**Оглавление**

[**Оглавление** 5](#_Toc515366257)

[**Введение** 6](#_Toc515366258)

[**1 Обзор состояния вопроса** 7](#_Toc515366259)

[**1.1 Технология сокетов** 7](#_Toc515366260)

**1.2** **Протокол TCP** 8

## **1****.3 Строгая** **аутентификация**9

[**2 Постановка задачи** 10](#_Toc515366262)

[**3 Моделирование и реализация ПО** 11](#_Toc515366263)

[**3.1 Архитектура ПО и описание протокола взаимодействия клиента и сервера** 11](#_Toc515366264)

[**3.2** **Серверная часть** 11](#_Toc515366265)

[**3.3 Клиентская часть** 11](#_Toc515366266)

[**4 Руководство пользователя** 11](#_Toc515366267)

[**4.1 Руководство пользователя сервера** 12](#_Toc515366268)

[**4.2 Руководство пользователя клиента** 12](#_Toc515366269)

[**Заключение** 16](#_Toc515366270)

[**Список использованной литературы** 17](#_Toc515366271)

[**Приложение А** 18](#_Toc515366272)

[**Приложение Б** 19](#_Toc515366273)

# **Введение**

Аутентификация наряду с авторизацией представляет собой фундаментальный атрибут информационной безопасности.

Термин «аутентификация» (authentication) происходит от латинского слова authenticus, которое означает подлинный, достоверный, соответствующий самому себе. Аутентификация, или, другими словами, процедура установления подлинности, может быть применима как к людям, так и другим объектам, в частности к программам, устройствам, документам.

В частности, при выполнении логического входа в защищенную систему пользователь должен пройти процедуру аутентификации, то есть доказать, что именно ему принадлежит введенный им идентификатор (имя пользователя). Аутентификация предотвращает доступ к сети нежелательных лиц и разрешает вход для легальных пользователей.

Цель курсовой работы — закрепить, систематизировать и комплексно обобщить знания пo предмету «Компьютерные системы и сети»; нaучитьcя практически применять полученные теоретические знания при решении конкретных задач. Реализовать сетевое приложение строгой аутентификации пользователей в компьютерной сети на основе многоразовых паролей.

# **1 Обзор состояния вопроса**

## **1.1 Технология сокетов**

Сокеты - название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одном компьютере, так и на различных компьютерах, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.

Чтобы две программы могли общаться друг с другом через сеть, каждая из них должна создать сокет. Каждый сокет обладает двумя основными характеристиками: протоколом и адресом, к которым он привязан. Протокол задаётся при создании сокета и не может быть изменён впоследствии. Адрес сокета задаётся позже, но обязательно до того, как через сокет пойдут данные. Формат адреса сокета определяется конкретным протоколом. В частности, для протоколов TCP и UDP адрес состоит из IP-адреса сетевого интерфейса и номера порта.

Протокол IP используется для ненадёжной доставки данных от одного узла сети к другому. Это означает, что на уровне этого протокола (третий уровень сетевой модели OSI) не даётся гарантий надёжной доставки пакета до адресата. В частности, пакеты могут прийти не в том порядке, в котором были отправлены, оказаться повреждёнными или не прибыть вовсе. Гарантии безошибочной доставки пакетов дают протоколы более высокого (транспортного) уровня сетевой модели OSI — например, TCP — которые используют IP в качестве транспорта.

В протоколе TCP также, как и в UDP, для связи с прикладными процессами используются порты. Номера портам присваиваются аналогичным образом: имеются стандартные, зарезервированные номера (например, номер 21 закреплен за сервисом FTP, 23 - за telnet), а менее известные приложения пользуются произвольно выбранными локальными номерами.

Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы. Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т.д.

Основные функции:

1. общие:
   1. socket - создать новый сокет и вернуть файловый дескриптор;
   2. send - отправить данные по сети;
   3. receive - получить данные из сети;
   4. close - закрыть соединение.
2. серверные:
   1. bind - связать сокет с IP-адресом и портом;
   2. listen - объявить о желании принимать соединения. Слушает порт и ждет когда будет установлено соединение;
   3. acept - принять запрос на установку соединения.
3. клиентские:
   1. connect - установить соединение.

