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 №3

на тему

**АТАКИ ПРИ УСТАНОВКЕ TCP-СОЕДИНЕНИЯ И ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ**

Выполнил Толстой Д.В.

Проверил Лещенко Е. А.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc160203598)

[2 Результат выполнения 4](#_Toc160203599)

[ЗАКЛЮЧЕНИЕ 6](#_Toc160203600)

[Приложение А Листинг кода 7](#_Toc160203601)

# 1 ЦЕЛЬ РАБОТЫ

Целью выполнения данной лабораторной работы является изучение теоретических сведений по работе адресации в сети Internet, физического, канального и транспортного уровней, а также разработка программы, реализующей работу TCP-протокола и атак на TCP-соединение.

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В ходе выполнения лабораторной работы было создано консольное приложение, реализующее работу TCP-протокола и атак на TCP-соединение. На рисунке 1 показана работа выполнения программы без различных атак на протокол TCP.

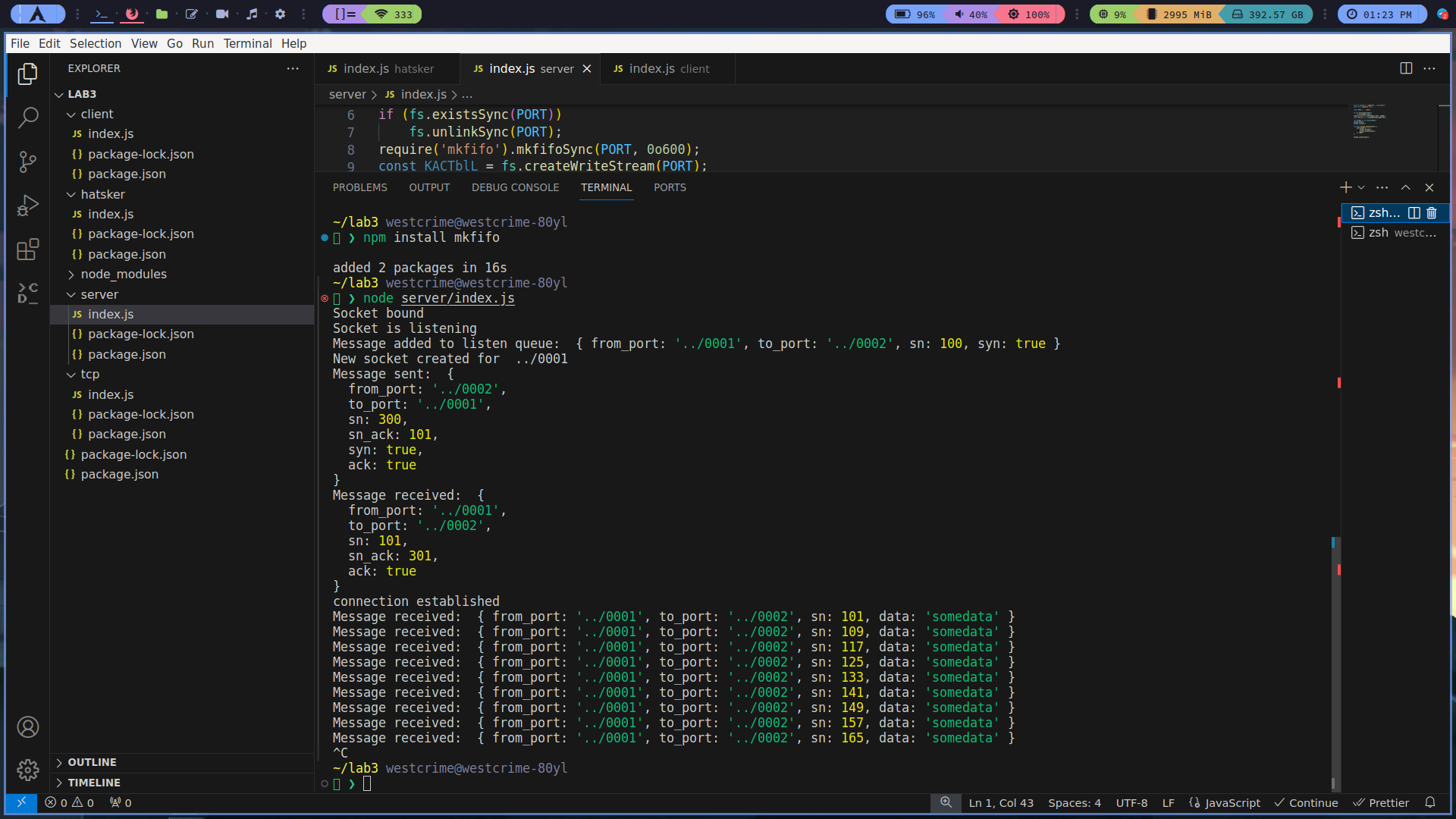


Рисунок 1 – Результат выполнения программы без атак

В качестве атак на протокол TCP были выбраны две атаки: SYN Flooding и TCP Reset Attack. На рисунке 2 показано выполнение работы атаки SYN Flooding. В этой атаке злоумышленник посылает большое количество SYN запросов на целевой сервер, но не завершает установку соединения, не отправляя ACK в ответ на SYN+ACK пакеты от сервера.

