**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

по дисциплине «Программирование сетевых приложений»

на тему:

**«**Программирование безопасного клиент-серверного

взаимодействия с использованием HTTPS**»**

Выполнил: студент гр. ИТИ-41

Гончар А.С.

Принял: преподаватель

Карабчикова Е.А.

Гомель 2020

**Цель работы:** научиться разрабатывать приложения с использованием пула потоков.

**Задание**: Разработать https­сервер ( использовать пул потоков для параллельной обработки запросов клиентов). В качестве клиентского приложения используется веб­браузер. Все заголовки формируются вручную. Браузер должен правильно отображать получаемую информацию. Рекомендуется на GET­запрос возвращать html страницу с формой ввода, и выполнять POST­запрос, на который сервер присылает решение.

Разработать конечный автомат для анализа надежности и корректности пароля.

Длина пароля не менее 6 символов. Разрешены только латинские буквы, цифры и знаки: пробел \_­. В пароле обязательно должны присутствовать знаки в верхнем и нижнем регистре и цифры.

Пользователю возвращается информация о том, можно ли использовать указанный пароль.



Рисунок 1 – Клиентское приложение



Рисунок 2 – Результат выполнения работы

**Вывод:** в ходе выполнения лабораторной работы был разработан безопасный https­сервер с использованием пула потоков.

**Приложение А**

**Листинг программы**

Client:

using System;

using System.IO;

using System.Net;

using System.Net.Security;

using System.Net.Sockets;

using System.Security.Authentication;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Text.RegularExpressions;

namespace Lab6PSP

{

// Класс-обработчик клиента

class Client

{

// Отправка страницы с ошибкой

private void SendError(TcpClient Client, int Code)

{

// Получаем строку вида "200 OK"

// HttpStatusCode хранит в себе все статус-коды HTTP/1.1

string CodeStr = Code.ToString() + " " + ((HttpStatusCode)Code).ToString();

// Код простой HTML-странички

string Html = "<html><body><h1>" + CodeStr + "</h1></body></html>";

// Необходимые заголовки: ответ сервера, тип и длина содержимого. После двух пустых строк - само содержимое

string Str = "HTTP/1.1 " + CodeStr + "\nContent-type: text/html\nContent-Length:" + Html.Length.ToString() + "\n\n" + Html;

// Приведем строку к виду массива байт

byte[] Buffer = Encoding.ASCII.GetBytes(Str);

// Отправим его клиенту

Client.GetStream().Write(Buffer, 0, Buffer.Length);

// Закроем соединение

Client.Close();

}

// Конструктор класса. Ему нужно передавать принятого клиента от TcpListener

public Client(TcpClient Client, X509Certificate serverCertificate)

{

SslStream sslStream = new SslStream(Client.GetStream(), false);

try

{

sslStream.AuthenticateAsServer(serverCertificate, false, SslProtocols.Tls12, true);

// Объявим строку, в которой будет хранится запрос клиента

string Request = "";

// Буфер для хранения принятых от клиента данных

byte[] Buffer = new byte[1024];

// Переменная для хранения количества байт, принятых от клиента

int Count;

// Читаем из потока клиента до тех пор, пока от него поступают данные

while ((Count = sslStream.Read(Buffer, 0, Buffer.Length)) > 0)

{

// Преобразуем эти данные в строку и добавим ее к переменной Request

Request += Encoding.ASCII.GetString(Buffer, 0, Count);

// Запрос должен обрываться последовательностью \r\n\r\n

// Либо обрываем прием данных сами, если длина строки Request превышает 4 килобайта

// Нам не нужно получать данные из POST-запроса (и т. п.), а обычный запрос

// по идее не должен быть больше 4 килобайт

if (Request.IndexOf("\r\n\r\n") >= 0 || Request.Length > 4096)

{

break;

}

}

// Парсим строку запроса с использованием регулярных выражений

// При этом отсекаем все переменные GET-запроса

Match ReqMatch = Regex.Match(Request, @"^\w+\s+([^\s]+)[^\s]\*\s+HTTP/.\*|");

// Если запрос не удался

if (ReqMatch == Match.Empty)

{

// Передаем клиенту ошибку 400 - неверный запрос

SendError(Client, 400);

return;

}

// Получаем строку запроса

string RequestUri = ReqMatch.Groups[1].Value;

// Приводим ее к изначальному виду, преобразуя экранированные символы

// Например, "%20" -> " "

RequestUri = Uri.UnescapeDataString(RequestUri);

// Если в строке содержится двоеточие, передадим ошибку 400

// Это нужно для защиты от URL типа http://example.com/../../file.txt

if (RequestUri.IndexOf("..") >= 0)

{

SendError(Client, 400);

return;

}

string Headers;

byte[] HeadersBuffer;

// Если строка запроса оканчивается на "/", то добавим к ней index.html

if (RequestUri.EndsWith("/") ||

RequestUri.EndsWith(".css") ||

RequestUri.EndsWith(".html") ||

RequestUri.EndsWith(".js") ||

RequestUri.EndsWith(".ico") ||

RequestUri.EndsWith(".htm"))