## **1.2 Протокол TCP**

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных. Сети и подсети, в которых совместно используются протоколы TCP и IP, называются сетями TCP/IP.

В стеке протоколов IP, TCP выполняет функции протокола транспортного уровня модели OSI.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от UDP, целостность передаваемых данных и уведомление отправителя о результатах передачи.

Реализации TCP обычно встроены в ядра ОС. Существуют реализации TCP, работающие в пространстве пользователя.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. TCP осуществляет надёжную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере (например, программы для электронной почты, для обмена файлами). TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.

Процесс начала сеанса TCP (также называемый «рукопожатие» (англ. handshake)), состоит из трёх шагов:

1. Клиент, который намеревается установить соединение, посылает серверу сегмент с номером последовательности и флагом SYN.

Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.

В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние SYN-RECEIVED.

В случае неудачи сервер посылает клиенту сегмент с флагом RST.

2. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.

Если клиент одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED.

Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.

Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.

3. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.

В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

Завершение соединения можно рассмотреть в три этапа:

1. Посылка серверу от клиента флага FIN на завершение соединения.
2. Сервер посылает клиенту флаги ответа ACK , FIN, что соединение закрыто.
3. После получения этих флагов клиент закрывает соединение и в подтверждение отправляет серверу ACK , что соединение закрыто.

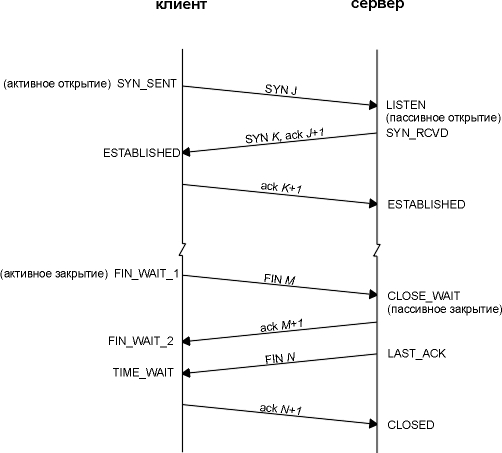


Рисунок 1.1 – Установление и закрытие TCP соединения

## **1.3 Строгая** **аутентификация**

Протокол аутентификации по квитированию вызова (Challenge Handshake Authentication Protocol, CHAP) входит в семейство протоколов PPR В этом протоколе предусмотрено 4 типа сообщений: Success (успех), Challenge (вызов), Response (ответ), Failure (ошибка).

Этот протокол используется, например, при аутентификации удаленных пользователей, подключенных к Интернету по коммутируемому каналу Здесь аутентификатором является сервер провайдера, а аутентифицируемым — клиентский компьютер (рис. 1). При заключении договора клиент получает от провайдера пароль (пусть, например, это будет сйово parol). Этот пароль хранится в базе данных провайдера в виде дайджеста Z = d(parol), полученного путем применения к паролю односторонней хэш-функции MD5.

Аутентификация выполняется в следующей последовательности.

1. Пользователь-клиент активизирует программу (например, программу дозвона) удаленного доступа к серверу провайдера, вводя имя и назначенный ему пароль. Имя (на рисунке это "Moscow") передается по сети провайдеру в составе запроса на соединение, но пароль не передается в сеть ни в каком виде. То есть здесь мы имеем дело со строгой аутентификацией.

2. Сервер провайдера, получив запрос от клиента, генерирует псевдослучайное слово-вызов (пусть это будет слово «challenge») и передает его клиенту вместе со значением, идентифицирующем сообщение в рамках данного сеанса (ID), и собственным именем (здесь "Paris"). Это сообщение типа Challenge. (Для защиты от перехвата ответа аутен-тификатор должен использовать разные значения слова-вызова при каждой процедуре аутентификации.)

3. Программа клиента, получив этот пакет, извлекает из него слово-вызов, добавляет к нему идентификатор и вычисленный локально дайджест Z = d(parol), а затем вычисляет с помощью все той же функции MD5 дайджест Y = d{(ID, challenge, J(parol)} от всех этих трех значений. Результат клиент посылает серверу провайдера в пакете Response.