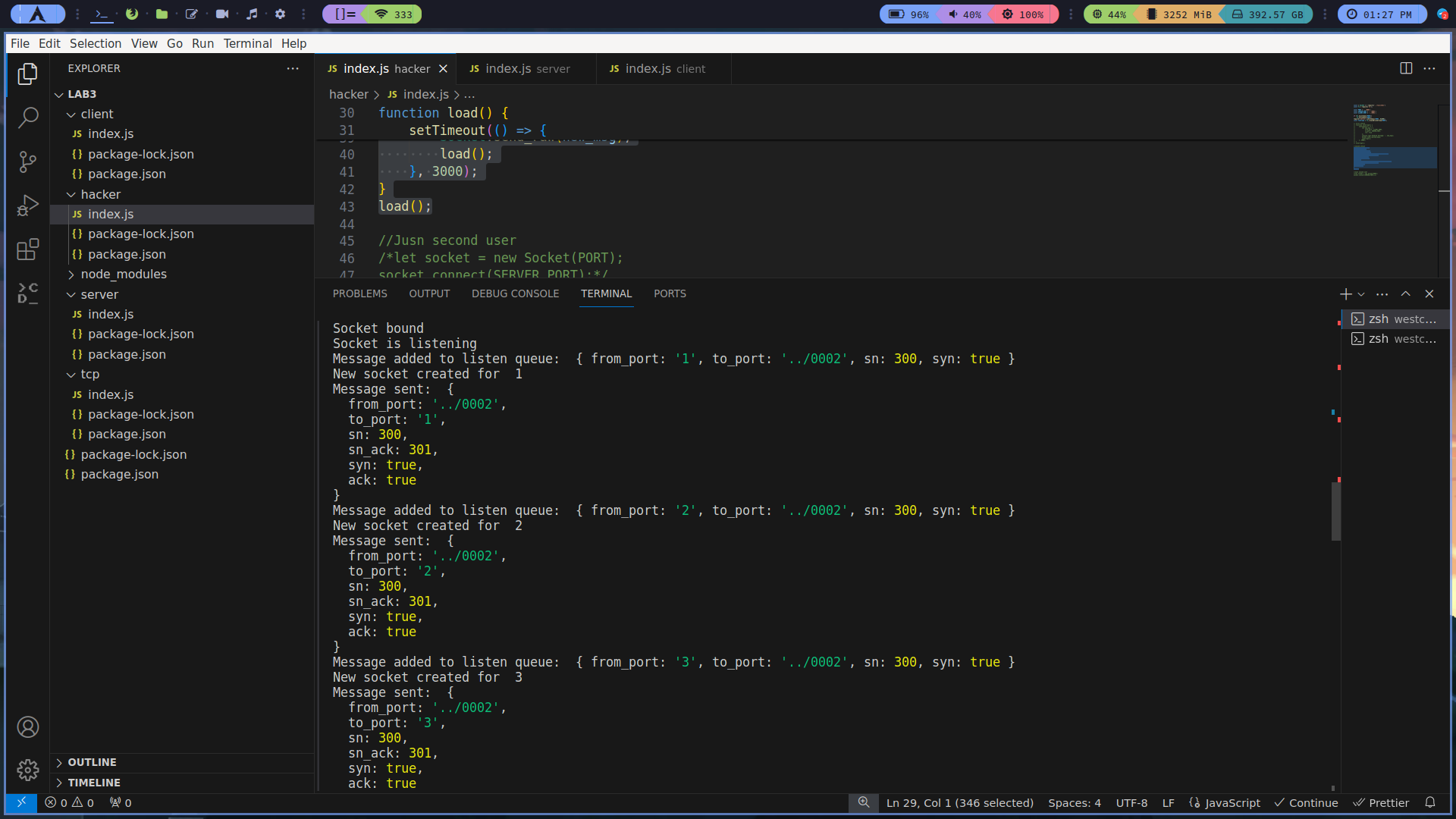


Рисунок 2 – Результат работы атаки SYN Flooding

Следующей атакой является TCP Reset Attack. Злоумышленник может подделать пакет RST (Reset) и отправить его на сервер или клиент, чтобы принудительно завершить TCP-соединение. Это может быть использовано для прерывания активного соединения и вызывать проблемы в работе приложения. На рисунках 3 и 4 показана работа выполнения атаки TCP Reset. Сервер игнорирует сообщение с RST.

