Real time chat web application with ECDH encryption, Digital Signature and perform Man in the middle attack.

J Component Project

Winter Semester 2022

Neel Rakesh Choksi

19BCE0990

B.Tech. Computer Science and Engineering

Amara Hemanth Kumar

19BCT0126

B.Tech. Computer Science and Engineering with Specialization in IoT

School of Computer Science and Engineering

Vellore Institute of Technology

Vellore

April, 2022

Abstract

Every web application consists of a real time component which enables the user to either get notifications in real time or get and send messages in real time . This is done using a socket server connection and web sockets . The sockets server might be vulnerable to Man in the middle attack at the application layer as well as the ethernet layer(data link and network layer) .Our project is aimed at implementing a real time web chat application using http server and web socket server and performing man in the middle attack at the application layer and at the ethernet layer. We have proposed a solution for end to end secure connection of socket server with two clients. The solution is inspired by TLS 1.2 handshake. We have also proposed a solution for jwt integration for authorization of users in the http server.

Introduction

1. Overall idea about the project.

Development of a realtime chat web application using node js , react js and sockets.io. Including features such as login , authentication , authorization , registration , befriending other users, sending messages to the other user via a chat box. Commenting on user profiles of other users.

Encryption of messages would be done using a shared secret key generated using ECDH approach.

Testing only ECDH approach by performing man in the middle attack .

Encryption of messages then will be done using a shared secret key generated by our hybrid proposed procedure of using Digital Signature in the Elliptic Curve Diffie Hellman Key Exchange mechanism.

Testing our approach by performing man in the middle attack

Testing security of messages stored in the database using NoSQL injection attack and against XSS attack.

ECDH [18]

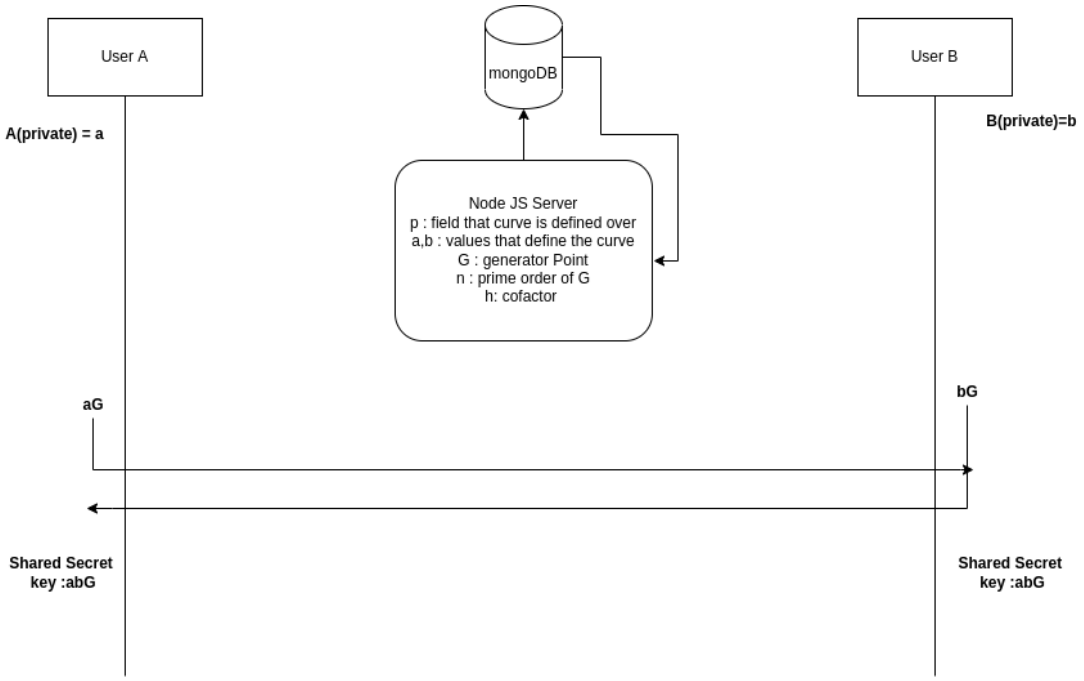


Figure 1. Elliptic Curve Diffie Hellman Key Exchange.

User A will be having the private key “a” and user B will be having the private key b . The curve used for Elliptic Curve Cryptography would be $y^2 + axy + by = x^3 + cx^2 + dx + g$. Addition would be performed “a” times on the random point on the curve selected by user A. The new point will be having $x_A y_A$ which would be transferred to the user B. User B will be having a private key “b” and a parameter $x_B y_B$ generated in the same way as User A. Both user A and user B would perform Addition with respect to the curve with their parameters and the received parameters and generate the shared secret key between them. This shared secret key could be used to perform encryption on the message sent to and from between user A and user B.[18]

Man In The Middle Attack [17]

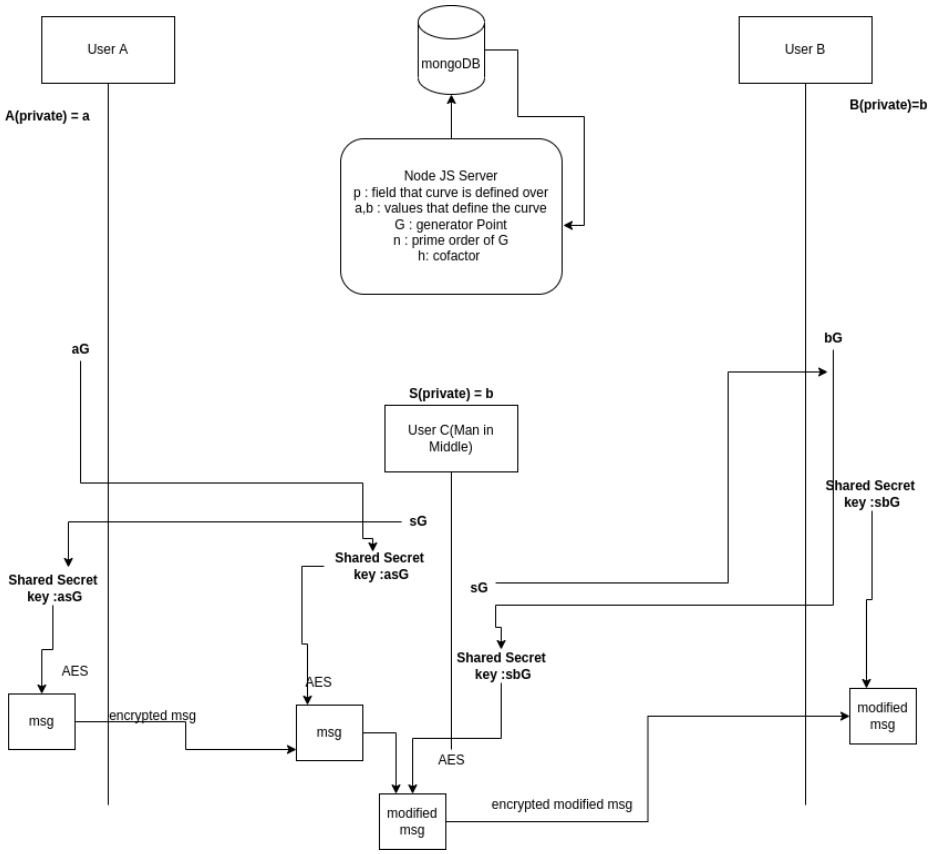
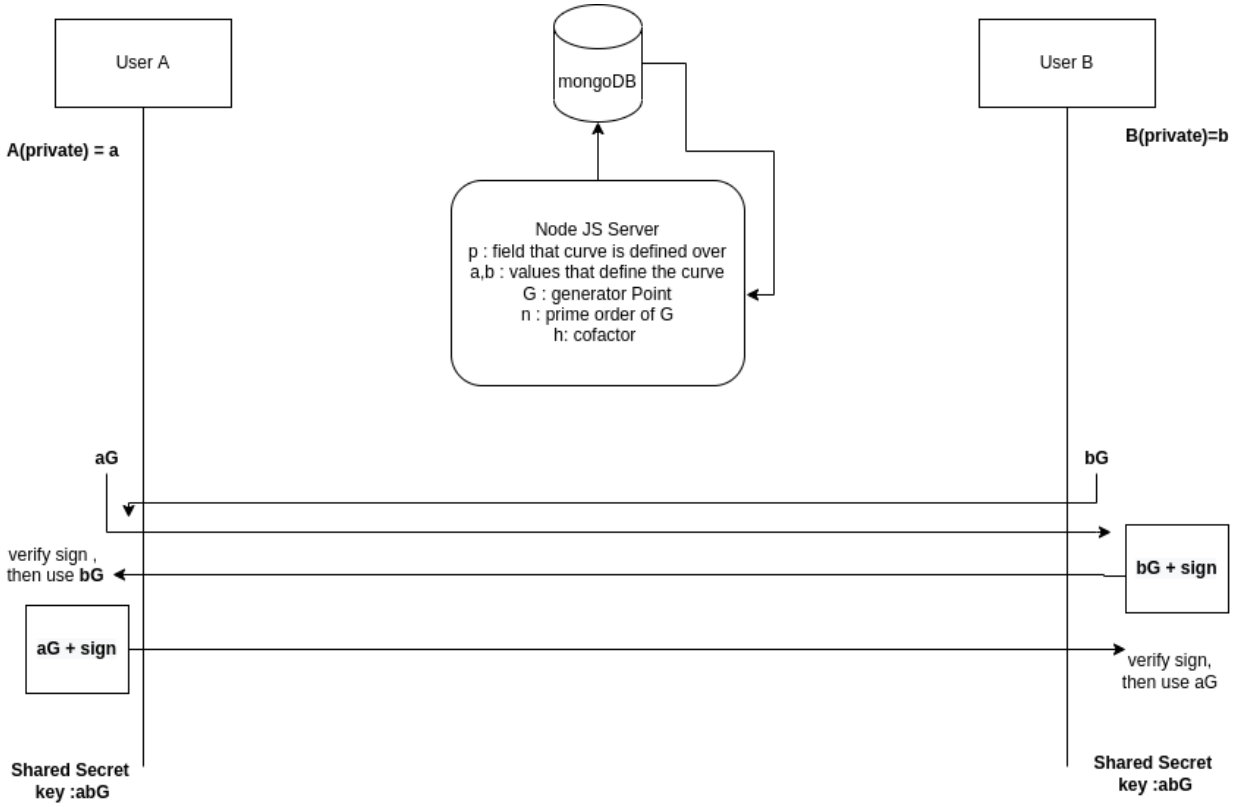