4. Сервер провайдера сравнивает полученный по сети дайджест Ус тем значением, которое он получил, локально применив ту же хэш-функцию к набору аналогичных компонентов, хранящихся в его памяти.

5. Если результаты совпадают, то аутентификация считается успешной и аутентификатор посылает партнеру пакет Success.

## 

## Рис 1.2 – Пример реализации строгой аутентификации

# **2 Постановка задачи**

Разработать сетевое приложение строгой аутентификации пользователей в компьютерной сети на основе многоразовых паролей. Клиент должен иметь графический интерфейс. Используется протокол TCP. Зарегистрированные пользователи должны храниться в файле в виде «login password», где password обязан храниться в хешированном виде. В главном окне предусмотреть поля для ввода login и password и кнопки регистрации, авторизации и выхода. При неправильном вводе должны появляться соответствующие графические окна. Для разработки приложения использовать интегрированную среду разработки Microsoft Visual Studio на платформе .NET.

Приложение должно обладать следующими возможностями:

* установление соединения;
* аутентификация пользователей

Исходные данные проекта:

* OC Windows;
* спецификация сокетов в OC Windows;
* интегрированная среда разработки MS Visuаl Studio;
* язык программирования C#.

# **3 Моделирование и реализация ПО**

## **3.1 Архитектура ПО и описание протокола взаимодействия клиента и сервера**

Связь между клиентом и сервером осуществляется при помощи технологии сокетов. Клиент подключается к серверу, после чего осуществляется диалог клиента и сервера. Клиент отправляет серверу сумму хэш-ID, хэш-ключевого слова, хэш-пароля.

Графическое представление алгоритмов хэш-функций приведено в Приложении А. Исходный код программ приведен в Приложении Б.

## **Серверная часть**

Сервер реализован в виде консольного приложения и реализован в одном классе.

## **3.3 Клиентская часть**

Клиент реализован в виде WPF-приложения с понятным графическим интерфейсом.

При запуске клиента открывается главная форма приложения с двумя полями для ввода и тремя кнопками.

UML диаграмма ключевых классов представлена на рис. 3.3.

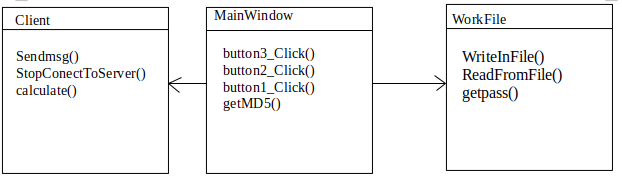


Рисунок 3.3 – UML диаграмма классов клиентского приложения

# **4 Руководство пользователя**

Для запуска приложения необходимо иметь на компьютере ОС Windows версии 7 и выше, а также .Net Framework версии 4.5.2 и выше.

## **4.1 Руководство пользователя сервера**

При запуске программы Serv0.exe открывается командная строка (рис. 4.1).

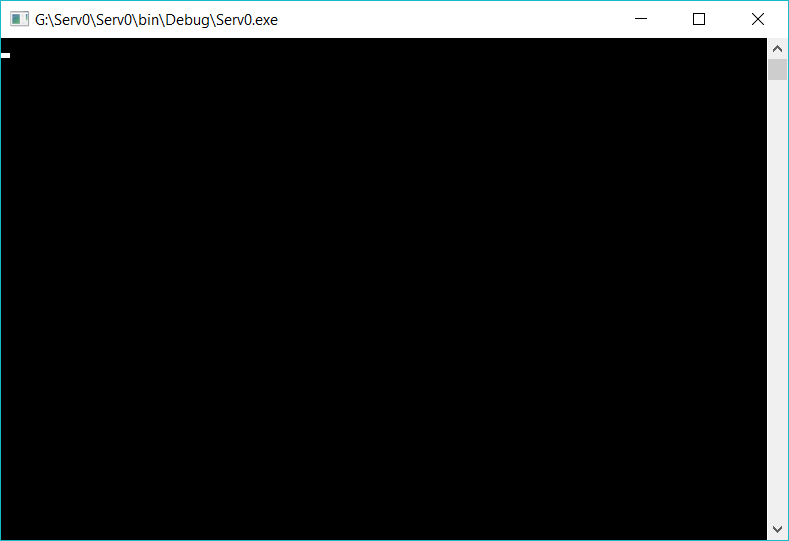


Рисунок 4.1 – Вид окна сервера

## **4.2 Руководство пользователя клиента**

При запуске программы Client.exe открывается окно программы (рис. 4.2).