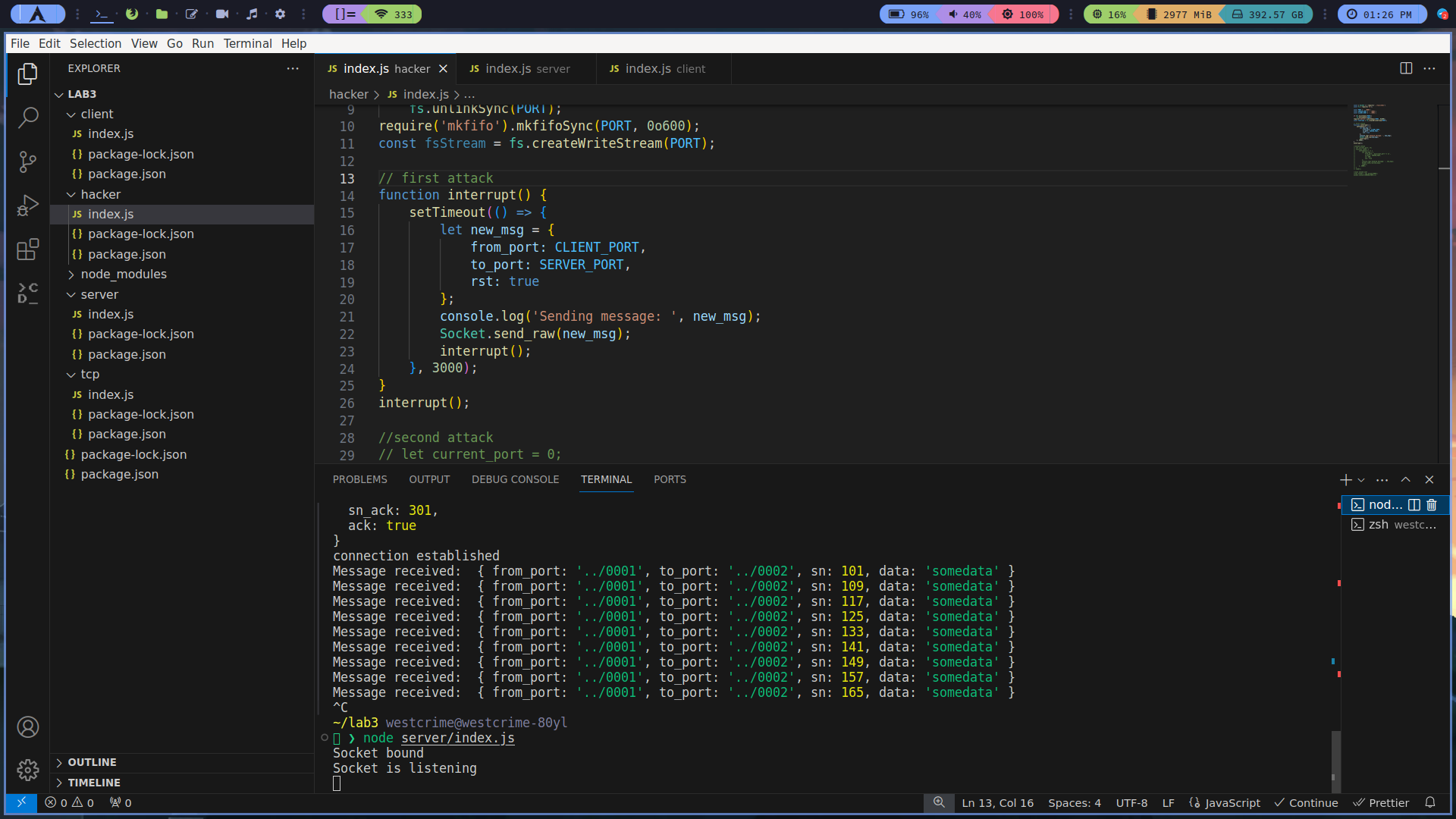


Рисунок 3 – Результат работы атаки TCP Reset со стороны сервера

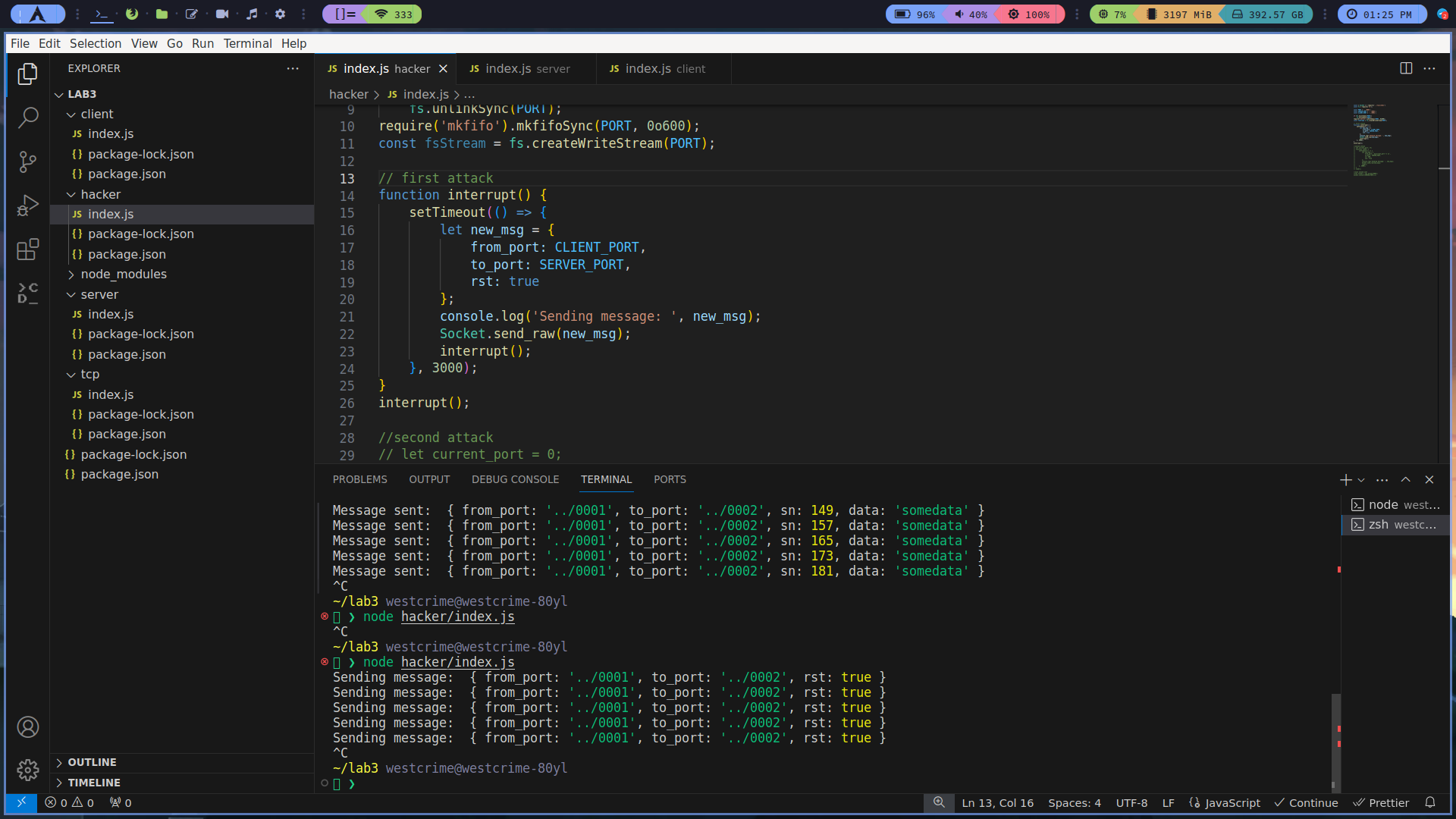


Рисунок 4 – Результат работы атаки TCP Reset со стороны злоумышленника

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены теоретические сведения по работе адресации в сети Internet, физического, канального и транспортного уровней, а также была разработана программа, реализующая работу TCP-протокола и атак на TCP-соединение