Figure 2. Man In The Middle Attack.

While performing Key Exchange , an attacker can interfere in the middle and form a shared secret key with User A as well as with User B and modify the messages sent by user A to user B and the other way around. [3]

Our Hybrid Approach of ECDH and Digital Signature



To prevent the man in the middle from tampering with the messages , the parameters could be signed by user B using its private key “b” and verified by user A using the public key of userB. While ECDSA algorithm already exists which uses digital signatures with elliptic curve cryptography to generate a shared secret between two users, we will attempt our own implementation of the algorithm by signing the parameters of B and getting it verified by A and signing the parameters of A and getting it verified by B. Our goal is to show that Elliptic Curve Cryptography with digital signatures is more secure than just generating a shared secret using Elliptic Curve Diffie Hellman Key Exchange by performing Man in the middle attack .

2. Background of the project.

ECDH with digital signature algorithm occurs at almost all the places in the web framework at the Transport Layer. But when it comes to application layer security for web applications , the packets transferred from the client to server then back to the client should also be encrypted , ensuring the security of the data of clients. Server should not be storing the client messages without encrypting them, since it becomes a vulnerable point wherein an attacker can access all the user information and data.

In this project we will be proposing and implementing a procedure to encrypt and store the messages safely in the server to provide privacy to the user. We will be using the server as a medium to store and provide the global parameters for shared key generation . Data will be encrypted at the application layer. When a user registers to a system, a private key and a public key will be generated for the user . Public key of all users will be stored in the server to transport

the messages. A private key will be generated at the user's end and stored locally in the browser. The private keys stored in the browser of the user will be used along with the public keys to generate a shared secret key . This shared secret key will be used to encrypt the messages using the AES algorithm . To test the application layer security of encryption using web application Penetration testing techniques . To Test hybrid(Digital signature with ECDH) key sharing procedure using Man in Middle.

3. Talk about the statistics related to the methods used.

Man in the middle attack performed between two users will involve a third user staying in between the userA and userB . There will be a shared secret key generated between user A and attacker , also between userB and attacker. The message sent by userA will be decrypted and modified by the attacker and sent to userB . Messages at userA and userB will be tested and compared to check the success rate of man in the middle attack.

NoSQL injection attack will involve sending a script in the input boxes of the web application in the login form , register form , comment section of user profile by an attacker.

XSS attack will involve sending a script in the comment section of the user profile by an attacker.

4. Advantages and disadvantages of the various methods along with your method.

ECDH

Advantage :

1. The key sizes are shorter than RSA , while providing the same level of security.
2. It is not possible to find the key from the global parameters since the global parameters only contain a part of the key.

Disadvantage :

1. It is vulnerable to man in the middle attack.

TLS Handshake [16]

Advantage :

1. It uses a shorter key size than RSA , with the same level of security.
2. It ensures that the data is encrypted while transported through the internet.Messages encrypted in the transportation medium therefore cannot be read when sniffed.

Disadvantage :

1. it does not cover securing the data after the data has been reached to the server. Application developed contains backend frameworks and databases storing data in plain text form which are vulnerable and need to be encrypted but cannot be ach

Our method - ECDH + Digital Signature at application layer

Advantage :

1. Tests the identity of the user therefore protects the confidentiality of the user and integrity of the message
2. Stores the messages in the database server in encrypted manner with keys with the users so only users can access the message

Disadvantage :

1. Application is still vulnerable to other attacks so the application security maintenance and updation according to latest vulnerabilities is necessary.

Literature Review

Basic Nosql Injection Analysis And Detection On Mongodb In this paper, they proposed the detection and basic attack of NoSQL injection. The main ideology of this paper is to make NoSQL programmers aware of the attacks which are possible and make them face to face to defeat the attack . It also detects those system or software features which are harmful for security . Most NOSQL injection attacks try to inject code at the input box. So, the solution is to sanitize the input before using it. Hackers try to inject malicious code either into input box or URL using GET or POST method. It can be injected by using PHP and Javascript code. 1 The most important role in a system is Information security . Developers should apply defense methods to make a secure system while writing code .By applying defense methods, user developers can prevent injection or any other attack. They proposed to make changes in the code that will accept only the limited numbers in the input. And then prevent the unwanted characters in the input. Most of the injected attacks can be removed by using the parameterized statement into the query. For example there is some code to find the character in a variable if it contains a number only or not, by checking the phone no .if yes, then it will pass the value otherwise rejected. [1]

An Open Tool Architecture for Security Testing of NoSQL-Based Applications In this paper , they proposed an open tool architecture which can take into consideration any NoSQL engines belonging to the four data stores categories whatever the programming language used. This tool is capable of detecting first vulnerable statements in the static mode on the developer side. Second, it also automatically detects injection attacks during run-time on the server side thanks to the added instrumenting statements during the first control (static mode). The tool is 2 sub tools 1 for developer side and other is server side. The lexical analyzer modules are used for parsing and detecting vulnerable NOSQL statements in the user application. Each table of operator tables contains the operators of a NOSQL engine that can be used in the injection and is used by the monitor methods for detecting injection attacks. Simply,In developer side If the statement includes any keyword then add an appropriate instrumented code before this statement - Go to next source code statement. On the server side, If the parsed code in execution is an

instrumented code then check if the argument of the next code includes any suspicious operator or value (If yes then stop the program and display: “Warning an attack is happening”) - Execute the code. Indeed and unfortunately, we are observing today the emergence of many new attacks which disturb a lot the proper functioning of NoSQL stores due to their exposure to the Internet users from many web applications.They proposed the easy adaptation of the proposed architecture to any NoSQL engine and/or any new injection attack as explained in section 4 makes it original, extensible and flexible.[2]

Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense The existing special characters, filtering of tages and maintaining a list of vulnerable sites etc cannot eliminate the XSS vulnerabilities completely. In this paper ,after testing and performing 2 attacks they proposed based on analytical results ,techniques of mitigating this XSS vulnerability by introducing a Sandbox environment on the web browser. XSS attacks steal the victim's cookies. So, they discuss an idea for the protection of cookies against XSS attacks by introducing a sandbox environment on the web browser which is a safe and isolated environment . Web server will generate the cookie and send this cookie to the client's web browser which is sandbox protected. Now this cookie value will neither leak into the windows nor it can be grabbed by any attacker. Sandbox executes a malicious script but it cannot give the authority to simply leak the cookie out of this protected environment. So it can be concluded that the sandbox environment will simply bye-pass the XSS attack. They made all the attacks and tested in order to bypass the vulnerabilities of XSS attack in the sandbox under some precautions . [3]

Detection of XSS in web applications using Machine Learning Classifiers In this paper , investigates the use of machine learning to build classifiers to allow the detection of XSS. They targeted detection via URLs and Javascript. To predict the level of XSS by using 4 machine learning algorithms -SVM, Random forest, KNN and Logistic Regression. Since by testing ,Random Forest classifier is best . This precision will ensure a method much more efficient to evaluate threatening XSS for the smooth functioning of the system. Based on the experiment processed an investigation of XSS attacks on various web applications involving two analyses. 1. we collected multiple URLs and extracted every minuscule of its characteristics by processed extraction.2. These features extracted, helped the machine learning classifiers to entail a thorough observation of the malicious and benign URLs and JavaScript codes injected under each domain. Random forest Classifier has highest accuracy of 98% This approach involving the extracted features ensures the detection of malicious and benign scripts or URLs in any web page or application and can be used in real-time reference to detect attacked sites.[4]