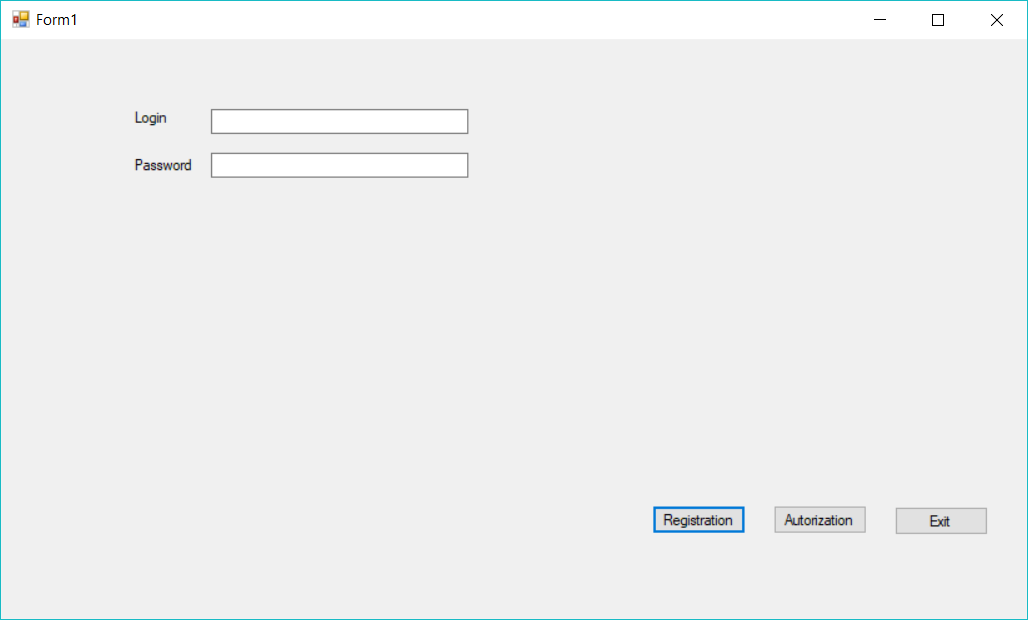


Рисунок 4.2 – Вид окна клиента

Если перед нажатием на кнопки регистрации или авторизации сервер не был запущен , то открывается другое окно (рис. 4.3).



Рисунок 4.3 – Ошибка подключения

При правильном вводе данных в полях login и password вы можете зарегистрироваться(для этого необходимо нажать кнопку Registration(рис. 4.4)) или авторизироваться(для этого необходимо нажать кнопку Autorization(рис. 4.5)), если такой пользователь уже зарегистрирован.

