**NON-REPUDIATION FOR A PEER-TO-PEER CONNECTION USING SOCKET PROGRAMMING**

**A PROJECT REPORT**

*Submitted by*

Keshav Moorthy

16BCI0066

Course Code: CSE2008

Course Title: Network Security

*In partial fulfilment for the award of the degree of*

**B.Tech**

**in**

**Computer Science and Engineering**

Under the guidance of

**Dr. K. Anil Kumar**

**Associate Professor, SCOPE,**

**VIT University, Vellore.**

****

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**NOVEMBER, 2016**

**INDEX**

1. Introduction
   1. Peer to Peer Connections
   2. Socket Programming
   3. Non Repudiation
2. Background of the Work
3. Overview of the Work
   1. Problem description
   2. Working model
   3. Design description
4. Implementation
   1. Description of Modules/Programs
   2. Source Code
   3. Test cases
   4. Execution snapshots
5. Conclusion and Future Directions
6. References

# ABSTRACT

This is a project which demonstrates a peer to peer messaging application where one client can speak with another client and no unauthorized user can enter the chat. If there’s an attack that occurs, the project makes sure that the hacker cannot modify messages. One client can also not deny against sending a message. Multiple clients in the same room is also possible and there are authentication protocols used to make sure attackers cannot attack a room.

# INTRODUCTION

## Peer to Peer connections

In its simplest kind, a peer-to-peer (P2P) network is made once 2 or additional PCs are connected and share resources while not browsing a separate server laptop. A P2P network are often an advertisement hoc connection—a few computers connected via a Universal Serial Bus to transfer files. A P2P network can also be a permanent infrastructure that links a half-dozen computers during a little workplace over copper wires. Or a P2P network are often a network on a way grander scale during which special protocols and applications found out direct relationships among users over the web.

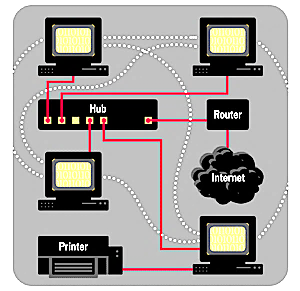


Figure 1 This diagram shows how a P2P network operates. The solid lines indicate physical, hard-wired network cables. The dotted lines indicate that each PC can communicate and share files with every other PC on such a network.

## Socket programming

Socket programming is a type of connection between two nodes on a network to communicate with one other. One socket or node listens on one particular kind of port within an IP, while another socket tries to reaches out to the other to form a type of connection. The server forms the listener socket while client reaches out to the server.

## Non-Repudiation

Nonrepudiation is a type of assurance that the sender cannot deny something. In other words, nonrepudiation refers to an ability to make sure that one party to a contract or a communication will not be able to deny the authenticity of their digital signature on one type of document or the sending of a message that it originated.

To try to repudiate means to deny or to leave. For many years, authorities have wanted to create repudiation not possible in some things. you may send registered post, for instance, therefore the recipient cannot deny that a letter was delivered. Similarly, a official document usually needs witnesses to sign language so the one who signs cannot deny having done thus.

On the net, a digital signature is employed not solely to make sure that a message or document has been electronically signed by the individual that speculated to sign the document, but also, since a digital signature will solely be created by one person, to make sure that someone cannot later deny that they well-found the signature.

Since no security technology is totally fool-proof, some specialists warn that a digital signature alone might not continuously guarantee nonrepudiation. it's recommended that multiple approaches be used, like capturing distinctive biometric data and alternative information regarding the sender or signer that put together would be troublesome to repudiate.

# BACKGROUND OF THE WORK

[1] addresses the delinquency of a direct secure communication between two arbitrary peers in a Peer2Peer network without anyone knowing the other's IP addresses. An effectual solution to this problem will give a virtually usable way of communication to privacy searching P2P applications. Traditionally, P2P architectures read this drawback as a haul of mutual namelessness of a message sender and receiver, however usual solutions suffer from numerous inefficiencies or complexities. By viewing the matter from a perspective of associate degree ad-hoc network, they were able to apply a well-known approach of multihop communication and ad-hoc routing algorithms to a P2P overlay. Introduced usage of a public key as a node's symbol adds additional safety features, as well as information integrity through digital signatures, or end-to-end secret writing. projected P2P overlay has been with success used for building a redistributed cryptocurrency exchange Coincer. several applications would like impulsive 2 peers to be able to communicate with one another, still a number of them extend this demand to additionally minimising a risk of exposure of peers' information processing addresses (identities). The motivation may be simply to tackle privacy issues or to stop targeted attacks against users of the applying. In our case, we have a tendency to develop a redistributed exchange of cryptocurrencies: Coincer [1]. By concealment information processing addresses we would like to produce users with an analogous level of privacy as cryptocurrencies typically supply. Moreover, as Coincer's commerce protocol might span from tens of minutes (due to inherent properties of the cryptocurrencies so as to take care of security of the trade) up to many hours (depending on degree of cooperation between the individual users) it's terribly useful to hide information processing addresses of peers, since it hinders efforts of attackers to stop their commerce partner from finishing the trade or perhaps directly assaultive them.