{

if (RequestUri.EndsWith("/"))

{

RequestUri += "index.html";

}

string FilePath = "Resources/" + RequestUri;

// Если в папке www не существует данного файла, посылаем ошибку 404

if (!File.Exists(FilePath))

{

SendError(Client, 404);

return;

}

// Получаем расширение файла из строки запроса

string Extension = RequestUri.Substring(RequestUri.LastIndexOf('.'));

// Тип содержимого

string ContentType = "";

// Пытаемся определить тип содержимого по расширению файла

switch (Extension)

{

case ".htm":

case ".html":

ContentType = "text/html";

break;

case ".css":

ContentType = "text/stylesheet";

break;

case ".js":

ContentType = "text/javascript";

break;

case ".jpg":

ContentType = "image/jpeg";

break;

case ".jpeg":

case ".png":

case ".gif":

ContentType = "image/" + Extension.Substring(1);

break;

default:

if (Extension.Length > 1)

{

ContentType = "application/" + Extension.Substring(1);

}

else

{

ContentType = "application/unknown";

}

break;

}

// Открываем файл, страхуясь на случай ошибки

FileStream FS;

try

{

FS = new FileStream(FilePath, FileMode.Open, FileAccess.Read, FileShare.Read);

}

catch (Exception)

{

// Если случилась ошибка, посылаем клиенту ошибку 500

SendError(Client, 500);

return;

}

// Посылаем заголовки

Headers = "HTTP/1.1 200 OK\nContent-Type: " + ContentType + "\nContent-Length: " + FS.Length + "\n\n";

HeadersBuffer = Encoding.ASCII.GetBytes(Headers);

sslStream.Write(HeadersBuffer, 0, HeadersBuffer.Length);

// Пока не достигнут конец файла

while (FS.Position < FS.Length)

{

// Читаем данные из файла

Count = FS.Read(Buffer, 0, Buffer.Length);

// И передаем их клиенту

sslStream.Write(Buffer, 0, Count);

}

// Закроем файл и соединение

FS.Close();

Client.Close();

}

else if (RequestUri.Length > 2)

{

string password = RequestUri.Remove(0, 1);

Console.WriteLine(password);

bool correct = false;

var regexstr = @"^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9])(?=.\*?[-\_]).{6,}$";

Regex regex = new Regex(regexstr);

if (regex.IsMatch(password))

{

if (!Regex.Match(password, @"[а-яА-Я]").Success && !Regex.Match(password, @"[/\*+=!@#$%^,.<>;:()&|\`~{}]").Success)

{

correct = true;

}

}

var correctStr = correct ? "Correct" : "Incorrect";

byte[] message = Encoding.ASCII.GetBytes(correctStr);

Headers = "HTTP/1.1 200 OK\nContent-Type: " + "text" + "\nContent-Length: " + message.Length + "\n\n";

HeadersBuffer = Encoding.ASCII.GetBytes(Headers);

sslStream.Write(HeadersBuffer, 0, HeadersBuffer.Length);

sslStream.Write(message, 0, message.Length);

}

}

catch(Exception exp)

{

Console.WriteLine(exp.Message);

}

finally

{

sslStream.Close();

Client.Close();

}

}

}

}

Server:

using System;

using System.Net;

using System.Net.Sockets;

using System.Security.Cryptography.X509Certificates;

using System.Threading;

namespace Lab6PSP

{

class Server

{

TcpListener Listener; // Объект, принимающий TCP-клиентов

static X509Certificate2 serverCertificate=null;

// Запуск сервера

public Server(int Port)

{

string s = IPAddress.Any.ToString();

serverCertificate = new X509Certificate2("server.cer", "1");

Listener = new TcpListener(IPAddress.Any, Port); // Создаем "слушателя" для указанного порта

Listener.Start(); // Запускаем его

Console.WriteLine("Server is running. https://127.0.0.1:80");

// В бесконечном цикле

while (true)

{

// Принимаем новых клиентов. После того, как клиент был принят, он передается в новый поток (ClientThread)

// с использованием пула потоков.

ThreadPool.QueueUserWorkItem(new WaitCallback(ClientThread), Listener.AcceptTcpClient());

}

}

static void ClientThread(Object StateInfo)

{

// Просто создаем новый экземпляр класса Client и передаем ему приведенный к классу TcpClient объект StateInfo

new Client((TcpClient)StateInfo, serverCertificate);

}

// Остановка сервера

~Server()

{

// Если "слушатель" был создан

if (Listener != null)

{

// Остановим его

Listener.Stop();

}

}

}

}

Program:

using System;

using System.Threading;

namespace Lab5PSP

{

class Program

{

static void Main(string[] args)

{

// Определим нужное максимальное количество потоков

// Пусть будет по 4 на каждый процессор

int MaxThreadsCount = Environment.ProcessorCount \* 4;

// Установим максимальное количество рабочих потоков

ThreadPool.SetMaxThreads(MaxThreadsCount, MaxThreadsCount);

// Установим минимальное количество рабочих потоков

ThreadPool.SetMinThreads(2, 2);

new Server(80);

}

}

}