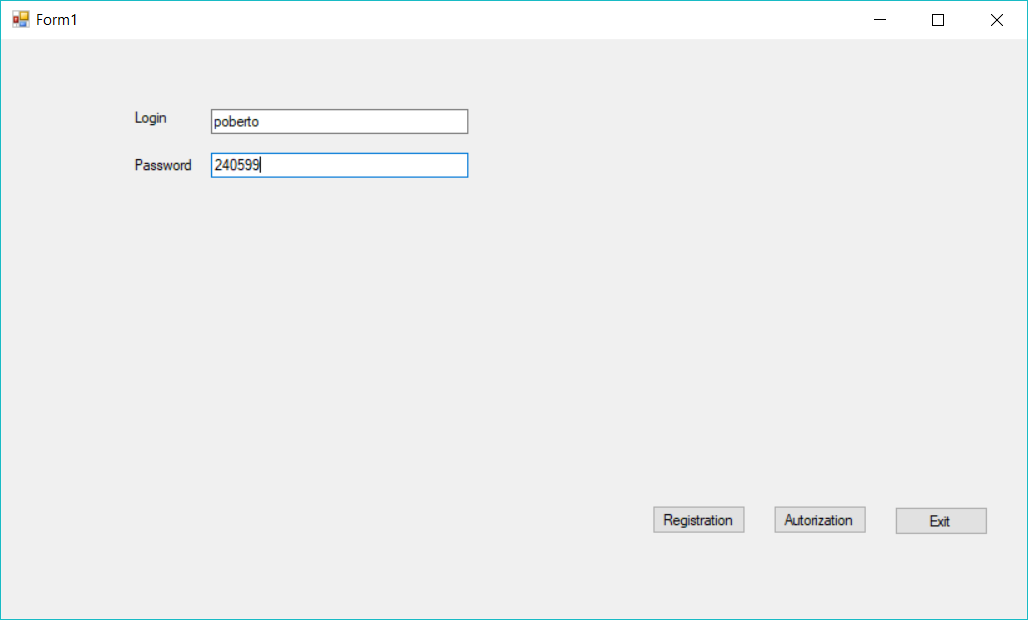


Рисунок 4.4 – Пример регистрации

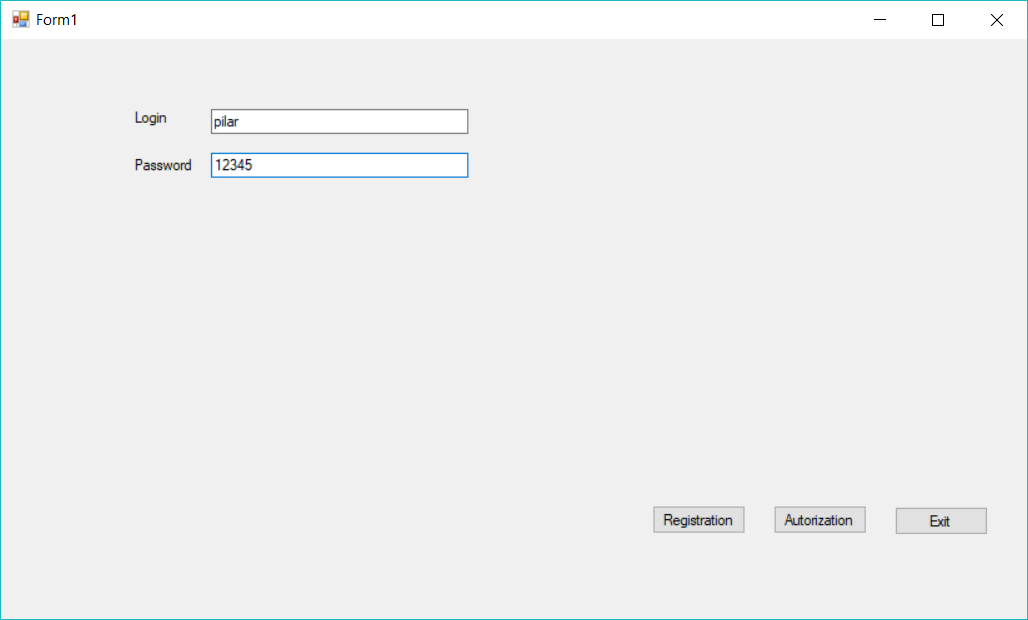


Рисунок 4.5 – Пример аворизации

Если пользователь правильно авторизировался, он может наблюдать следующее окно со стороны пользователя(рис. 4.6), а на стороне клиента в консоле выводится(рис. 4.7).