Digital streaming Internet applications such as online gaming, multimedia playback, presentations, news feeds, and stock quotes involve end-users with very low tolerance for high latency, low data rates, and playback interruption. To protect such delay-sensitive streams against malicious attacks, security mechanisms need to be designed to efficiently process long sequence of bits. [2] studies the problem of efficient authentication for real-time and delay-sensitive streams commonly seen in content distribution, multicast, and peer-to-peer networks. They propose a unique signature amortization technique supported trapdoor hash functions for authenticating individual knowledge blocks in an exceedingly stream. Our technique provides: 1) Resilience against transmission losses of intermediate blocks within the stream;

2) little and constant memory/compute needs at the sender and receiver;

3) stripped constant communication overhead required for transmission of authenticating info. Our planned technique renders authentication of digital streams sensible and economical. They substantiate this claim by constructing DL-SA, a discrete-log-based internal representation of the planned technique. DL-SA provides adaptational stream verification, wherever the receiver has management over modulating computation value versus buffer size. Our performance analysis demonstrates that DL-SA incurs the smallest amount per-block communication and signature generation overheads compared to existing schemes with comparable options. several web-based services involve distribution of content like digital audio, video, software, games, stock quotes, streaming shows, and live news feeds through distributed networking technologies, like content distribution networks (CDN's), multicast networks, and peer-to-peer networks. sadly, these trendy distributed systems, designed to distribute content to an oversized cluster of users, additionally offer a platform for adversarial users to launch a myriad attack with widespread consequences. Adversaries will masquerade as legitimate content suppliers to distribute malicious content probably infected with worms, viruses, etc. Adversaries may also place themselves within the content distribution path, for instance, by compromising internet caches, and modify the content in ways in which will doubtless hurt shopper devices. Authenticating the content plays a vital role in preventing these attacks.

# OVERVIEW OF THE WORK

## Problem Description

An important feature in this globe of software development is the security of data that streams through open communication channels. In our web applications, there is a rigorous exchange of data via various protocols like https, http, etc between client applications which offered as browser, mobile and desktop applications and server side applications. The reputation and confidentiality of data may be different depending on the details of the web application, and the likelihood of interception by a third party increases with excellence of hacking techniques in the world of IT. What can be done to prevent access to the data by your traffic listener? If we exchange with data between the client applications and server we don’t want the data to be stored as open text on the server, which will be available in case of server crack or social engineering, or man in the middle attack. How do we avoid it?

## Working Model And Design Description

* A peer to peer network communication application is created using Socket Programming
* The network is launched as a web application
* Many different clients communicate to another by creating a unique name along with a password and joins a room
* The password is hashed using SHA and hence the hash cannot be revealed to the client
* Every client knows his/her own password which acts as a private key and the room key which acts as a public key.
* Clients outside a room cannot join room without the public key of the room
* Server watches the clients in all respective rooms and gets notified when new clients arrive. Attackers who have a reputation can easily be identified hence
* Every message is hashed and is encrypted using the private key of the sender and the public key which is the room key
* Do view the message the message must be decrypted with the public key of the room.
* Hence the communication is end to end encrypted
* When one user clicks on the button for checking non repudiation he gets notified if the message’s hash matches with the original message.
* This can easily be verified by decrypting the message with the public key.
* Hence clients can know if a message is altered or modified by an attacker