# ПРИЛОЖЕНИЕ А Листинг кода

Server/index.js

const { Socket } = require('tcp');

const fs = require('fs');

const PORT = '../0002';

if (fs.existsSync(PORT))

    fs.unlinkSync(PORT);

require('mkfifo').mkfifoSync(PORT, 0o600);

const FsStream = fs.createWriteStream(PORT);

let socket = new Socket(PORT);

socket.bind();

socket.listen();

function accept\_connections() {

    setTimeout(() => {

        socket.accept();

        accept\_connections();

    }, 1000);

}

accept\_connections();

client/index.js

const { Socket } = require('tcp')

const fs = require('fs');

const PORT = '../0001';

const SERVER\_PORT = '../0002';

if (fs.existsSync(PORT))

    fs.unlinkSync(PORT);

require('mkfifo').mkfifoSync(PORT, 0o600);

const FsStream = fs.createWriteStream(PORT);

let socket = new Socket(PORT);

function send() {

    setTimeout(() => {

        socket.send("somedata");

        send();

    }, 3000);

}

function wait\_connect() {

    setTimeout(() => {

        if (!socket.connected()) {

            wait\_connect();

        }

        else {

            send();

        }

    }, 1000);

}

socket.connect(SERVER\_PORT);

wait\_connect();

tcp/index.js

const fs = require('fs');

const readline = require('readline');

const SLICING\_SIZE = 100;

const socket\_states = {

    closed: 0,

    listen: 1,

    syn\_sent: 2,

    syn\_received: 3,

    established: 4,

};

class Socket {

    state = socket\_states.closed;

    bound = false;

    constructor(port) {

        this.from\_port = port;

    }

    bind() {

        if (this.state != socket\_states.closed) {

            throw new Error('Cannot bind open socket');

        }

        this.bound = true;

        this.message\_queue = [];

        console.log("Socket bound");

    }

    listen() {

        if (!this.bound) {

            throw new Error('Socket must be bound to listen');

        }

        this.state = socket\_states.listen;

        this.start\_processing();

        console.log("Socket is listening");

    }

    close() {

        this.bound = false;

        this.data = undefined;

        this.message\_queue = undefined;

        this.state = socket\_states.closed;

    }

    send(data) {

        for (let i = 0; i < data.length; i += SLICING\_SIZE) {

            let sent\_size;

            if (i + SLICING\_SIZE >= data.length) {

                sent\_size = data.length - i;

            }

            else {

                sent\_size = SLICING\_SIZE;

            }

            let new\_msg = {

                from\_port: this.from\_port,

                to\_port: this.to\_port,

                sn: this.sn,

                data: data.substring(i, i + sent\_size)

            };

            Socket.send\_raw(new\_msg);

            console.log('Message sent: ', new\_msg);

            this.sn += sent\_size;

        }

    }

    recv() {

        if (this.data.length != 0) {

            if (self.state != socket\_states.established) {

                throw new Error('Socket cannot receive');

            }

            result = this.data;

            this.data = '';

            return result;

        }

    }

    accept() {

        if (this.state != socket\_states.listen) {

            throw new Error('Cannot accept on non-listening socket');

        }

        if (this.message\_queue.length != 0) {

            let msg = this.message\_queue.shift();

            let new\_socket = new Socket(this.from\_port);

            console.log("New socket created for ", msg.from\_port);

            new\_socket.state = socket\_states.syn\_received;

            new\_socket.to\_port = msg.from\_port;

            new\_socket.sn\_ack = msg.sn + 1;

            new\_socket.sn = 300;

            new\_socket.data = '';

            Socket.sockets[new\_socket.from\_port + new\_socket.to\_port] = new\_socket;

            let new\_msg = {

                from\_port: new\_socket.from\_port,

                to\_port: new\_socket.to\_port,

                sn: new\_socket.sn,

                sn\_ack: new\_socket.sn\_ack,

                syn: true,

                ack: true

            };

            Socket.send\_raw(new\_msg);

            console.log('Message sent: ', new\_msg);

            new\_socket.sn += 1;

            return new\_socket;