MERN Stack Explained MERN Stack includes MongoDB as the database . MongoDB is a document database. Express.js is a Node.js framework to create API. React.js is the client side javascript framework used to develop modern GUIs. It involves fetching data from the API from the Node.js web server . Node.js is a web server that is asynchronous and event driven . It uses callbacks to handle multiple concurrent requests . React JS is the client side framework that helps in building the dynamic HTML pages. It fills the HTML pages with the data fetched from the

server using JavaScript. Express.js and Node.js are 3 exporting data stored in MongoDB. Models , views , controllers and routes are set up in the Node.js web server with express framework . Models are used to structure data to be stored in the mongodb, Routes help to map data with the URL . Controllers are the functions that are responsible to do operations on the data by fetching data or by updating the existing data or by adding new data or deleting the data after searching using a query. To abstract the mongodb drivers from Node.js , mongoose is used which is a ODM(Object Data Modeling) tool used in Node.js to perform operations on MongoDB databases. Mongodb database is a NoSQL document database. Data is stored in the form of JSON . It stored data in the form of documents and collections. Document is the main storage unit storing data in the form of JSON . Collections are a group of the same documents in the database. To execute relations between documents, either collections can be linked using Collection ID or sub documents can be made. [5] [6]

Full encryption: an end to end encryption mechanism in GaussDB. This paper aims at providing an end to end encryption approach to query data from GaussDB . It proposes methods to improve efficiency of query execution and availability of data using cryptographic algorithms and Trusted Execution Environments. The full encryption solution of the authors combines software , hardware modes using data encryption schemes with Trusted Execution Environment(TEE). The full encryption architecture uses two types of components for query processing , TEE or REE based on the functional partitioning and cost evaluation. The sensitive data is stored in column level granularity in a gauss db server with encrypted key. Client encryption driver 4 encrypts the data and sends it to the server in cipher text format . The keys are managed and logged separately. The model proposed by the authors is tested to query in the three scenarios including storing data in the cloud, bank application , e-business. [7]

PICADOR: End-to-end encrypted Publish–Subscribe information distribution with proxy re-encryption. The goal of the paper is to propose a Publish – Subscribe end to end encryption model which uses proxy re-encryption and eliminates the sharing of encryption and decryption keys. Multiple Publishers can post their information to Pub/Sub brokers which will use proxy re-encryption and make the information available only to subscribers. The broker is unable to read the information sent to it from the Publisher. The broker uses proxy re-encryption to send the data received from the publisher to all its subscribers without the subscribers ever handling their keys with the publishers. If Alice is the publisher and Bob is the subscriber , the system will work in the following manner. A policy authority will generate a public and a secret key for Alice. Bob will generate a public ,private key and send the public key to policy authority. The policy authority will regenerate the keys for brokers using a secret key of Alice and public key of Bob. Alice will encrypt the data using her public key and send the cipher text to the broker , where re-encryption of the cipher text will take place using regenerated keys for the broker. Finally that re-encrypted cipher text will be sent to Bob and he will be able to decrypt it using his secret key. The proposed architecture makes sure that the decrypted and unprotected data in the channel while sending to the broker and while sending to the subscribers , cannot be accessed. The model uses RSA signing or ECDSA to maintain integrity, Clustering to maintain availability

and Policy Authority centric key authentication to maintain authenticity of the keys transferred.[8]

Websocket to Support Real Time Smart Home Applications In this paper , the authors have used the websockets and polling method to transfer data from a sensor to the monitoring device.The architecture used by the authors involve a MySQL database, php server, web browser. The php server interfaces the connection between the smart home and the web browser. Polling in HTTP involves the client sending continuous requests , repeating in a duration of time to the server using HTTP protocol . The server responds to each HTTP request sent by the client. There is a waiting time between the request sent by the client and the response sent by the server.The server responds with the new message if the message is updated at the server side.In websockets protocol , the client and server establish a connection at first and maintain the connection for a fixed interval of time. During the interval of the connection , any updates in the data are notified to both the parties without implementing the 5 HTTP protocol from the beginning . The update in data could be from any of the two parties .Websockets have proven to be more effective than polling methods.[9]

Introduction to socket.io It is a library that is used to make real time bi directional connections between the browser and the server , that are event based . It has a Node.js server implementation and a javascript library for the client side. WebSocket communication protocol is used between the client and the server. WebSocket protocol allows the user to establish a full duplex , low latency channel between the client and server. At the server end, a socket is created , which is a constant connection between the client and server. Different types of events can be associated with the socket which defines actions to be taken by the server when a client request is sent. Some of the instances when an event is triggered are sending a message, connecting to the server. It does not directly use the WebSockets, it adds some metadata to the packets so the server and client both should use sockets.io libraries. Reliability is offered by this library by falling back to the polling method in case the Websocket connection cannot be established. [10]

Evaluation of web application security risks and secure design patterns Writing secure web application code and testing it is a challenging task .In this paper the author has defined security design patterns for web applications which help the web application developers to implement security in their applications at different stages. The types of scenarios covered in the paper include Software engineering involving Software Architecture , Management of computing and Information systems involving Security and protection , authentication and unauthorized access.Vulnerabilities in these areas have been reviewed and patterns to protect the web application form these vulnerabilities has been proposed.Cryptographic design, Authentication , Secure loggers, Single Access Points, Privacy and access control patterns are reviewed by the author.Based on the patterns the web application security risks are divided into Technical , configurational , security. Technical vulnerabilities cause attacks like XSS , Injection. Configurational Vulnerabilities involve encryption flaw, broken authentication and session management , server misconfiguration , information leakage, insufficient authentication. Security

vulnerabilities are due to flaws in transport , network layer and causes Manin in the middle and Denial of service attacks. [11]

Eye on the End User: Application Layer Security Every level of the application should be audited and be protected against common vulnerabilities. Common threats in the application layer where the users interact with the system and methods to mitigate them are discussed in this article. Application layer is the 7th layer in the OSI model. Application is most vulnerable because it is open to all users, whereas layers below it are accessed by developers and users that are conscious about security. DDos attacks , HTTP floods, SQL injections , Cross-site scripting are the most common threats . These attacks focus on imitating legitimate users to flood the application with traffic or to inject malicious code to retrieve sensitive information from the system.Mitigations involve guarding the input form the users since most of the web applications are a victim to SQL injection.Monitoring user activity , connection and network loads helps prevent DDoS attacks . [12]

A3:2017-Sensitive Data Exposure Sensitive Data Exposure is one of the top 10 vulnerabilities classified and ranked by the OWASP organization. The attackers steal the keys and execute man in the middle attacks instead of attacking directly . Since they are the man in the middle , the system identifies them as legitimate users. These attackers steal information off the server while it is being transmitted to the client.GPUs are used to brute force the passwords of the users. The weakness is to not encrypt the sensitive information stored in the server. Weak key generation and management leads to stealing of keys. Weak passwords tend to easily brute force by the GPUs. This leads to loss of sensitive information including Health records , credentials, credit cards, personal data . This can be prevented by using strong salted hashing functions to hash passwords, to encrypt the data in the Presentation Layer using TLS protocol . HTTPS should be used in the application layer to protect the data. [13]

Efficient Online-friendly Two-Party ECDSA Signature. ECDSA is applied between two parties in two phases including an online phase and offline phase where the online phase involves decryption of cipher text and the offline phase involves more executions of multiplicative inverse . This paper suggests an approach which enables lightweight online friendly two party ECDSA online phase. The proposed solution includes linear sharing of nonce and resharing of secret keys. Threshold signing is a process where the message is signed by a set number of parties and then it is verified. To achieve a distributed message signing technique , multiplicative-to- addition functionality has to be achieved. The MtA is quite computationally intensive and is done more than once in various systems . The computation is divided into online and offline in many 7 systems. This paper proposes a method to optimally compute the online phase of the multiple signing system and execute MtA only once in offline phase. The proposed two party 2 ECDSA has faster online computation and offline phase requires single execution of MtA [14]

Multiple Handshakes Security of TLS 1.3 Candidates. Transport Layer Security protocol is widely deployed over many networks for securing communications. This paper suggests treatment of multiple handshakes protocols in TLS 1.3 by providing a multilevel and stage

security model to provide security guarantees. TLS is the successor of the SSL protocol in the transport layer. It has two major operations including TLS handshake and TLS record protocol . The handshake establishes a connection between the client, server and conducts generation and exchanges of the required keys for encryption and decryption. TLS record protocol is to protect the application data. TLS handshakes occur in three modes including full handshake , resumption , renegotiation , a man in the middle attack performed by a malicious server was able to impersonate the client in the honest client server architecture. The authors of the paper have suggested various TLS handshake protocols which cover all the compositional interaction between various TLS handshake modes. TLS handshakes happen over multiple connections and correct implementation of the multiple handshake protocol is shown. [15]