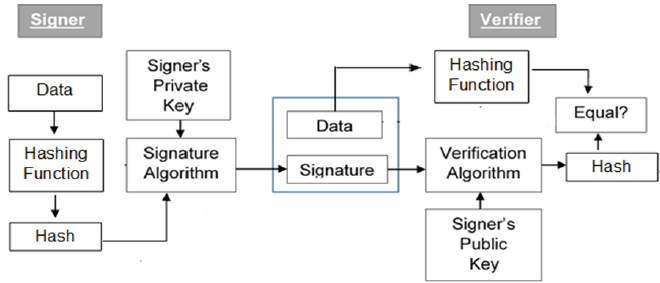


Figure 2 The sample flow type of every message sent using asymmetric encryption

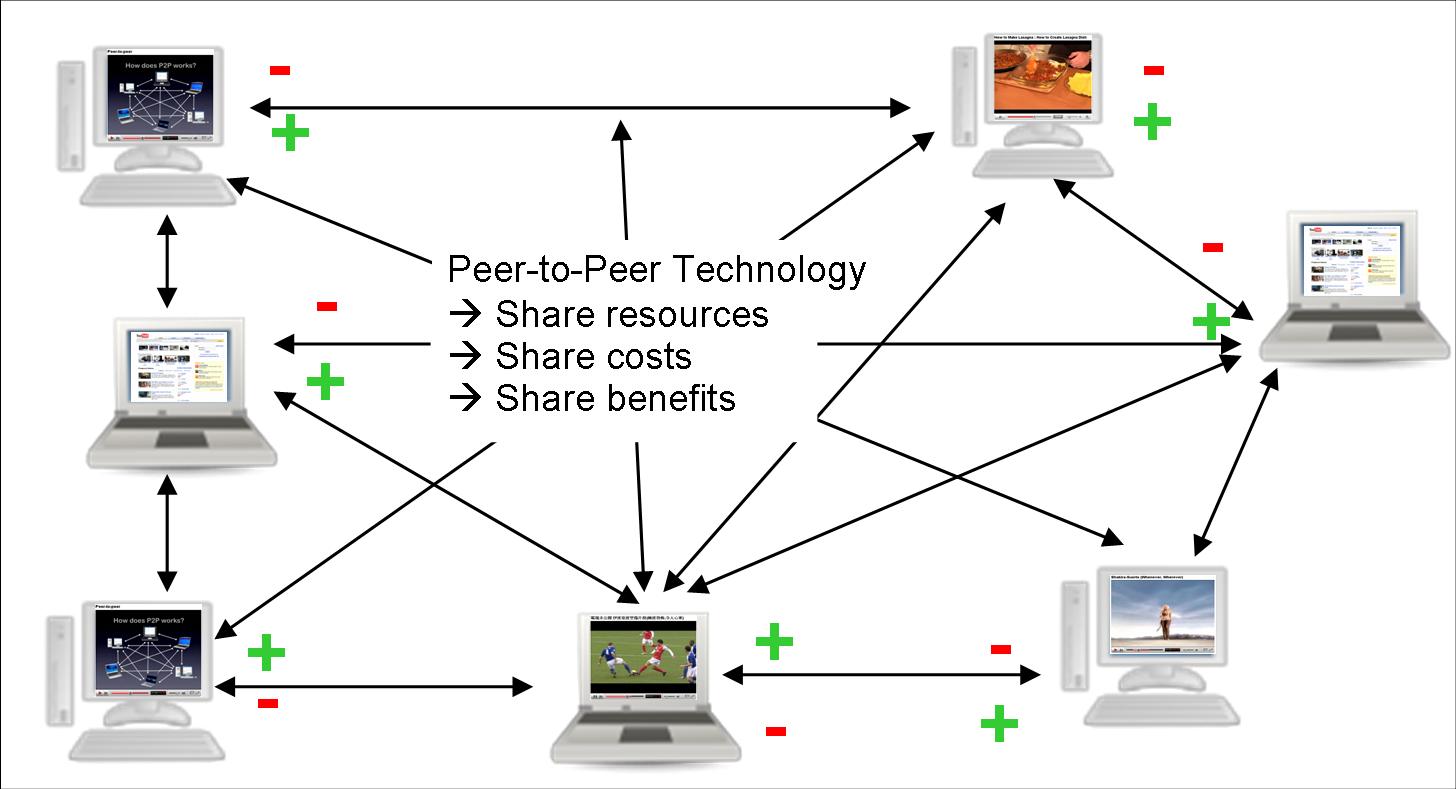


Figure 3 A peer to peer network

# IMPLEMENTATION

## Description Of Modules/Programs

### Socket.IO

Socket.IO is a JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it is event-driven.

Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.

It can be installed with the npm tool.

### Express.JS

Express.js, or simply Express, is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

The original author, TJ Holowaychuk, described it as a Sinatra-inspired server, meaning that it is relatively minimal with many features available as plugins. Express is the backend part of the MEAN stack, together with MongoDB database and AngularJS frontend framework.

### Mongoose.JS

Mongoose is a JavaScript framework that is commonly used in a Node.js application with a MongoDB database.

### Node.JS

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser.

Node.js lets developers use JavaScript for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server side and client side scripts.

## Server Module

The server can start a connection and has access to all public and private keys of clients and has the ability to decrypt any message sent by any client. They can additionally send a group message which whill be counted as a broadcast message.

## Source Code For Server Module

const path = require('path');

const http = require('http');

const express = require('express');

const socketIO = require('socket.io');

const {Message} = require('./utils/message');

const {isRealString} = require('./utils/validation');

const {Users} = require('./utils/users');

const bcrypt = require('bcrypt');

const publicPath = path.join(\_\_dirname, '../public');

const port = process.env.PORT || 3000;

var app = express();

var server = http.createServer(app);

var io = socketIO(server);

var users = new Users();

var msgs = new Message();

var incorrectPass;

// var {User} = require('./utils/user.js');

app.use(express.static(publicPath));

var hashIt = (pass) => {

bcrypt.genSalt(10, (err, salt) => {

bcrypt.hash(pass, salt, (err, hash) => {

console.log("New user key is",hash);

return hash;

});

});

}

io.on('connection', (socket) => {

console.log('New user connected');

socket.on('join', (params, callback) => {

if (!isRealString(params.name) || !isRealString(params.room)) {

return callback('room and name are required.');

}

if(incorrectPass){

return callback('Password incorrect');

}

socket.join(params.room);

users.removeUser(socket.id);

// var result = [{ "id": "1132", "city": "Manila Central Post Office", "province": "Manila" }];

// var val = JSON.parse(JSON.stringify(result));

// console.log(val[0].city);

// if(all.includes(params.name))

// if(compare(params.password), users.users[password][all.indexOf(params.name)])

// alert("works");

// else {

// alert("incorrect");

// }

// else{

// var password = hashIt(params.password);

if(users.getUser(params.name, params.password) =="Success")

console.log(`${params.name} Logged in`);

else if (users.getUser(params.name, params.password) =="Username does not exist"){

// console.log(hashIt(params.password));

users.addUser(socket.id, params.name, hashIt(params.password), params.room, params.roomkey);

io.to(params.room).emit('updateUserList', users.getUserList(params.room));

socket.emit('newMessage', msgs.generateMessage('Admin', 'Welcome to the chat app'));

socket.broadcast.to(params.room).emit('newMessage', msgs.generateMessage('Admin', `${params.name} has joined.`));

callback();

// console.log(users.getUser(params.name, params.password));

}

else if(users.getUser(params.name, params.password) =="Password incorrect"){

return callback('Password incorrect');

// console.log("incorrect");

}

// }

});

socket.on('createMessage', (message, callback) => {

var user = users.getUserDetails(socket.id);

if (user && isRealString(message.text)) {

console.log(user);

io.to(user.room).emit('newMessage', msgs.generateMessage(user.name, message.text));

}

callback();

});

// socket.on('createLocationMessage', (coords) => {

// var user = users.getUser(socket.id);

//

// if (user) {

// io.to(user.room).emit('newLocationMessage', generateLocationMessage(user.name, coords.latitude, coords.longitude));

// }

// });

socket.on('disconnect', () => {

var user = users.removeUser(socket.id);

if (user) {

io.to(user.room).emit('updateUserList', users.getUserList(user.room));

io.to(user.room).emit('newMessage', msgs.generateMessage('Admin', `${user.name} has left.`));

}

});

});

server.listen(port, () => {

console.log(`Server is up on ${port}`);

});

## Client Module

The client module can join a room and access his private key which is the hashed value of his/her own password. They can send a message to another client in the same room.