        }

    }

    connect(port) {

        if (this.state != socket\_states.closed) {

            throw new Error('Cannot connect open socket');

        }

        this.sn = 100;

        this.data = '';

        this.to\_port = port;

        this.state = socket\_states.syn\_sent;

        let new\_msg = {

            from\_port: this.from\_port,

            to\_port: this.to\_port,

            sn: this.sn,

            syn: true

        };

        Socket.send\_raw(new\_msg);

        console.log('Message sent to server: ', new\_msg);

        this.sn += 1;

        this.start\_processing();

    }

    connected() {

        return this.state == socket\_states.established;

    }

    start\_processing() {

        const stream = fs.createReadStream(this.from\_port);

        var rl = readline.createInterface({

            input: stream

        });

        //const FsStream = fs.createWriteStream(this.from\_port);

        rl.on('line', (line) => {

            let msg = JSON.parse(line);

            this.process\_message(msg);

        });

    }

    process\_message(msg) {

        switch(this.state) {

            case socket\_states.listen:

                if ((msg.to\_port+msg.from\_port) in Socket.sockets)

                {

                    Socket.sockets[msg.to\_port + msg.from\_port].process\_message(msg);

                }

                else if (msg.syn && msg.sn) {

                    console.log('Message added to listen queue: ', msg);

                    this.message\_queue.push(msg);

                }

                break;

            case socket\_states.syn\_sent:

                if (this.to\_port == msg.from\_port) {

                    if (msg.rst) {

                        console.log("Connection interrupted!");

                        this.close();

                    }

                    if (msg.sn\_ack == this.sn && msg.sn && msg.syn && msg.ack) {

                        console.log('Message received: ', msg);

                        this.sn\_ack = msg.sn + 1;

                        this.state = socket\_states.established;

                        console.log("connection established");

                        let new\_msg = {

                            from\_port: this.from\_port,

                            to\_port: this.to\_port,

                            sn: this.sn,

                            sn\_ack: this.sn\_ack,

                            ack: true

                        };

                        Socket.send\_raw(new\_msg);

                        console.log('Message sent: ', new\_msg);

                    }

                }

                break;

            case socket\_states.syn\_received:

                if (this.to\_port == msg.from\_port) {

                    if (msg.rst) {

                        console.log("Connection interrupted!");

                        this.close();

                    }

                    if (this.sn == msg.sn\_ack && msg.ack) {

                        console.log('Message received: ', msg);

                        this.state = socket\_states.established;

                        console.log("connection established");

                    }

                }

            case socket\_states.established:

                if (this.to\_port == msg.from\_port) {

                    if (msg.rst) {

                        console.log("Connection interrupted!");

                        this.close();

                    }

                    if (msg.data && this.sn\_ack == msg.sn) {

                        console.log('Message received: ', msg);

                        this.sn\_ack += msg.data.length;

                        this.data += msg.data;

                    }

                }

            default:

                break;

        }

    }

    static send\_raw(msg) {

        if (fs.existsSync(msg.to\_port))

        {

            const stream = fs.createWriteStream(msg.to\_port);

            stream.write(JSON.stringify(msg) + '\n');

        }

    }

    static sockets = {};

}

module.exports = { Socket };

hacker/index.js

const { Socket } = require('tcp')

const fs = require('fs');

const PORT = '../0003';

const SERVER\_PORT = '../0002';

const CLIENT\_PORT = '../0001';

if (fs.existsSync(PORT))

    fs.unlinkSync(PORT);

require('mkfifo').mkfifoSync(PORT, 0o600);

const KACTblL = fs.createWriteStream(PORT);

// first attack

/\*function interrupt() {

    setTimeout(() => {

        let new\_msg = {

            from\_port: CLIENT\_PORT,

            to\_port: SERVER\_PORT,

            rst: true

        };

        console.log('Sending message: ', new\_msg);

        Socket.send\_raw(new\_msg);

        interrupt();

    }, 3000);

}

interrupt();\*/

//second attack

let current\_port = 0;

function load() {

    setTimeout(() => {

        let new\_msg = {

            from\_port: `${current\_port += 1}`,

            to\_port: SERVER\_PORT,

            sn: 300,

            syn: true

        };

        console.log('Sending message: ', new\_msg);

        Socket.send\_raw(new\_msg);

        load();

    }, 3000);

}

load();

//Jusn second user

/\*let socket = new Socket(PORT);

socket.connect(SERVER\_PORT);\*/