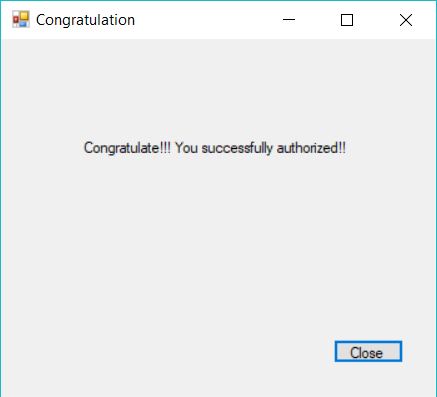


Рисунок 4.6 – Окно сохранения файла

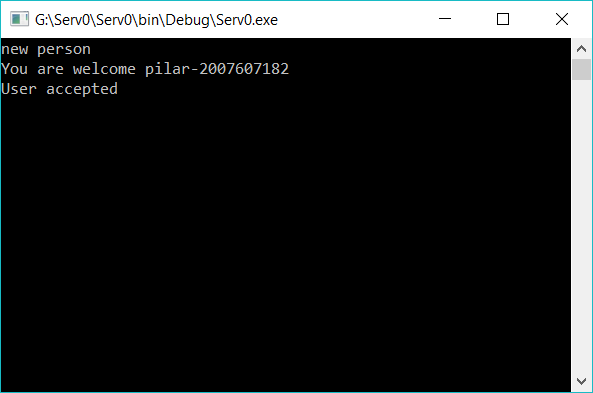


Рисунок 4.7 – Окно со стороны сервера

По завершению работы пользователь может нажать на кнопку Exit или красный крестик в правом верхнем углу(рис. 4.8).

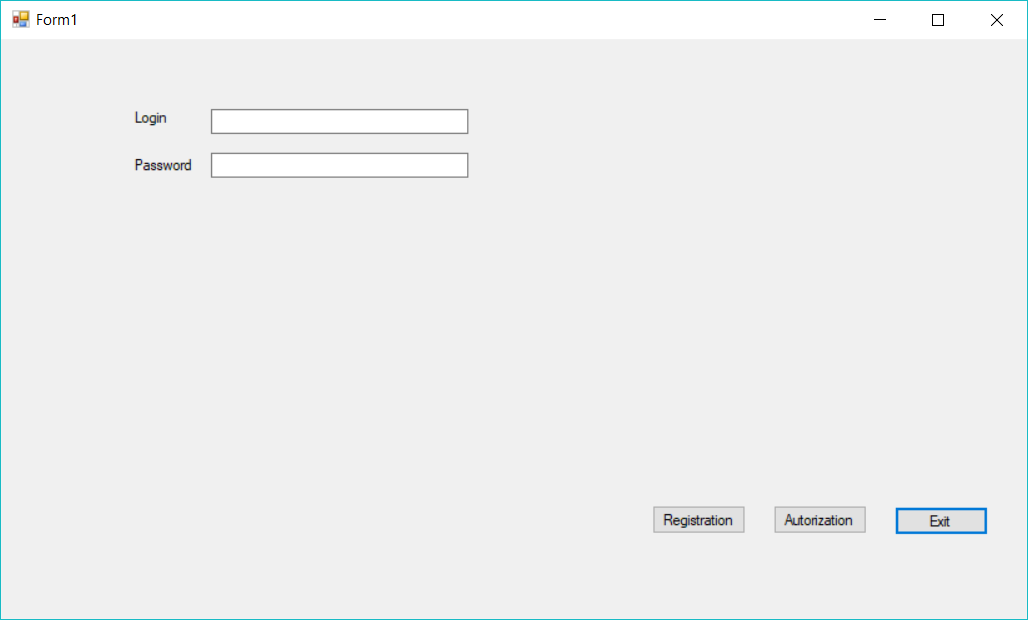


Рисунок 4.8 – Пример выходачерез кнопку Exit

# **Заключение**

В данном проекте было разработана программа, реализующая наглядный пример строгой аутентификации на основе многоразовых паролей. Для работы с пользователем было разработан графический интерфейс.

Программа имеет достаточно ясный и понятый дружелюбный пользовательский интерфейс, обеспечивающий удобство в работе.

Как показало тестирование, приложение работает корректно, и является законченным программным продуктом, поэтому поставленную задачу можно считать выполненной.

# **Список использованной литературы**

1. Э. Таненбаум, “Компьютерные сети” 5-е издание, изд. Питер, 2014 г.
2. Э. Кровчик, В. Кумар, «.Net. Сетевое программирование для профессионалов», изд. Лори, 2005 г.
3. В.Г. Олифер, Н.А. Олифер, «Компьютерные сети. Принципы, технологии, протоколы», изд. Питер, 2006г.
4. Эндрю Троелсен, "Язык программирования C# 5.0 и платформа .NET 4.5", 6-е издание, изд. Вильямс, 2012.