## Source Code For Client Module

var socket = io();

var actual = [];

function scrollToBottom () {

var hash;

jQuery('h6').click(function(){

pos = jQuery(jQuery(jQuery(this).parent())).parent().index();

text = jQuery(jQuery(jQuery(this).parent()).parent().children(".message\_\_body")).children("p").text();

if(actual[pos] === text)

alert("This message is unaltered");

else {

alert("THIS IS ALTERED. Original msg: "+actual[pos]);

}

});

// Selectors

var messages = jQuery('#messages');

var newMessage = messages.children('li:last-child')

// Heights

var clientHeight = messages.prop('clientHeight');

var scrollTop = messages.prop('scrollTop');

var scrollHeight = messages.prop('scrollHeight');

var newMessageHeight = newMessage.innerHeight();

var lastMessageHeight = newMessage.prev().innerHeight();

if (clientHeight + scrollTop + newMessageHeight + lastMessageHeight >= scrollHeight) {

messages.scrollTop(scrollHeight);

}

}

socket.on('connect', function () {

var params = jQuery.deparam(window.location.search);

socket.emit('join', params, function (err) {

if (err) {

alert(err);

window.location.href = '/';

} else {

console.log('No error');

}

});

});

socket.on('disconnect', function () {

console.log('Disconnected from server');

});

socket.on('updateUserList', function (users) {

var ol = jQuery('<ul></ul>');

users.forEach(function (user) {

ol.append(jQuery('<li></li>').text(user));

});

jQuery('#users').html(ol);

});

socket.on('newMessage', function (message) {

actual.push(message.text);

console.log(actual);

var formattedTime = moment(message.createdAt).format('h:mm a');

var template = jQuery('#message-template').html();

var html = Mustache.render(template, {

text: message.text,

from: message.from,

createdAt: formattedTime

});

jQuery('#messages').append(html);

scrollToBottom();

});

jQuery('#message-form').on('submit', function (e) {

e.preventDefault();

var messageTextbox = jQuery('[name=message]');

socket.emit('createMessage', {

text: messageTextbox.val()

}, function () {

messageTextbox.val('')

});

});

