МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №6

по дисциплине: **«**Программирование сетевых приложений**»**

на тему: «Программирование безопасного клиент­серверного

взаимодействия с использованием *HTTPS*»

Выполнил: студент гр. ИТП-41

Дорошко Д. А.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2022

**Цель:** изучить организацию параллельной обработки *htpp*-запросов клиентов с использованием пула потоков.

**Задание:**

Для задания представленном ниже для коммуникации использовать HTTPS протокол (генерировать тестовый сертификат с использованием OpenSSL, можно использовать сертификаты генерируемые инструментами

Windows). ( Вся работа должна быть выполнена с использованием стандартных сокетов, все HTTPS заголовки формируются вручную) 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.

Задание согласно варианту: *MegreSort*

**Ход работы:**

Создание сертификата представлено на рисунке 1.

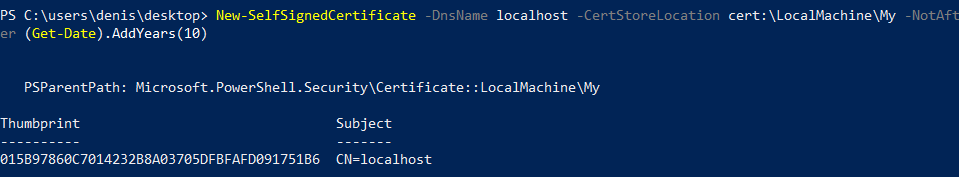


Рисунок 1 – Главная страница

На рисунке 2 представлен биндинг сертификата на заданный адрес.



Рисунок 2 – Биндинг сертификата на заданный адрес

Сертификат представлен на рисунке 3.

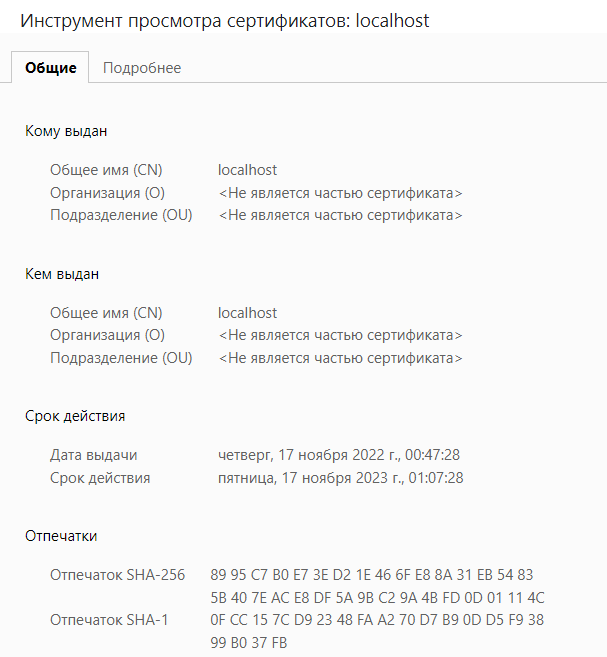


Рисунок 3 – Отображение информации о сертификате

Форма ввода массива представлена на рисунке 4.

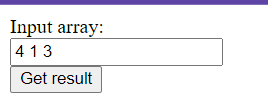


Рисунок 4 – Форма ввода массива

**Вывод:** в ходе выполнения лабораторной работы получены создания сертификатов и использования протокола *https*.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

using System.Security.Cryptography.X509Certificates;

using System.Security.Cryptography;

using System.Net;

using System.Text;

using Server.Sorting;

using System.Text.RegularExpressions;

using System.IO;

using System.Linq;

using System.Net.Sockets;

namespace Server

{

internal class Program

{

static async Task Main(string[] args)

{

HttpListener server = new HttpListener();

// установка адресов прослушки

server.Prefixes.Add("https://127.0.0.1:5006/");

server.Start(); // начинаем прослушивать входящие подключения

while (true)

{

var context = await server.GetContextAsync();

ThreadPool.QueueUserWorkItem(async

state =>

{

await ProcessClient(state as HttpListenerContext);

}, context);

}

}

private static async Task ProcessClient(HttpListenerContext context)

{

var request = context.Request;

var response = context.Response;

// отправляемый в ответ код htmlвозвращает

var responseText = GetResponseMessage(request, response);

byte[] buffer = Encoding.UTF8.GetBytes(responseText);

// получаем поток ответа и пишем в него ответ

response.ContentLength64 = buffer.Length;

using Stream output = response.OutputStream;

// отправляем данные

await output.WriteAsync(buffer);

await output.FlushAsync();

}

private static string GetResponseMessage(HttpListenerRequest request, HttpListenerResponse response)

{

var encoding = request.ContentEncoding;

var reader = new StreamReader(request.InputStream, encoding);

string body = reader.ReadToEnd();

if (request.Url.PathAndQuery.Equals("/"))

{

return request.HttpMethod switch

{

"GET" => GetMainPage(response),

\_ => GetNotFoundPage(response),

};

}

else if (request.Url.PathAndQuery.Equals("/sort/mergeSort.txt", StringComparison.InvariantCultureIgnoreCase))

{

return request.HttpMethod switch

{

"GET" => GetMergeSortPage(response),

"POST" => GetMergeSortResponsePage(response, body),

\_ => GetNotFoundPage(response),

};

}

return GetNotFoundPage(response);

}

private static string GetMainPage(HttpListenerResponse response)

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append("<a href='/sort/mergeSort.txt'>Merge sort</a>");

string body = bodyBuilder.ToString();

ResponseHeaders(response);

return body;

}

private static string GetNotFoundPage(HttpListenerResponse response)

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append("Page not found");

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<a href='/'>Go to main page</a>");

ResponseHeaders(response, responseStatus: 404);

return bodyBuilder.ToString();

}

private static string GetBadPage(HttpListenerResponse response, string message = "Something went wrong")

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append(message);

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<a href='/'>Go to main page</a>");

string body = bodyBuilder.ToString();

ResponseHeaders(response);

return body;

}

private static string GetMergeSortPage(HttpListenerResponse response)

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append("<form method='post'>");

bodyBuilder.Append("Input array:");

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<input type='text' name='array'>");

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<input type='submit' value='Get result'>");

bodyBuilder.Append("</form>");

string body = bodyBuilder.ToString();

ResponseHeaders(response);

return body;

}

private static string GetMergeSortResponsePage(HttpListenerResponse response, string body)

{

int[] array;

var stringArray = body.Split('=')[1].Split("+", StringSplitOptions.RemoveEmptyEntries);

StringBuilder bodyBuilder = new StringBuilder();

array = stringArray.Select(item => int.Parse(item)).ToArray();

var sorter = new MergeSortSolver();

var sortedArray = array.ToArray();

sorter.Sort(sortedArray);

bodyBuilder.Append("Array before sorting:");

bodyBuilder.Append("\n");

bodyBuilder.Append(string.Join(",", array));

bodyBuilder.Append("\n");

bodyBuilder.Append("Sorted array:");

bodyBuilder.Append("\n");

bodyBuilder.Append(string.Join(",", sortedArray));

ResponseHeaders(response, contentType: "application/octet-stream");

return bodyBuilder.ToString();

}

public static void ResponseHeaders(HttpListenerResponse response, string contentType = "text/html;charset=UTF-8", int responseStatus = 200)

{

response.Headers.Set("Content-Type", contentType);

response.Headers.Set("Connection", "Close");

response.Headers.Set("Server", "lab6-server");

response.StatusCode = responseStatus;

}

}

}