << "Ошибка создания сокета." << endl;

WSACleanup();

return 1;

}

sockaddr\_in serverAddr{};

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(SERVER\_PORT);

inet\_pton(AF\_INET, SERVER\_IP, &serverAddr.sin\_addr);

if (connect(clientSocket, (sockaddr\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

cerr << "Не удалось подключиться к серверу!" << endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

cout << "Подключено к серверу." << endl;

thread receiveThread(ReceiveMessages, clientSocket);

receiveThread.detach();

thread sendThread(SendMessages, clientSocket sendThread.detach();

while (true) {

}

closesocket(clientSocket);

WSACleanup();

return 0;

}