Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №3-4**

по дисциплине: «Сети и телекоммуникации».

Выполнил:

студент 4 курса, гр. ИВТАПбд-41

Кондратьев Павел Сергеевич.

Проверил:

преподаватель кафедры ВТ

Мартынов Антон Иванович.

г. Ульяновск, 2019

**1) Задание:**

* Разработать клиентское приложение осуществляющее взаимодействие при помощи механизма сокетов TCP/IP с сервером, который работает в локальной сети по определенному IP-адресу и порту. Формат команд, которые должен обрабатывать клиент определяется согласно варианту задания.
* Клиентская часть должна реализовывать функции, соответствующие номеру варианта. Обмен информацией с сервером осуществляется в текстовом режиме посредством определенных команд, каждая из которых выполняет определенное действие. Каждая команда сервера состоит из служебного слова и параметров. Сервер не должен быть чувствителен к регистру команд. Запрос должен завершаться символом конца строки (0x10) для обеспечения совместимости сервера со стандартными клиентами, такими как Telnet.
* Разработка клиента и сервера будет реализована на websockets js (по договоренности с преподавателем).

**2) Краткие теоретические сведения:**

Протокол WebSocket (стандарт RFC 6455) предназначен для решения любых задач и снятия ограничений обмена данными между браузером и сервером.

Он позволяет пересылать любые данные, на любой домен, безопасно и почти без лишнего сетевого трафика.

Протокол WebSocket работает над TCP.

Это означает, что при соединении браузер отправляет по HTTP специальные заголовки, спрашивая: «поддерживает ли сервер WebSocket?».

Если сервер в ответных заголовках отвечает «да, поддерживаю», то дальше HTTP прекращается и общение идёт на специальном протоколе WebSocket, который уже не имеет с HTTP ничего общего.

Соединение WebSocket можно открывать как WS:// или как WSS://. Протокол WSS представляет собой WebSocket над HTTPS.

Кроме большей безопасности, у WSS есть важное преимущество перед обычным WS – большая вероятность соединения.

Дело в том, что HTTPS шифрует трафик от клиента к серверу, а HTTP – нет.

Если между клиентом и сервером есть прокси, то в случае с HTTP все WebSocket-заголовки и данные передаются через него. Прокси имеет к ним доступ, ведь они никак не шифруются, и может расценить происходящее как нарушение протокола HTTP, обрезать заголовки или оборвать передачу.

А в случае с WSS весь трафик сразу кодируется и через прокси проходит уже в закодированном виде. Поэтому заголовки гарантированно пройдут, и общая вероятность соединения через WSS выше, чем через WS.

**Чистое закрытие**

При закрытии соединения сторона, желающая это сделать (обе стороны в WebSocket равноправны) отправляет закрывающий фрейм (опкод 0x8), в теле которого указывает причину закрытия.

В браузерной реализации эта причина будет содержаться в свойстве reason события onclose.

Наличие такого фрейма позволяет отличить «чистое закрытие» от обрыва связи.

В браузерной реализации событие onclose при чистом закрытии имеет event.wasClean = true.

**Коды закрытия**

Коды закрытия вебсокета event.code, чтобы не путать их с HTTP-кодами, состоят из 4 цифр:

1000

Нормальное закрытие.

1001

Удалённая сторона «исчезла». Например, процесс сервера убит или браузер перешёл на другую страницу.

1002

Удалённая сторона завершила соединение в связи с ошибкой протокола.

1003

Удалённая сторона завершила соединение в связи с тем, что она получила данные, которые не может принять. Например, сторона, которая понимает только текстовые данные, может закрыть соединение с таким кодом, если приняла бинарное сообщение.

**3) Порядок выполнения работы:**

Сервер поддерживает переписку между пользователями, также дает возможность кодировать и декодировать входное сообщения от пользователя.

* encrypt <сообщение>, <пароль> - должен реализовывать метод шифрования из лабораторной работы №3 или №4 (на усмотрение студента) по дисциплине «Информационная безопасность и защита информации» согласно варианту
* decrypt <сообщение>, <пароль> - должен реализовывать метод дешифрования из лабораторной работы №3 или №4 (на усмотрение студента) по дисциплине «Информационная безопасность и защита информации» согласно варианту

Запуск приложения производить с помощью двух терминальных команд:

1. Start Server(Backend)

node server

1. Start Client web page(Frontend - <http://localhost:8080/>)

http-server

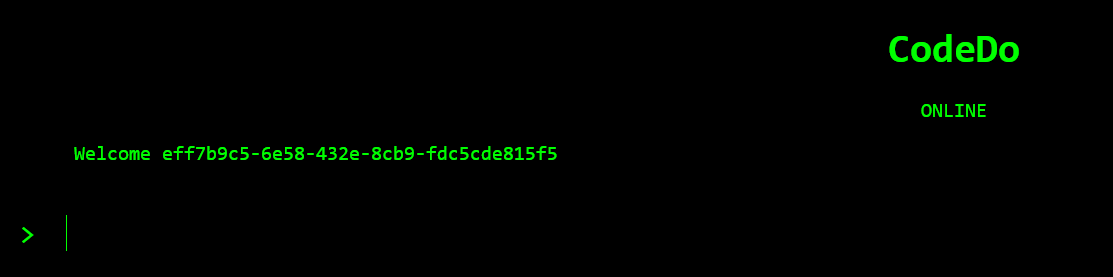


Рис. 1 Client web page(Frontend)