# EXECUTION SNAPSHOTS

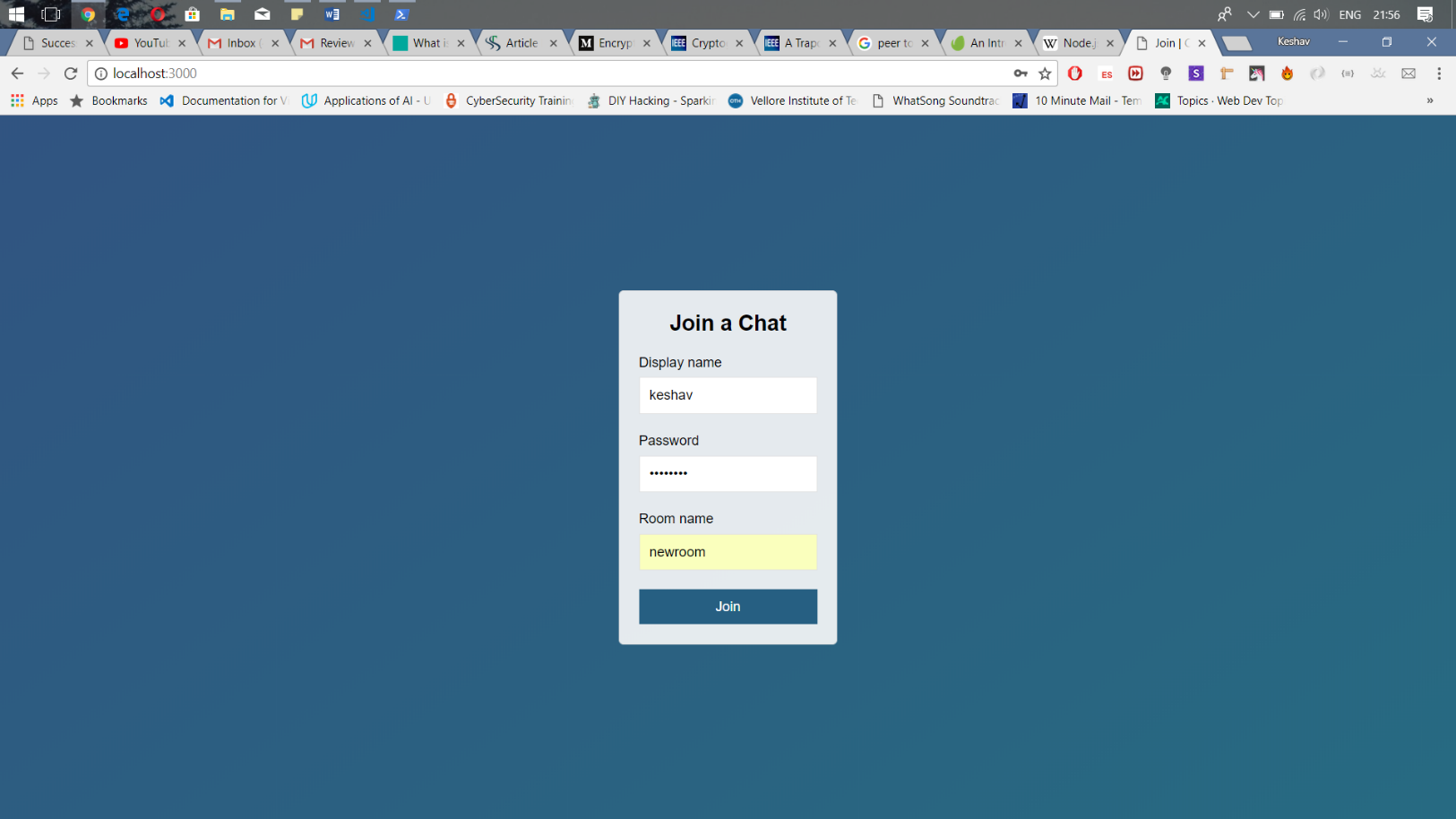


Figure 4 Client side login