Preventing Man-In-The-Middle Attack in Diffie-Hellman Key Exchange Protocol. This paper aims at showing how the Diffie Hellman key exchange is vulnerable to man in the middle attack , and proposes a defense mechanism for securing it using Geffe generation of binary sequences to hash and encrypt the data. The authors of the paper have suggested an encryption system that takes in the eight character password and converts it into binary sequence of the password using ASCII code .This sequence is randomized using the geffe random generator which uses 3 registers . There are statistical tests performed on the newly generated sequence. The user will be asked to retype a new password if the statistical tests give negative results. The private keys for both ends are generated using this method. The server hashes the passwords , adding salt and using an admin key and stores it in the database. During login , both parties are redirected to another page where public key parameters are provided to them .Both parties now use this information and generate a shared secret key . The shared key is hashed and sent to the server, the server verifies if both hashes are the same , if yes then it lets the users interact further confirming their identities and ensuring , no party is performing a man in the middle attack. Statistical tests carried out to test the randomness of the generated keys using the proposed method, insurance of private key storage in the server in hashed format.[16]

Conversion of the Diffie-Hellman Key Exchange Algorithm Based on Elliptic Curve Equations to Elliptic Curve Equations with Private Parameters. Key sizes in public key cryptographic systems are significantly reduced by Elliptic curves but to make it more stable , this article presents a method to convert the elliptic curve equations to hidden parameter equations rather than increasing the key sizes. Elliptic Curve Cryptography has a discrete log problem and uses the properties of an elliptic curve to generate public, private key pairs and generate a shared secret key between the two users.It is vulnerable to man in the middle attack since a third user can interfere the channel between two users and create shared secret key with both the users, furthermore the third user can transact messages on both sides and tamper them or log them into its database. The authors of the paper have suggested an addition in the hidden parameter in the diffie hellman key exchange algorithm to improve the security. The confidentiality parameter is R. The transition of original equations with the R parameter is shown by the authors , claiming that the confidentiality of the message will be maintained if the R parameter is integrated into the original ECDH algorithm. Hidden parameter R is added to the

Elliptic Curve Diffie Hellman algorithm and changes from discrete logarithm problem to degree parameter problem. Authors claim that since three is no method to select a R sequentially is not determined other than a strong attack , their proposed algorithm with hidden parameters is reliable. [17]

On the Adoption of the Elliptic Curve Digital Signature Algorithm (ECDSA) in DNSSEC. DNS runs parallel to HTTP in the Application layer , used to translate IP addresses of servers to the domain names . The packet sizes used in DNS protocol were increasing due to RSA digital signature. In this paper, the authors have proposed the Elliptic Curve Digital Signature Algorithm for authenticity , integrity of DNS data. The DNS has a critical vulnerability called cache poisoning which if exploited can redirect users to malicious websites. According to the review of authors, Elliptic Curve Cryptography leads to lesser key sizes than RSA hence improving the performance of the DNS. Adaptations of ECDSA in domains including .gov, .nl , . com , .net , .org , .com , .net, .org are identified and compared with their RSA implementations. These domains are managed by operators and the transition from RSA to ECDSA is complex since the operators have full fledged implementations of RSA . The adaptation should be tested to avoid technical vulnerabilities. [18]

Securing Digital Signature Algorithm against Side Channel Attacks. Although the Digital Signature Algorithm secures the channel , during the process of securing it is vulnerable to side channel attacks including timing , fault , power analysis attacks . Side channel attacks are a cryptanalysis technique. These attacks exploit the execution of the cryptographic algorithms with timing ,power , electromagnetic , fault , memory attacks. The key information is correlated to the non constant processing time for its generation. Power dissipated in the circuits during the generation of the keys is correlated to the key information. Electromagnetic Radiation emitted by the machines executing cryptographic algorithms are correlated to the key information. External force including radiations, lasers forces the machine to recompute the cryptographic operations. .Authors in the paper proposed an algorithm to safeguard against these attacks on ECC algorithm execution. [19]

Fault attack to the elliptic curve digital signature algorithm with multiple bit faults. The fault attacks aim at inducing errors during calculation of cryptographic primitives and exploit the secret keys generated. This paper proposed a fault attack methodology that will identify the secret keys by exploiting the information leaks induced when altering the execution of the modular arithmetic operations. The secret keys could be revealed by inducing single bit faults in the signature generation of the ECDSA . The authors have extended the attack by inducing multiple bits in a single datapath. Since there are multiple bits that are faulted, the information leaked will be less precise . Their framework analyzes the obtained erroneous information leaks and determines if it can be used to exploit the secret key. Since the fault attack methodology does not rely on the elliptic curve mathematical structure , it can be used against curves both in prime fields and binary fields. It is applied on an embedded platform to reveal secret keys which were securely stored , it shows that choice of elliptic curve parameters with greater security level does not bring any benefit to protect the secret keys since the methodology is not dependent on it. [20]

A Framework for Secure End-to-End Delivery of Messages in Publish/Subscribe Systems This paper presents a framework for secure end-to-end message delivery for distributed messaging infrastructures using publish/subscribe paradigm. It is made sure that the brokers distribute the messages only to authorized subscribers. A message contains topic information , message headers and the content payload. The content payload is made secure. A key management center is used to centrally allot and manage the keys involved in encryption between brokers , publishers and subscribers . The key management center is used to secure particular topics in the content payload. The secret key associated with the topic is distributed among the interested subscribers,the publisher encrypts 10 and signs the content payload and sends it to the broker. The authors have proposed a fine grained security approach over the Transport Layer Security. The broker can distribute the message to the subscribers , who in turn can verify the signature. Brokers can control the access of content from subscribers since they can enforce authorization rules, this is done to protect the content dissemination. Framework proposed is resistant to Denial of service attacks and replay attacks. [21]

Security and Privacy for Real Time Video Streaming Using Hierarchical Inner Product Encryption Based Publish-Subscribe Architecture This paper aims at proposing an architecture to use the publish / subscribe model to distribute real time video from multiple sources to multiple end users while protecting private information of the end users. Hierarchical Inner Product Encryption(HIPE) along with a Broker is used to provide end to end encryption to the users. The system ensures the users are only able to access data according to their role . The broker publishes the content to the user based on the encrypted topic. HIPE is used by the broker for data subscription , The topics are even hidden from the broker in this system. [22]

Threshold ECDSA from ECDSAAssumptions: The Multiparty Case," 2019 IEEE Symposium on Security and Privacy. Due to cryptocurrencies , threshold signing of ECDSA keys have gained more popularity. These threshold protocols involve signing key is shared among n parties , out of which a threshold of t have to sign to verify the message. Extension to the previous scheme where the threshold was limited to 2 is provided in this paper.The protocol involves a new multiparty functionality , replacing the key exchange functionality in the earlier 2 party threshold protocol using Instance Key Multiplication, Secret Key Multiplication , Consistency Checks and Signing. The protocol is evaluated among groups up to 256 , colocated in 128 geographic locations. The protocol outperforms all other protocols in LAN and WAN settings. [23]

Secure Two-party Threshold ECDSA from ECDSAAssumptions. It is difficult to implement threshold mechanisms in traditional ECDSA , custom protocols have to be implemented in order to achieve threshold key signing architecture.A custom protocol for key generation and signing for a threshold of two is suggested by the authors of the paper. Threshold signatures are used to create a joint authority over a message from more than one party. To implement threshold signatures in ECDSA, high computational power is required which affects the performance. The authors have proposed a threshold signing algorithm that is solely based on Elliptic Curves. The method includes using multiplicative shares of secret key and nonce to

achieve two secure multiplications over integer modulo of the ECDSA curve 11 parameters. This used to be done earlier using paillier additive homomorphic encryption but the authors propose a method to share the products , eliminating the need of homomorphic encryption. The two parties could jointly sign the document in just over 2 milliseconds. [24]

How Secure and Quick is QUIC? Provable Security and Performance Analyses QUIC protocol was implemented by Google in the Chrome browser in 2013. It was designed to minimize latency experienced by the traditional protocols.The authors of the paper have created a security model to analyze protocols like QUIC and attempted to attack the QUIC protocol by bit flipping and forcing it to fall back to traditional TLS handshakes or by failing proper connection between client and the server. QUIC uses two key and data exchange phases , whereas the other protocols just use open . QUIC is not built on top of TCP but implements some of its features, this is done to increase the speed of execution. The proposed security model captures the different levels of security guaranteed for the data encrypted between initial and final sessions. Furthermore they test if QUIC could provide forward secrecy as in the TLS-DHE model. [25]

Security issues with certificate authorities The State Of Security of the Internet currently relies on the TLS and certificate authority model . This paper shows the problems of certificate authority governance and the work done to mitigate the problems. To maintain authenticity , certificates are verified by third party Certificate Authorizers. The browsers rely on these parties to encrypt the data provided to their users. But these certificate authorities become a single point of failure and if they are using outdated protocols and algorithms which are crackable by the current computational infrastructure, the users are at risk . Also there have been incidents when these third party authorities have been impersonated by malicious parties , leading to data breaches. Untrustworthy Certificate Authorities, lack of accountability and transparency in certificates leads to security concerns.Public Key Pinning and Certificate Transparency technologies are emerging to solve the above issues. [26]