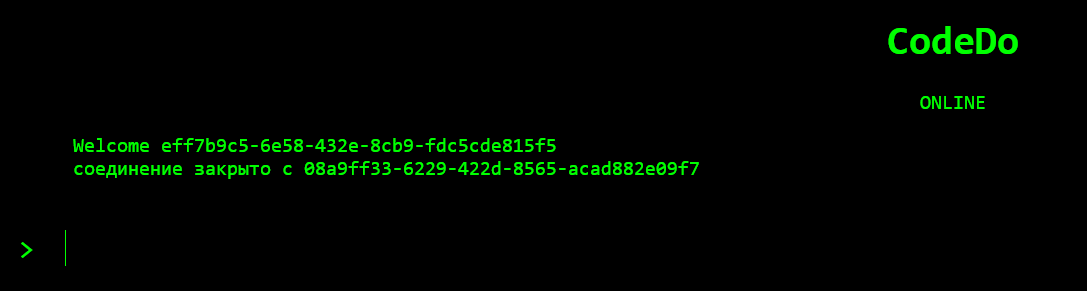
 

Рис. 2 Пример отключения пользователя от сервера

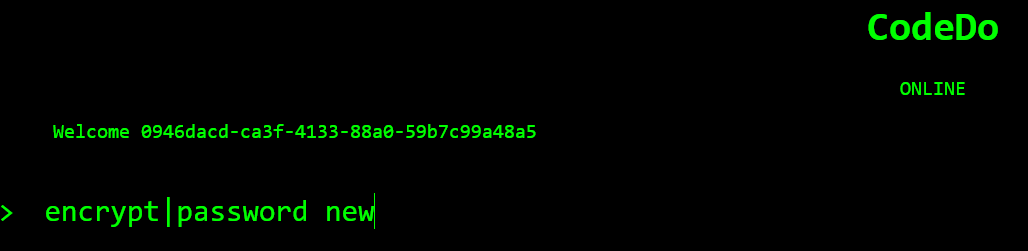
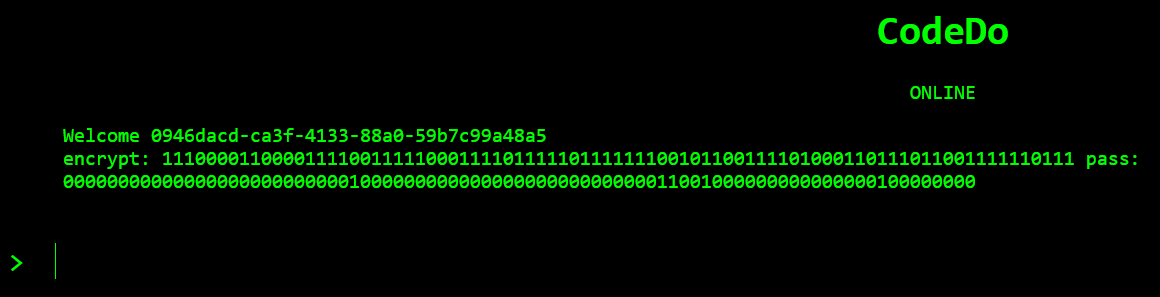
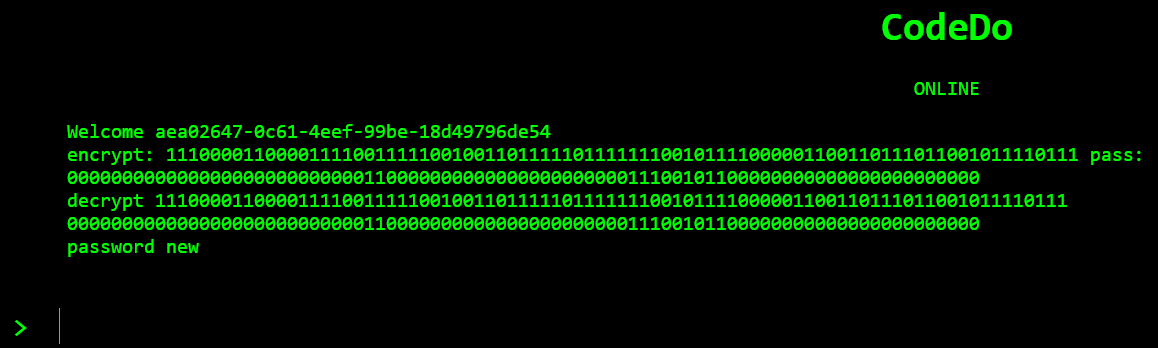
  

Рис. 3 encrypt и decrypt (server)

**Вывод:**

В ходе выполнения данной лабораторной работы, мне удалось приобрести практические знания по настраиванию и проектированию сетей, также научиться работать с сокетами и строить многопоточные серверные(клиентские) приложения на примере сервера шифрования.