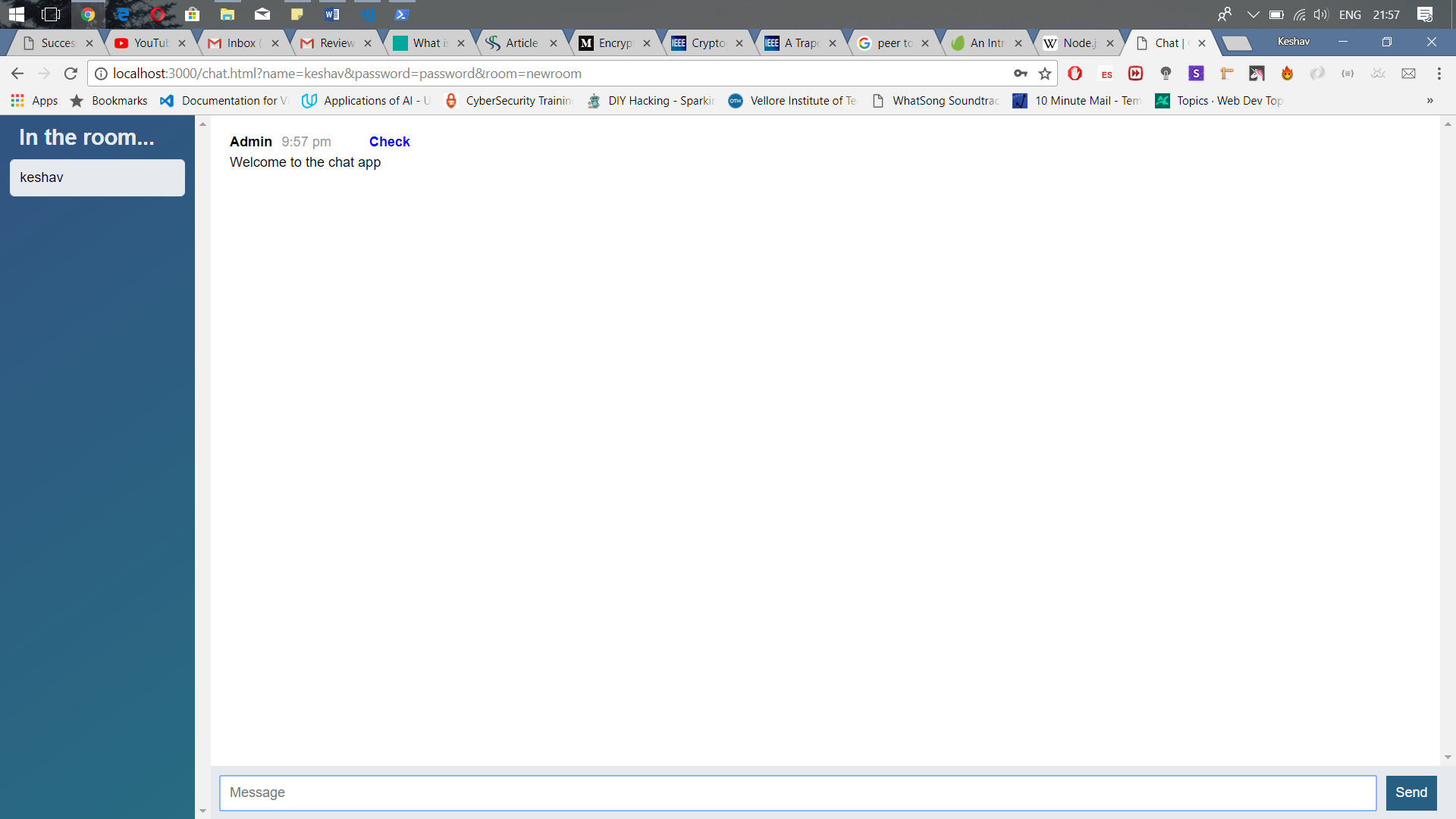


Figure 5 Client authorized to room

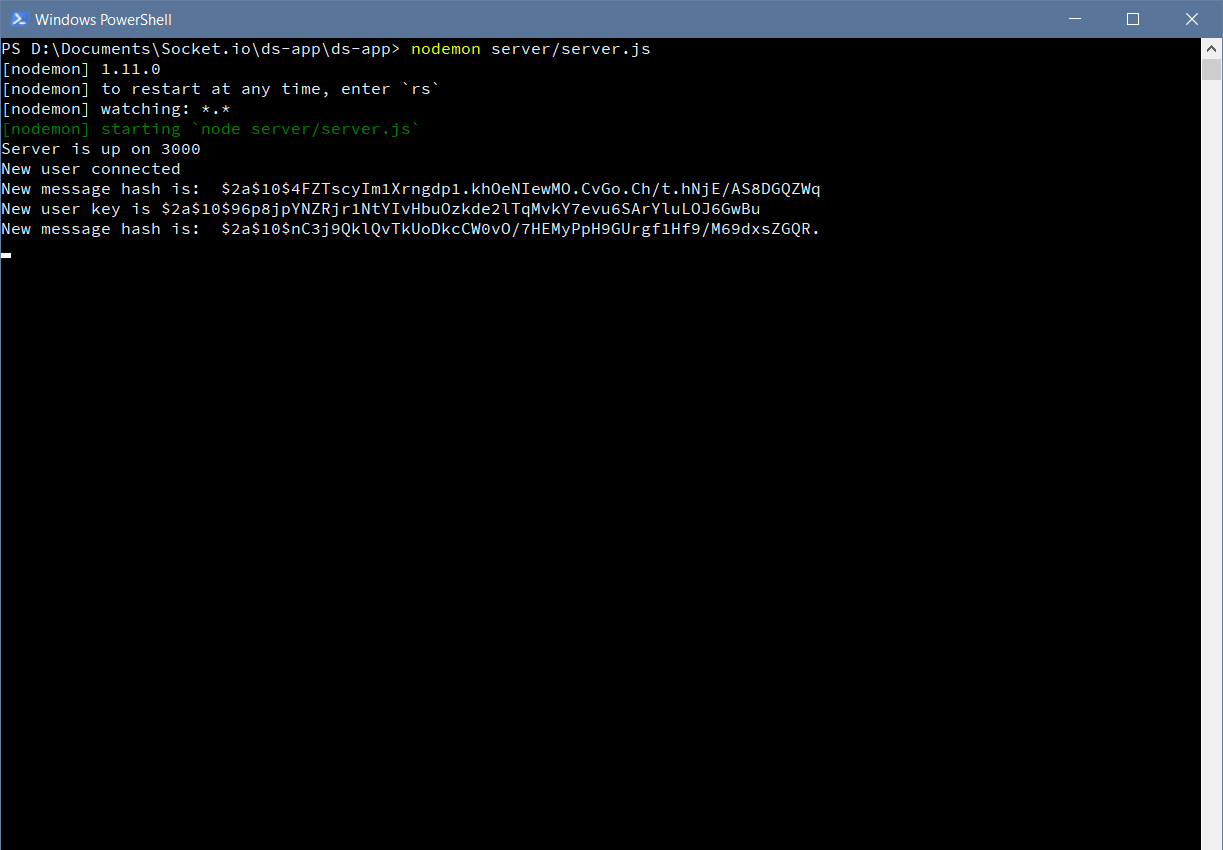