# **Приложение А**

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—10701216 02-2018-01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | Тырля |  |  | | Строгая аутентификация  на основе многораозвых паролей | Лист | | | Лист | Листов |
| Руковод. | Белова |  |  | |  | Д |  | **1** | **3** |
| Консульт. | Белова |  |  | | 1-40-01-01 БНТУ  г.Минск | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Виктор\Desktop\MD5.png | | | | | | | | | | |
|  |  |  |  |  | КП—10701216 02-2018-01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | Тырля |  |  | | Алгоритм MD5 | Лист | | | Лист | Листов |
| Руковод. | Белова |  |  | |  | Д |  | **2** | **3** |
| Консульт. | Белова |  |  | | 1-40-01-01 БНТУ  г.Минск | | | | |
| Н.контр. | Белова |  |  | |
| Зав.каф. | Полозков |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C:\Users\Виктор\Desktop\188.gif | | | | | | | | | | |
|  |  |  |  |  | КП—10701216 02-2018-01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | Тырля |  |  | | Tcp соединение | Лист | | | Лист | Листов |
| Руковод. | Белова |  |  | |  | Д |  | **3** | **3** |
| Консульт. | Белова |  |  | | 1-40-01-01 БНТУ  г.Минск | | | | |

# **Приложение Б**

**Server**

using System;

using System.Collections;

using System.IO;

using System.Net.Sockets;

using System.Net;

using System.Text;

namespace Serv0 {

class Program {

public static void Main(string[] args) {

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 11000);

Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

Try {

sListener.Bind(ipEndPoint);

sListener.Listen(5);

while (true) {

Socket client = sListener.Accept();

String username = null;

String hash = null;

Console.WriteLine("new person");

byte[] buffer = new byte[256];

client.Receive(buffer);

username += Encoding.ASCII.GetString(buffer);

buffer = new byte[256];

Console.WriteLine("You are welcome " + username);

hash = Hash(username);

ArrayList list = new ArrayList();

list = GetStringFromFile(@"G:\Serv0\Serv0\bin\Debug\text.txt");

string user = User(username);

string password = Parser(list, user);

if (AcceptUser(Convert.ToInt32(hash), DateTime.DaysInMonth(2018, 5), "word", password)) {

Console.WriteLine("User accepted");

}

}

}

catch (Exception e){

Console.WriteLine(e);

sListener.Close();

}

finally{

Console.ReadLine();

}

private static string User(string userAndHash){

char[] mas = userAndHash.ToCharArray();

int index = 0;

StringBuilder user = new StringBuilder();

for (int i = 0; i < mas.Length; i++) {

if (mas[i] == '-') {

index = i;

}

}

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

user.Append(mas[i]);

}

return user.ToString();

}

private static string Hash(string userAndHash) {

char[] mas = userAndHash.ToCharArray();

int index = 0;

StringBuilder hash = new StringBuilder();

for(int i = 0; i < mas.Length; i ++) {

if (mas[i] == '-') {

index = i;

}

}

for( int i = index + 1; i < mas.Length; i++) {

hash.Append(mas[i]);

}

return hash.ToString();

}

private static string Parser(ArrayList list, string username) {

foreach (string line in list) {

var strings = line.Split(' ');

if (strings[0].Equals(username)) {

return strings[1];

}

}

return null;

}

private static ArrayList GetStringFromFile(string filename) {

StreamReader sr = new StreamReader(filename);

ArrayList text = new ArrayList();

string temp = String.Empty;

while (temp != null) {

temp = sr.ReadLine();

text.Add(temp);

}

return text;

}

private static bool AcceptUser(int hash, int id, string word, string password) {

int buf = id.GetHashCode() + word.GetHashCode() + password.GetHashCode();

if (hash == buf) {

return true;

}

return false;