A new approach of elliptic curve Diffie-Hellman key exchange ECC (Elliptic Curve Cryptography) can be used for encryption , key exchange, and digital signatures.This paper proposes a new methodology to perform the elliptic curve diffie hellman key exchange. In their proposed method of ECDH , both Alice and Bob generate private keys in the same manner as in the standard manner. After the transmission of public keys by both parties, a 12 special public key for Bob is made by Alice integrating Bob’s public key and her private key .Same is done by Bob and these special public keys specific to Alice and Bob are exchanged. A secret key is generated by one party by selecting a random point on the Elliptic curve, this secret key is encrypted using Bob's public key and Alice's special public key that was sent to Bob by Alice. This way a shared secret is generated between the two parties. The protocol is faster since it has fewer operations.[27]

Overall Architecture

Level - 1 Architecture :

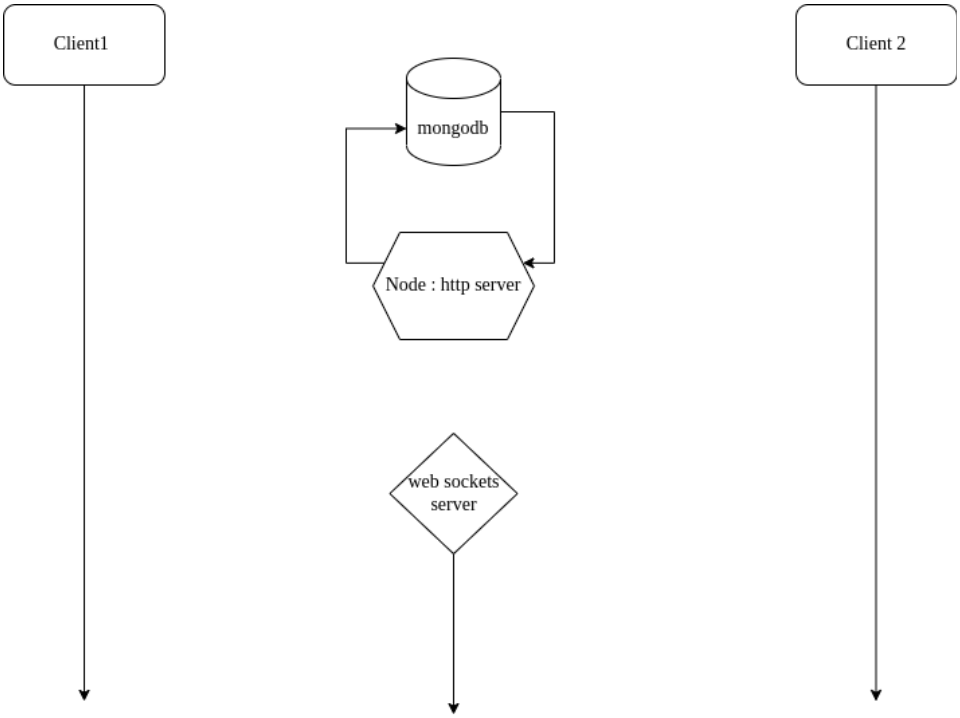


Figure 1. Overall Architecture of Real Time Chat Web Application.

Client1 and Client2 have opened the chat web application on their respective browser windows. The Node http server is connected to the mongodb database . The websockets server is used to establish a real time connection between the Client 1 and Client 2 for messaging.

Level - 2 Architectures :

Node http server and the mongodb

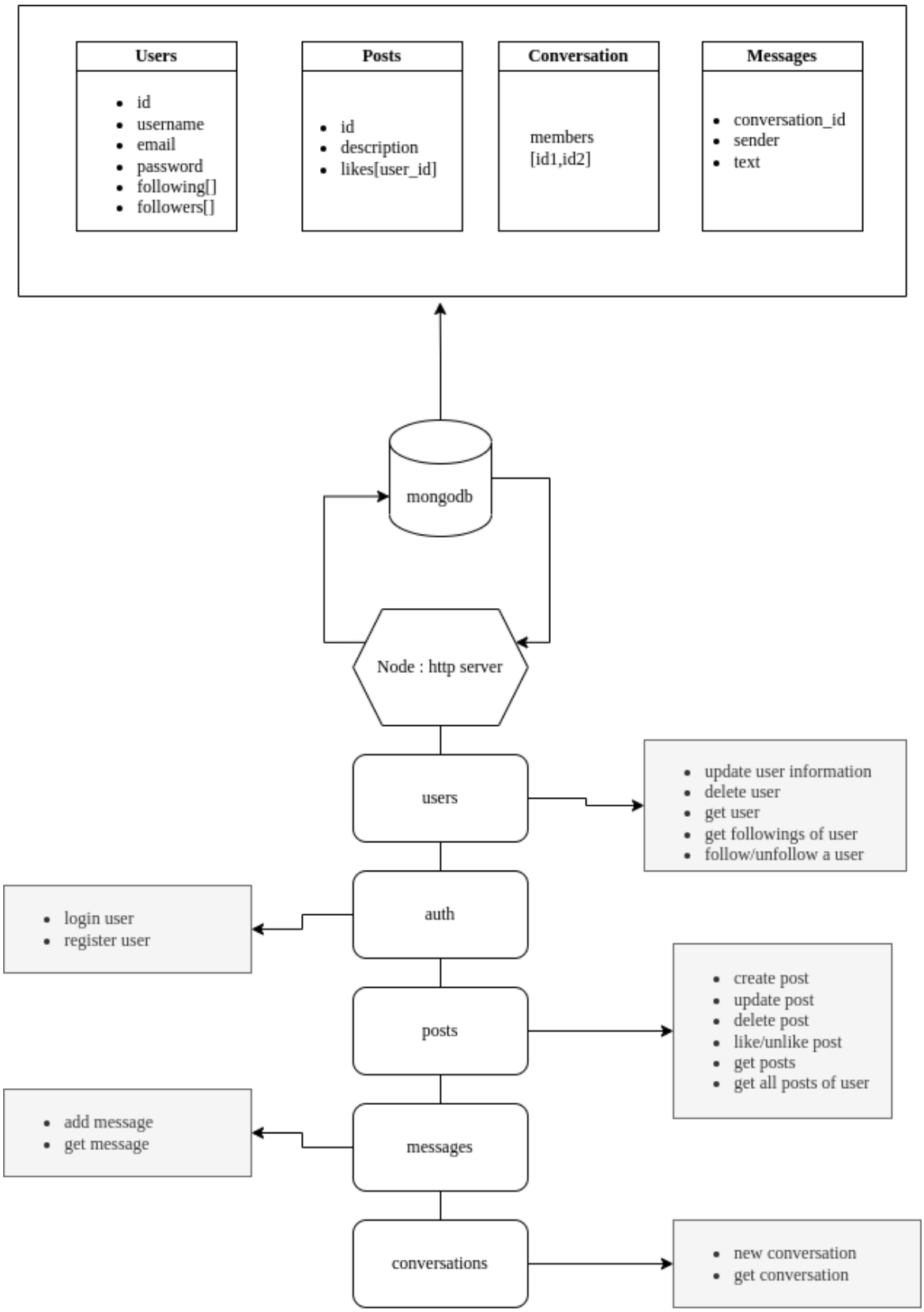


Figure 2. NodeJS Server API Endpoints and MongoDB Collections.

Mongodb collections include the users, posts, conversations and the messages. The node js API provides numerous endpoints to perform operations on the data stored in the collections. The users collection has a schema to store the username, email , password, followers , followings and personal information . The users endpoint enables the clients to update their profile information by providing the fields to update. Users can be deleted by providing a user id . A user can only delete his/her own account. The user can request the server to get the profile information of another user by entering userId or username in the get parameters. Other users that follow the current user can be accessed by the users endpoint. A user can be followed or unfollowed by another user.

The user can register into the system by providing the username , password and email id, a new user object will be created according to the mongodb user schema and will be stored in the database, the user object will be returned to the client. A user can login into the system by providing the email and the password.The server fetches the user object from the database corresponding to the email and password and returns the object to the client.

The user can post a post into the system by adding a description and sending it to the server, the server sends. The server will respond with the post and add it to the database collection . A user can follow another user and start a conversation with him/her. In the conversation tab, the user can send a message to another user, the message is transmitted to another user using the sockets server connection as well as by posting it to the mongodb database for fetching messages on revisit.

Client frontend application workflow

Client frontend is developed using React. The API requests are sent from the client based on the sequence of events occurring. Requests sent by the client are handled by the node js api and changes in the data in the mongodb are made . The requests are mainly made to Login , Register , View Profile ,Follow , Unfollow other people, Like unlike posts of friends,Manage Posts, Open messenger, Post chat to other user , get conversations of that user.

The user logs into the system by sending a login post request to the server from the client . The server responds with a user object which is stored in the client's browser local storage. The client can post using the post form , after posting , the client resends the get posts requests and retrieves the posts, this is done using context api in react js. The client connects to the socket server for real time communication. It can send messages using the message box and receive messages in real time using the socket server events.