Figure 7 Server Side details of login

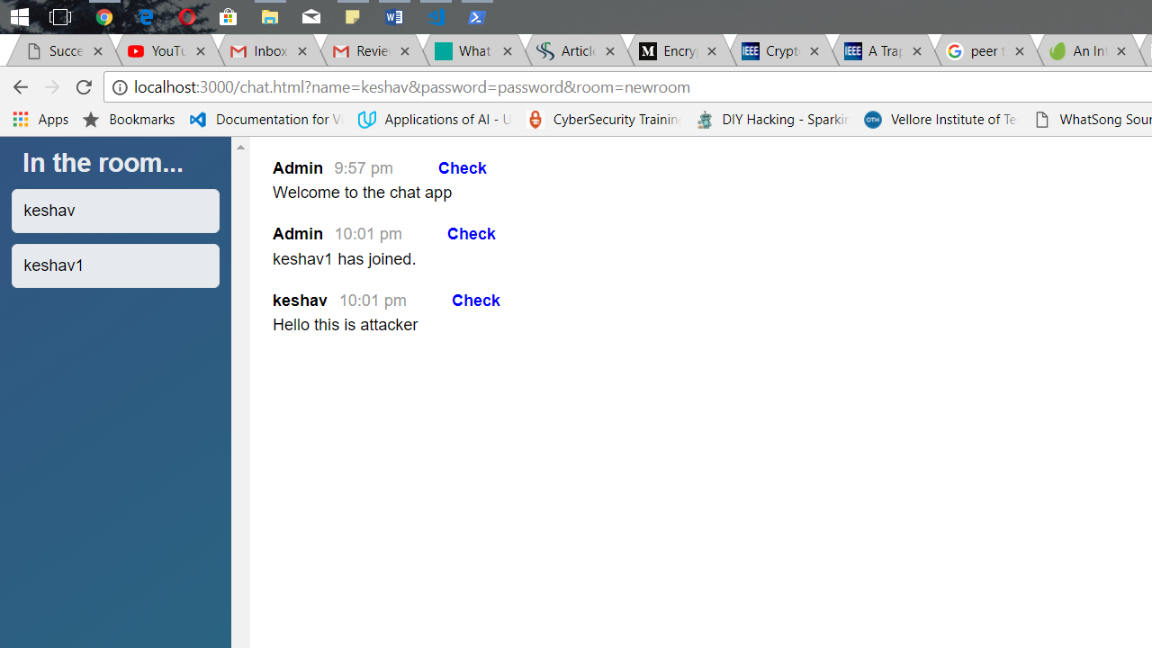
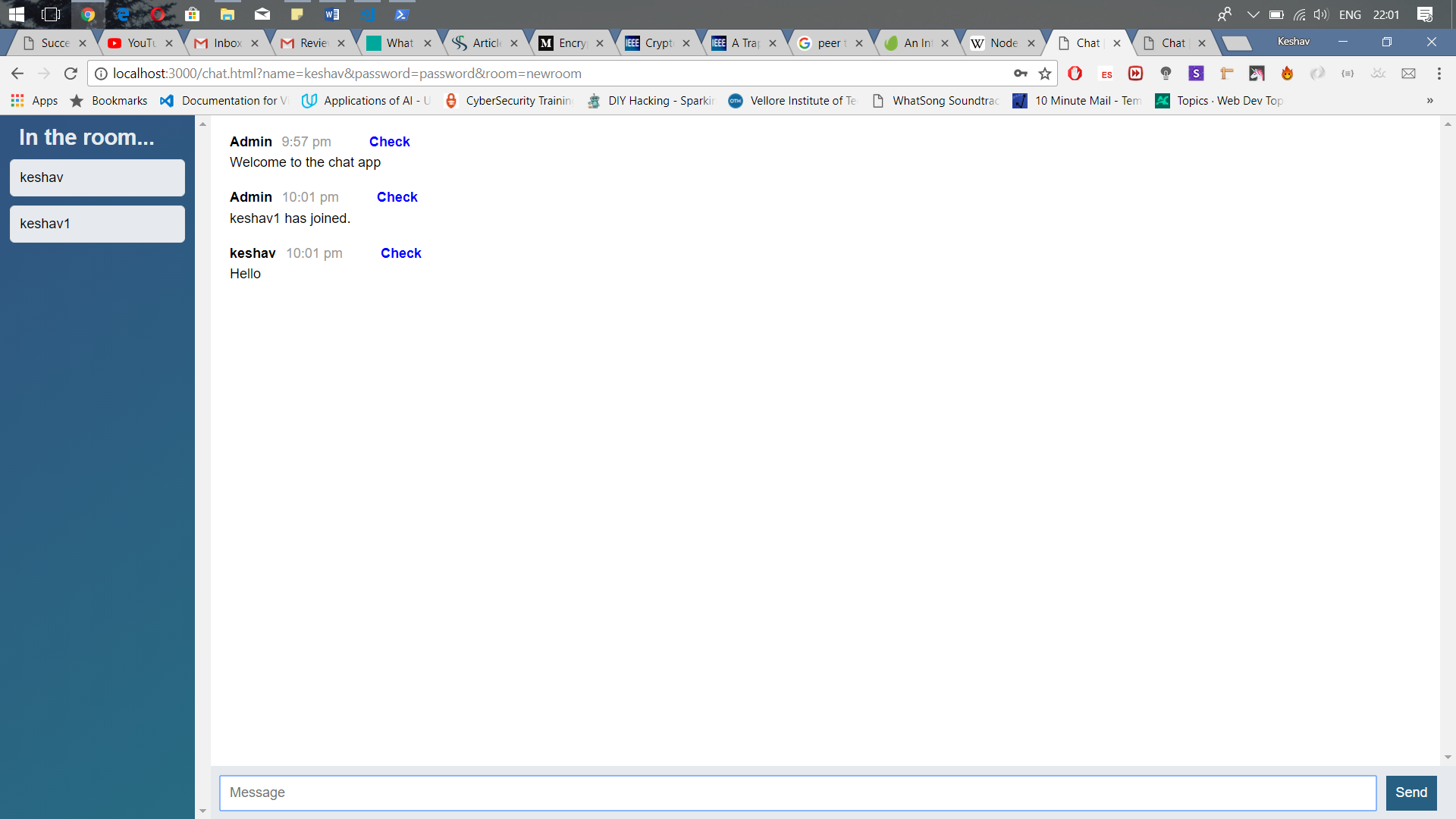