**Список литературы:**

* Основы JavaScript https://html5book.ru/osnovy-javascript/ (Дата обращения 13.12.19).
* Лабораторная работа Мартынов Антон Иванович - Сети и телекоммуникации.
* WebSocket https://learn.javascript.ru/websockets (Дата обращения 13.12.19).

**Приложение 1**

Server.js

const WebSocket = require("ws");

require("pidcrypt/seedrandom")

var pidCrypt = require("pidcrypt")

require("pidcrypt/aes\_cbc")

const port = 5000;

const server = new WebSocket.Server({ port });

// подключённые клиенты

let clients = {};

let aes = new pidCrypt.AES.CBC();

generateUUID = () => {

return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace( /[xy]/g, ( c ) => {

let r = Math.random() \* 16 | 0;

return ( c == 'x' ? r : ( r & 0x3 | 0x8 ) ).toString( 16 );

} );

};

sendMessage = (message) => {

server.clients.forEach(client => {

if (client.readyState === WebSocket.OPEN) {

client.send(message);

}

});

}

function Geffe(L1, L2, L3, n) {

let holder = "";

for(let i = 0; i < n; i++) {

L1 = (L1 << 1) | (((L1 >> 29)^ (L1 >> 28) ^ (L1 >> 25) ^ (L1 >> 23)) & 1);

L2 = (L2 << 1) | (((L2 >> 30)^ (L2 >> 27)) & 1);

L3 = (L3 << 1) | (((L3 >> 31)^ (L3 >> 30) ^ (L3 >> 29) ^ (L3 >> 28) ^ (L3 >> 26) ^ (L3 >> 24)) & 1 );

holder += ((((L3 >> 32) & 1 )\*((L1 >> 30) & 1)) ^ ((((L3 >> 32) & 1) ^ 1) \* ((L2 >> 31) & 1)) );

}

return holder;

}

function getRandomInt(min, max) {

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \* (max - min)) + min; //Максимум не включается, минимум включается

}

function StringBin(mystring) {

let mybitseq = "";

let bin = "";

let end = mystring.length;

for(let i = 0 ; i < end; i++){

bin = mystring[i].charCodeAt().toString(2);

mybitseq += bin;

}

return mybitseq;

}

function roughScale(x, base) {

const parsed = parseInt(x, base);

if (isNaN(parsed)) { return 0 }

return parsed;

}

function BinString(bin\_string) {

let mybitseq = '';

let hex\_seq = bin\_string[0];

let end = bin\_string.length;

let char = '';

for(let i = 1 ; i <= end; i++) {

if(hex\_seq.length % 7 == 0) {

char = String.fromCharCode(roughScale(hex\_seq, 2));

hex\_seq = '';

mybitseq += char;

}

hex\_seq += bin\_string[i];

if (hex\_seq === '100000') {

hex\_seq = "";

mybitseq += ' ';

continue;

}

}

return mybitseq;

}

server.on("connection", ws => {

let id = generateUUID();

clients[id] = ws;

ws.on("message", message => {

if (message === 'disconnect') {

ws.close()

} else {

let res = message.split('|');

switch (res[0]) {

case 'encrypt':

let text = StringBin(res[1]);

let len = text.length;

let l1 = getRandomInt(1, 29); // 29

let l2 = getRandomInt(1, 30); // 30

let l3 = getRandomInt(1, 31); // 31

let sequence = Geffe(l1, l2, l3, len);

let crypted = '';

for(let i = 0; i < len; i++) {

crypted += (text[i] - 0) ^ (sequence[i] - 0);

}

sendMessage('encrypt: ' + crypted + ' pass: ' + sequence);

break;

case 'decrypt':

let size = res[1].length;

let bin\_sequence = res[1];

let geff = res[2];

let bin\_text = '';

for(let i = 0; i < size; i++) {

bin\_text += (bin\_sequence[i] - 0) ^ (geff[i] - 0);

}

let decrypted = BinString(bin\_text);

sendMessage(decrypted);

break;

default:

server.clients.forEach(client => {

if (client.readyState === WebSocket.OPEN) {

client.send(message);

}

});

break;

}

}

});

ws.on('close', function() {

server.clients.forEach((client) => {

if (client.readyState === WebSocket.OPEN) {

client.send(`соединение закрыто c ${id}`);

}

});

delete clients[id];

});

ws.send(`Welcome ${id}`);

});

App.js

const status = document.getElementById('status');

const messages = document.getElementById('messages');

const form = document.getElementById('form');

const input = document.getElementById('input');

const ws = new WebSocket('ws://localhost:5000')

function setStatus(value) {

status.innerHTML = value;

}

function printMessage(value) {

const li = document.createElement('li');

li.innerHTML = value;

messages.appendChild(li);

}

form.addEventListener('submit', e=>{

e.preventDefault();

ws.send(input.value);

input.value=''

})

ws.onopen = () => setStatus('ONLINE')

ws.onclose = () => setStatus('DISCONNECTED')

ws.onmessage = response => printMessage(response.data)