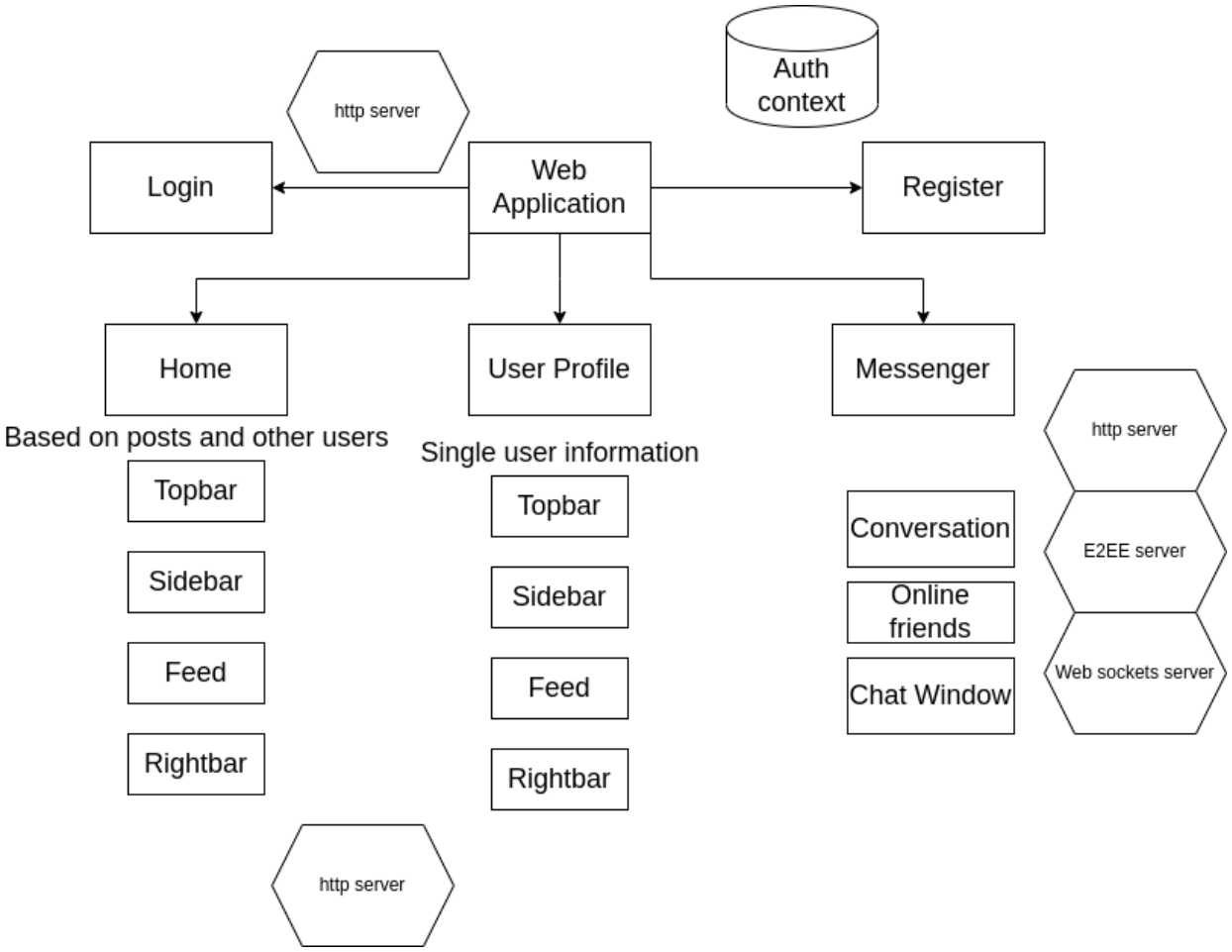


Figure 3. ReactJS Client GUI Components.

Web socket server connection for real time messaging

The client1 sends a request to the socket server to establish a sockets connection between the socket server and client1 once the clients click on the messenger icon. On establishing the connection ,it triggers an addUser event to get connected to the sockets server. The client2 follows the same process and both the clients get the users array which has the object of client user id with the socket id . The connection is maintained and both the clients respond to events triggered either by the sockets server or by the clients. To send a message , the client1 triggers a sendMessage event by providing the message text and the receiver id, to the sockets server which

triggers a getMessage event for client2. Responding to the getMessage event, the client2 receives the message from the sockets server.

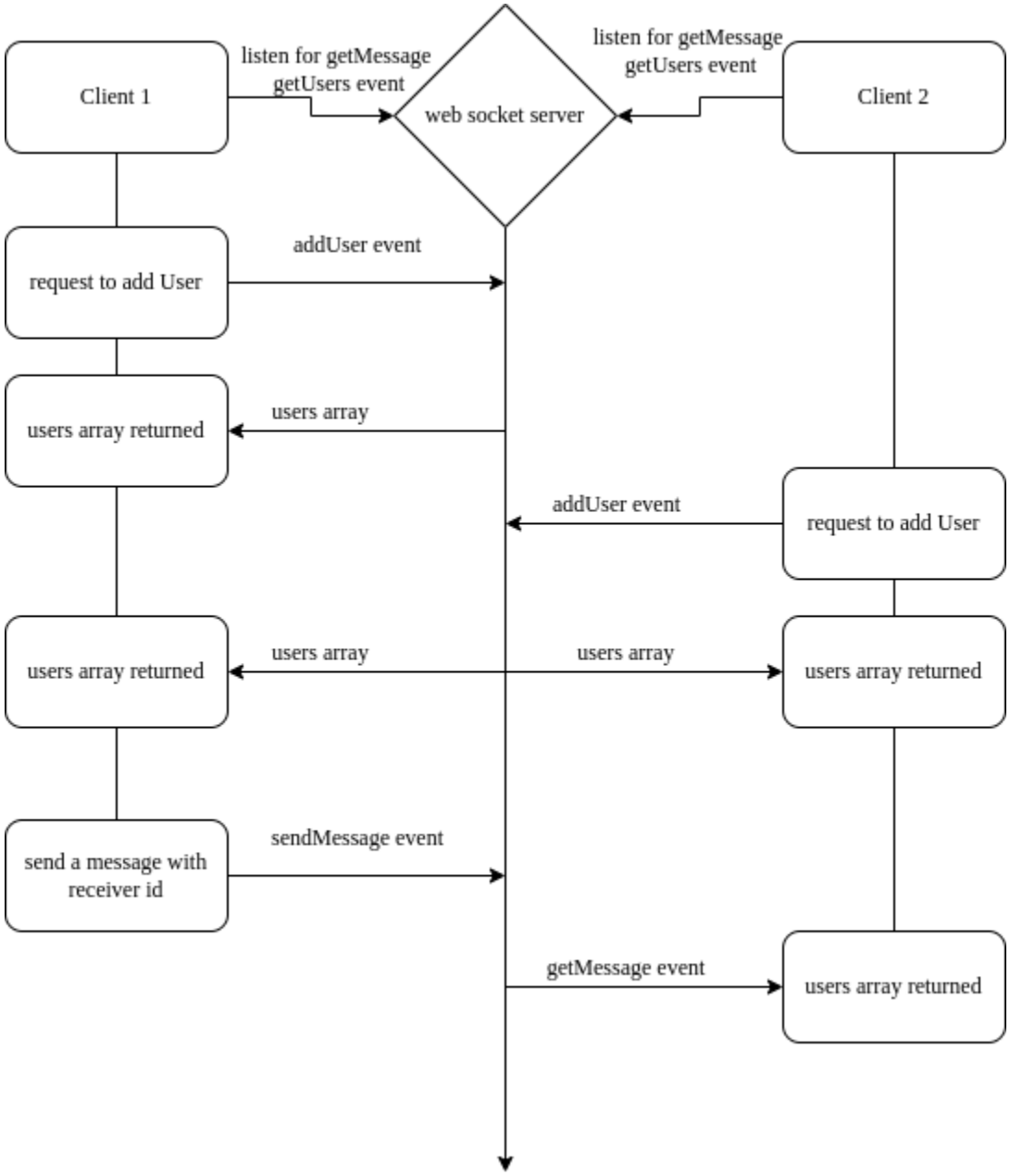


Figure 4. Web Socket Server connection with Client1 and Client2.

Deployment environment

Two client browsers with different accounts are opened in vm named lubuntu2 with IP address of 192.168.122.214 , sending a request to the host machine where the client code is hosted. The legitimate sockets server and the application layer man in the middle sockets server are hosted on the host machine on ports 8900 and 9900. The host machine(192.168.100.219) is

behind the virtual bridge (192.168.122.1). The kali linux vm is used to perform Man in the Middle attack , it has the ip address : 192.168.122.73.

Methodology

Man in the middle Attack between client and host

The man in the middle attack is performed by placing the kali VM in the middle of lubntu2 vm and the wireless access point, similarly placing kali VM between lubuntu20.04 and the access point. The message packets are intercepted by the kali vm . This is achieved by deceiving both lubuntu vms by arp poisoning.