Figure 6 Client sends message to people in room

Figure 8 Message altered by attacker

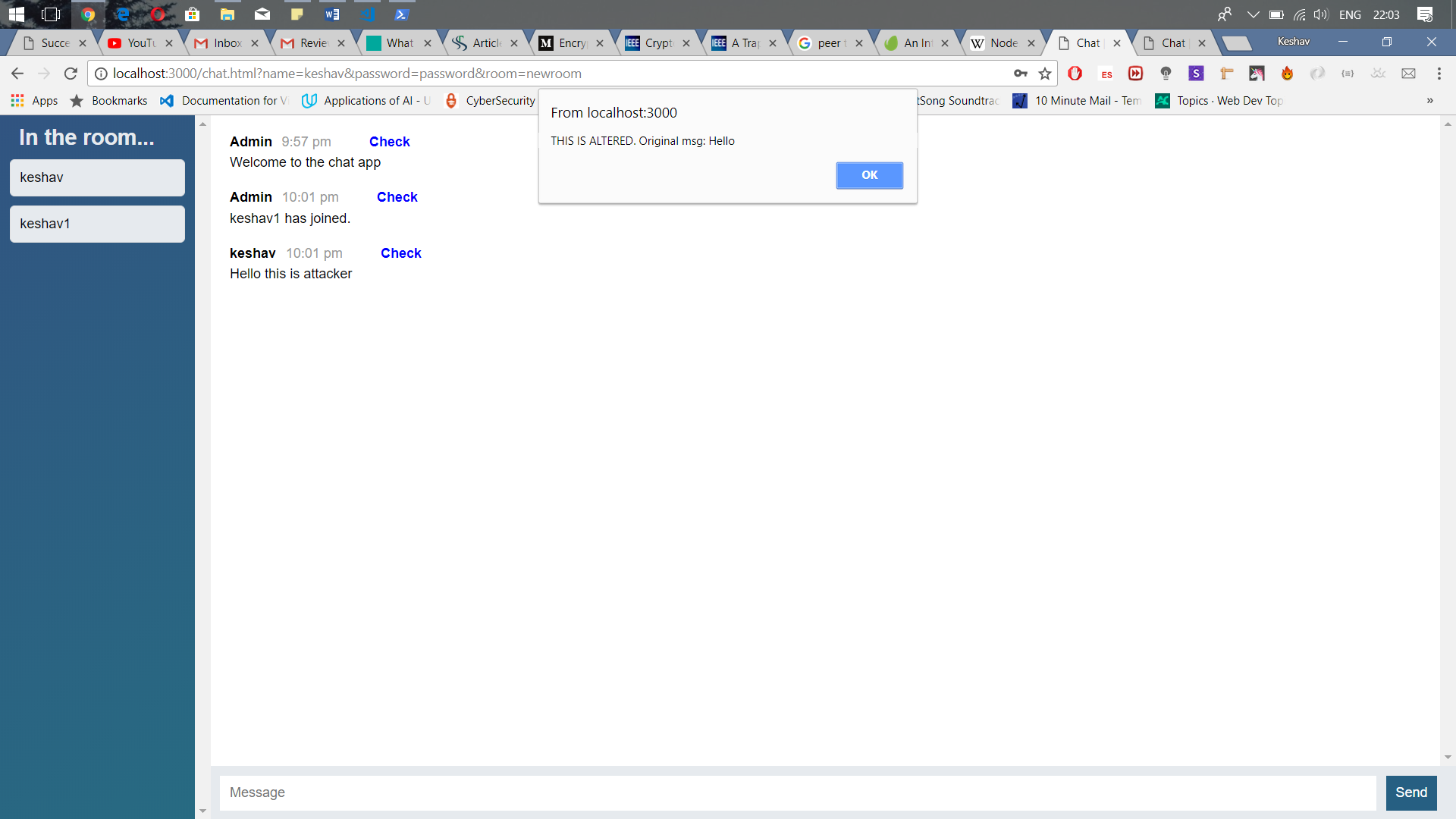


Figure 9 Application alerts when client clicks CHECK

# CONCLUSION

This project was successfully implemented by using javascript programming. The idea can further be extended to applications like Whatsapp, Facebook Messenger and many more client to client applications in the market existing today. It could also be used in emails to verify if a message was altered or not. The user can immediately be altered if it is. Messages which contain sensitive information cannot be manupalted as per the given schema of the application

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Title** | **Page No.** |
| **1** | P2P network operation | **4** |
| **2** | Message Flow | **9** |
| **3** | Peer to Peer Technology | **9** |
| **4** | Client Side Login | **18** |
| **5** | Client Authorized to Room | **19** |
| **6** | Server notified for login | **19** |
| **7** | Client message | **20** |
| **8** | Message Attacked | **21** |
| **9** | Client alerted for attack | **21** |

**REFERENCES**

[1] J. Jang-Jaccard, S. Nepal, B. Celler, and B. Yan, “WebRTC-based

video conferencing service for telehealth,” Computing, vol. 98, no.

1–2, pp. 169–193, 2016.

[2] M. Phankokkruad and P. Jaturawat, “An Evaluation of Technical

Study and Performance for Real-Time Face Detection Using Web

Real-Time Communication,”, no. I4ct, pp. 162–166, 2015.

[3] G. Carullo, M. Tambasco, M. Di Mauro, and M. Longo, “A

Performance Evaluation of WebRTC over LTE,” in 12th Annual

Conference on Wireless On-demand Network Systems and Services

(WONS), pp. 170–175, 2016.

[4] L. O. D. Nedberg, “Quality of Experience of WebRTC based video

communication Eirik Fosser,” Norwegian University of Science and

Technology, 2016.

[5] Sam Dutton, “Getting Started with WebRTC,” 2014.

[6] C. Cola and H. Valean, “On multi-user web conference using

WebRTC,” in18th International Conference on System Theory,

Control and Computing, ICSTCC, pp. 430–433, 2014.

[7] C. Y. Chiang, Y. L. Chen, P. S. Tsai, and S. M. Yuan, “A video

conferencing system based on WebRTC for seniors,” in

Proceedings - 1st International Conference on Trustworthy Systems

and Their Applications, TSA, pp. 51–56, 2014.

[8] I. T. Management, “WebRTC in the Enterprise,” 2016.

[9] H. Rahaman, “A Survey on Real-Time Communication for Web,”

vol. III, no. Vii, pp. 39–45, 2015.

[10] M. Schindler, C. Von Harscher, J. Kinzig, and S. Alekseev,

“Evaluating Framework for Monitoring and Analyzing WebRTC

Peer-to-Peer Applications,”, no. Inc, pp. 171–175, 2016.

[11] M. Deshpande, “Integration of WebRTC with SIP – Current

Trends,” Int. J. Innov. Eng. Technol. Integr., vol. 6, no. 2, pp. 92–

96, 2015.

[12] Schahin Rajab, “Comparing different network topologies for

WebRTC conferencing,” 2015.

[13] S. a S. T. Miner, “Getting Started with,” Packt>, pp. 1–41, 2013.

[14] R. Rai, Socket. IO Real-time Web Application Development.

BIRMINGHAM - MUMBAI: PACKT, 2013.

[15] B. Sredojev, D. Samardzija, and D. Posarac, “WebRTC technology

overview and signaling solution design and implementation,” in

38th International Convention on Information and Communication

Technology, Electronics and Microelectronics, MIPROProceedings,

no. May, pp. 1006–1009, 2015.

[16] C. Notice and A. Notice, “WebRTC to complement IP

Communication Services,” 2016.

[17] N. M. Edan, A. Al-Sherbaz, S. Turner, and S. Ajit, “Performance

evaluation of QoS using SIP & IAX2 VVoIP protocols with

CODECS,” in Proceedings of SAI Computing Conference, SAI, pp.

631–636, 2016.

[18] M. Grinberg, “socketio Documentation,” 2016.

[19] D. C. B. Adam Bergkvist, B. A. Cullen Jennings, Anant Narayanan,

and B. and Taylor, “Real-time Communication Between Browsers,”

W3C, 2017. [Online]. Available: https://w3c.github.io/webrtc-pc/.

[Accessed: 30-Aug-2017].

[20] Ana Pol González, “DEFINITION OF A MENA OPINION SCORE

FOR VP8 OVER REAL-TIME CONNECTIONS,” Universida de

Vigo, 2017.