}

}

}

**Client.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

namespace Client {

class Client {

static IPHostEntry ipHost = Dns.GetHostEntry("localhost");

static IPAddress ipAddr = ipHost.AddressList[0];

static IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 11000);

Socket socket = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

public Client() { }

public bool sendmsg(string username, string hash) {

try {

socket.Connect(ipEndPoint);

string message = username + "-";

byte[] buffer = System.Text.Encoding.ASCII.GetBytes(message);

int bytesSend = socket.Send(buffer);

message = hash;

buffer = System.Text.Encoding.ASCII.GetBytes(message);

bytesSend = socket.Send(buffer);

Console.ReadLine();

return true;

} catch (SocketException e) {

ServerIsNotStart serverIsNotStart = new ServerIsNotStart();

serverIsNotStart.Show();

return false;

}

}

public void stopConectToServer() {

socket.Shutdown(SocketShutdown.Both);

socket.Close();

}

public int calculate(int id, string word, string password) {

return id.GetHashCode() + word.GetHashCode() + password.GetHashCode();

}

}

}

**Form.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Security.Cryptography;

using System.Text;

namespace Client{

public partial class Form1 : Form {

public Form1(){

InitializeComponent();

}

private void button3\_Click(object sender, EventArgs e){ //Close

Application.Exit();

}

private void button2\_Click(object sender, EventArgs e){ //autorization

if (textBox1.Text == "" || textBox2.Text == ""){

EmptyRegistration reg = new EmptyRegistration();

reg.Show();

}

else if (WorkFile.readFromFile(textBox1.Text)){

Client client = new Client();

string a = (DateTime.DaysInMonth(2018, 5).GetHashCode() + "word".GetHashCode() + getMD5(textBox2.Text).GetHashCode()) + "";

bool send = client.sendmsg(textBox1.Text, a);

if (send){

Congratulation congrat = new Congratulation();

congrat.Show();

}

} else{

NotRegistrated notRegistrated = new NotRegistrated();

notRegistrated.Show();

}

textBox1.Text = "";

textBox2.Text = "";

}

private void button1\_Click(object sender, EventArgs e){ //registration

if (textBox1.Text == "" || textBox2.Text == ""){

EmptyRegistration reg = new EmptyRegistration();

reg.Show();

}

else if (WorkFile.readFromFile(textBox1.Text)){

NotCurrentLogin notCurrentLogin = new NotCurrentLogin();

notCurrentLogin.Show();

}

else{

string login = textBox1.Text;

string password = getMD5(textBox2.Text);

WorkFile.writeInFile(login, password);

}

textBox1.Text = "";

textBox2.Text = "";

}

private string getMD5(string text){

MD5 md = new MD5CryptoServiceProvider();

byte[] result = md.ComputeHash(Encoding.UTF8.GetBytes(text));

StringBuilder sb = new StringBuilder();

foreach (byte b in result){

sb.Append(b.ToString("x2"));

}

return sb.ToString();

}

}

}

**WorkFile.cs**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Client{

class WorkFile{

public WorkFile(){}

public static void writeInFile(string login, string password){

using (StreamWriter sw = new StreamWriter(@"G:\Serv0\Serv0\bin\Debug\text.txt", true, System.Text.Encoding.Default)){

sw.WriteLine(login + " " + password);

}

}

public static bool readFromFile(string login){

using (StreamReader sr = new StreamReader(@"G:\Serv0\Serv0\bin\Debug\text.txt", System.Text.Encoding.Default)){

string line = null;

while ((line = sr.ReadLine()) != null){

if (!string.IsNullOrEmpty(line.Trim())){

string firstLineWord = line.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0];

if (firstLineWord.Equals(login)){

return true;

}

}

}

}

return false;

}

public static string getpass(string login){

string pass = " ";

using (StreamReader sr = new StreamReader("test.txt", System.Text.Encoding.Default)){

string line = null;

while ((line = sr.ReadLine()) != null){

string firstLineWord = line.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)[0];

if (firstLineWord.Equals(login)){

if (!string.IsNullOrEmpty(line.Trim())){

string[] mas = line.Split(new char[] { ' ' });

pass = mas[1];

}

}

}

}

return pass;

}

}

}