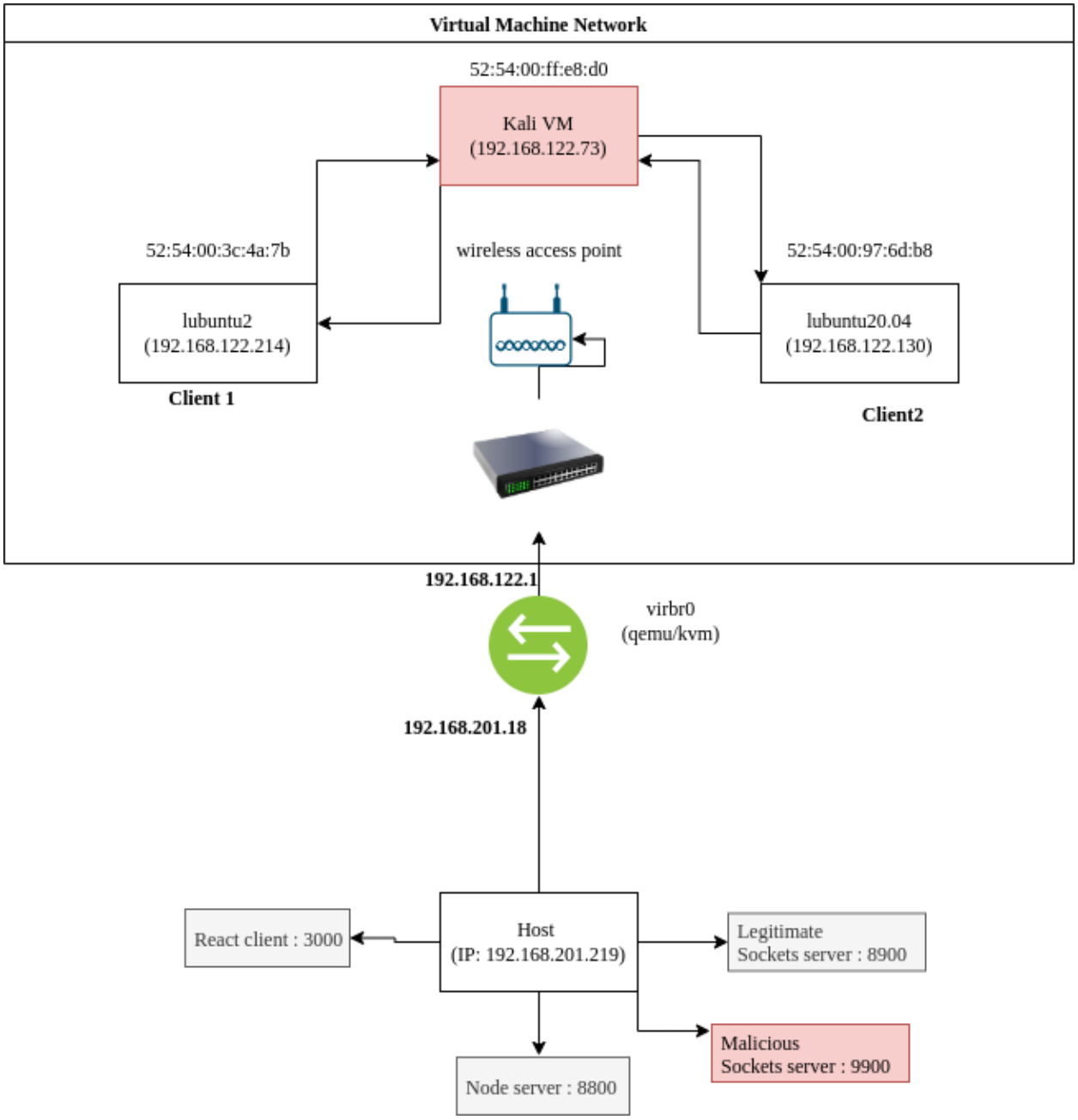


Figure 5. Deployment Network for NodeJS server, Malicious and Legitimate Server and Client server.

Man in the middle attack at application layer

Another type of man in the middle attack that can occur is that a penetration tester manages to replace the legitimate socket server by placing the malicious socket server in the middle. Now, the packets transferred from userA and userB are intercepted by the hacker, he/she may modify the message or just sniff the messages.Here the hacker has convinced the userA that

hacker is userB , and vice versa with userB , this is done by the hacker by generating a secret key each with user A as well as user B.

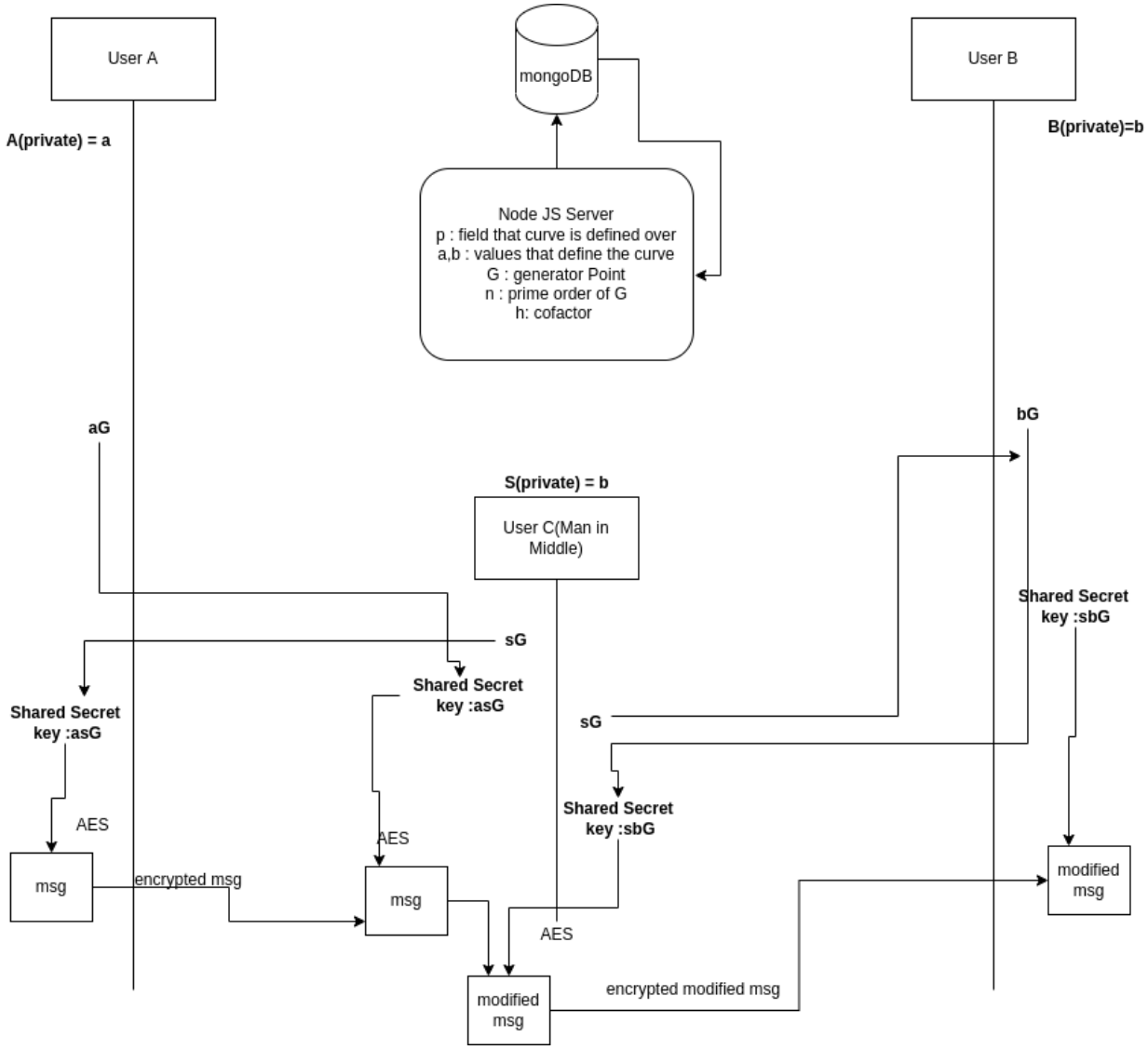


Figure 6. Man in the middle Attack on the Application Layer.

Proposed remedy for the man in the middle attack at the application layer inspired by TLS handshake.

To prevent the man in the middle attack at the application layer, a set of steps need to be performed that are inspired by TLS handshake. The client needs to send a key generation request to the encryption server. Once the clients receive their private and public keys, Client 1 can send a chat request to client2 ,this message is not encrypted . Client2 sends its public key to client1 along with the algorithm to be used for key exchange along with a digital signature signed by client2 using private key of client 2. The client1 can verify the digital signature of client2 by the public key of client2 . On successful verification , client1 sends its public key to client2 . At this

point , both clients have each others public keys and their own private keys. Using their private keys and public keys of the other client , they can generate a secret key which is a shared secret generated using the Elliptic Curve Cryptography. The exchange done earlier by the clients was a diffie hellman key exchange along with Digital Signature Algorithm Verification. After generation of the shared secret key , both clients will connect to the real time chat app web sockets server to exchange messages, these messages sent through the channel will now be encrypted and decrypted using the AES algorithm and the shared secret key. If a man in the middle attacker tries to intervene in the system , he will have to forward the digital signature done by client2 to client1 for verification . Also to be in the middle , he will have to give his public key to client1. The client1 will not be able to verify the digital signature of client2 using the public key of man in the middle attacker, hence this framework safeguards the end to end encryption scheme and protects it against the man in the middle attack.

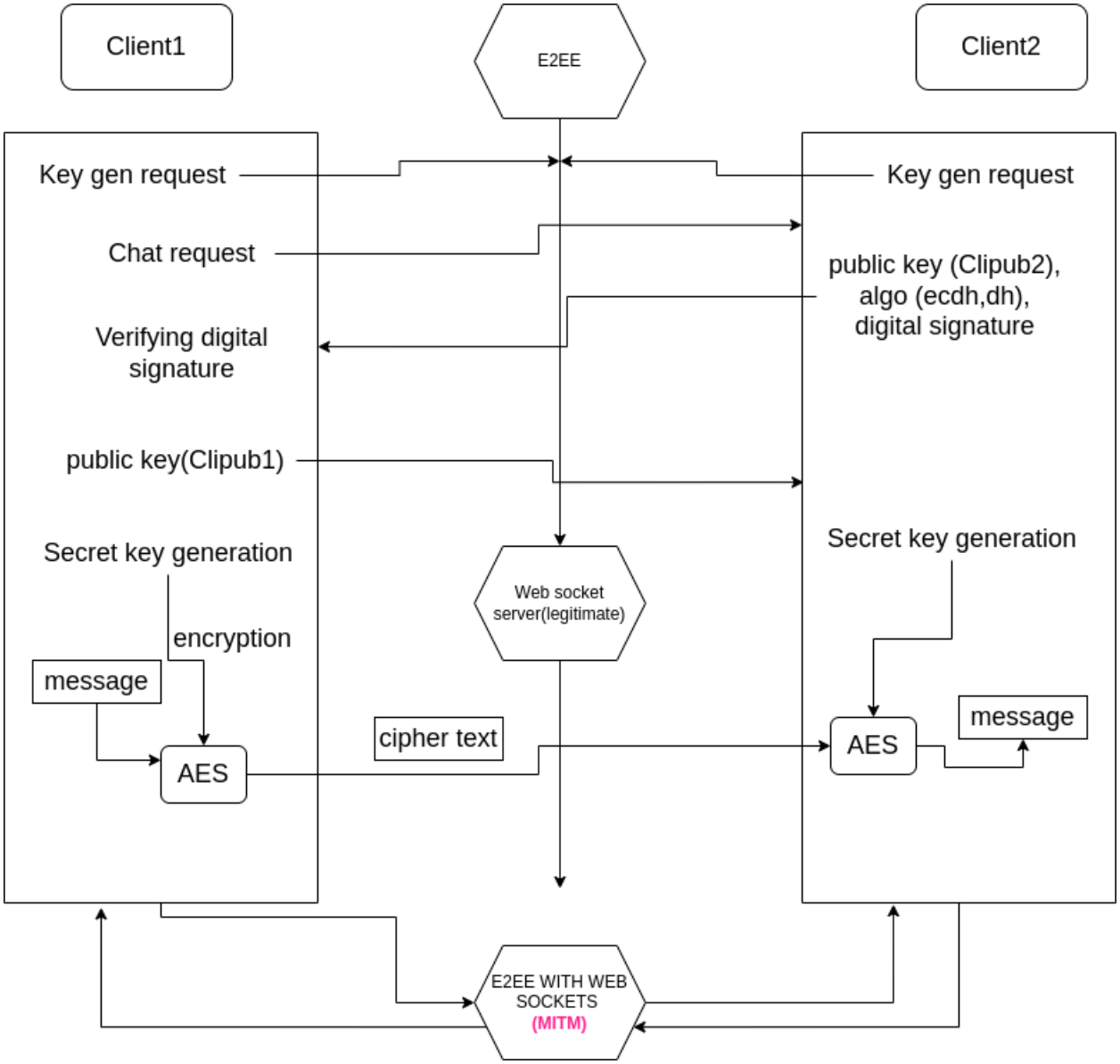


Figure 7. Security Mechanism for Socket Server proposed to prevent the Man In The Middle Attack at the Application Layer .

Proposed jwt authorization for the interaction of user with server.

The man in the middle attack depicted in Figure 5 involves a man in the middle attacker between the client and the host.Similarly , malicious users can send requests to the server on behalf of the user using the session variables of a user . To protect against such unauthorized requests, JSON Web tokens are used. JSON Web Token is created by the server on the first request to the server by the client. The token consists of the username and email address and other user data as payload, and the signature of the server . The token is sent to the user. After that , during the login session of the user, the user has to send the jwt token along with the createPost, likePost, createConversation , createMessage requests. Since the token has the signature of the server, it can verify it using the private key. The signature does not get verified if the data in the token is changed. The mechanisms of jwt token creation is shown in Figure 9. and the usage of jwt token in the existing system is shown in Figure 8.

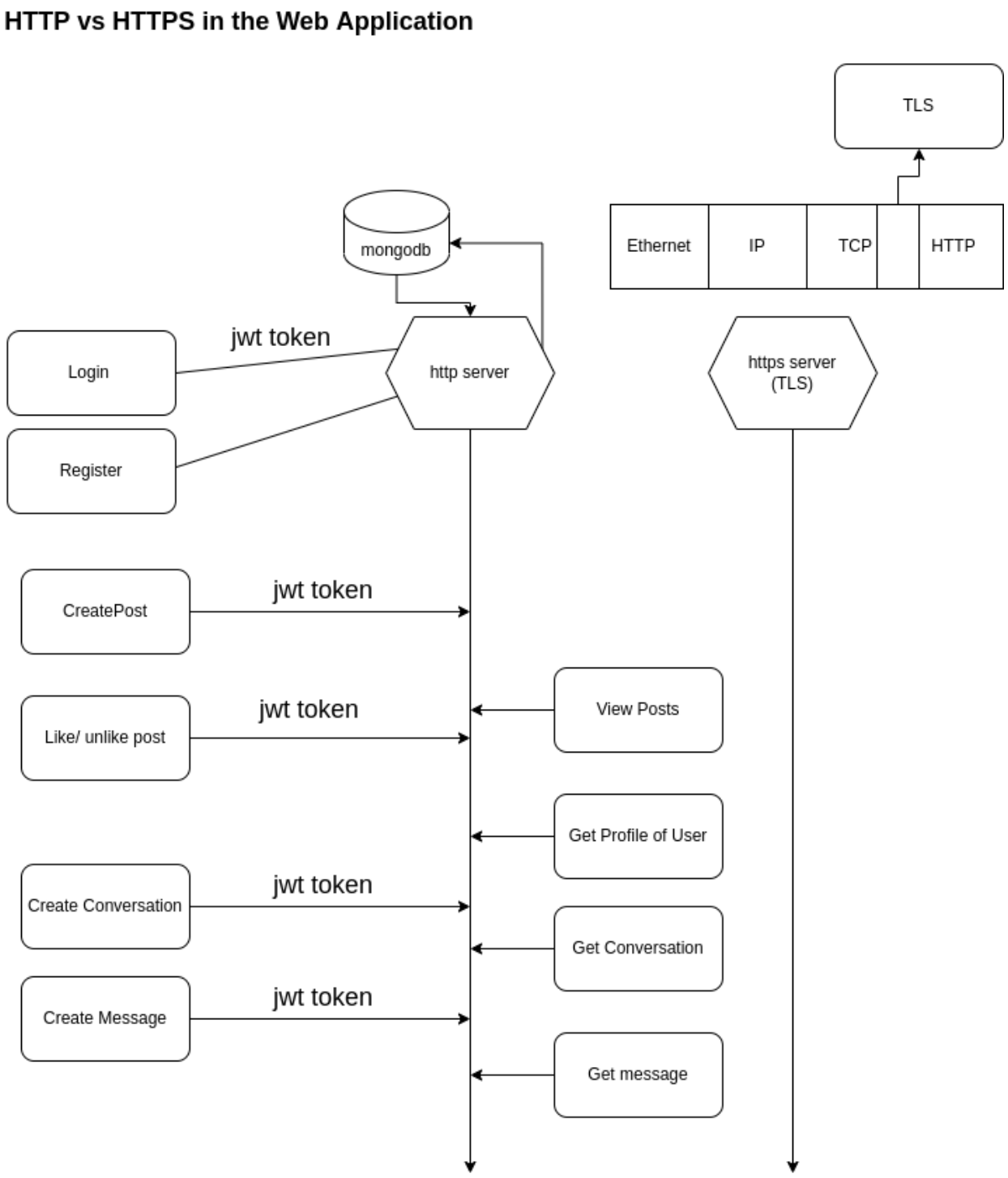


Figure 8. Authorizing User Requests at every Endpoint using JSON Web Tokens.

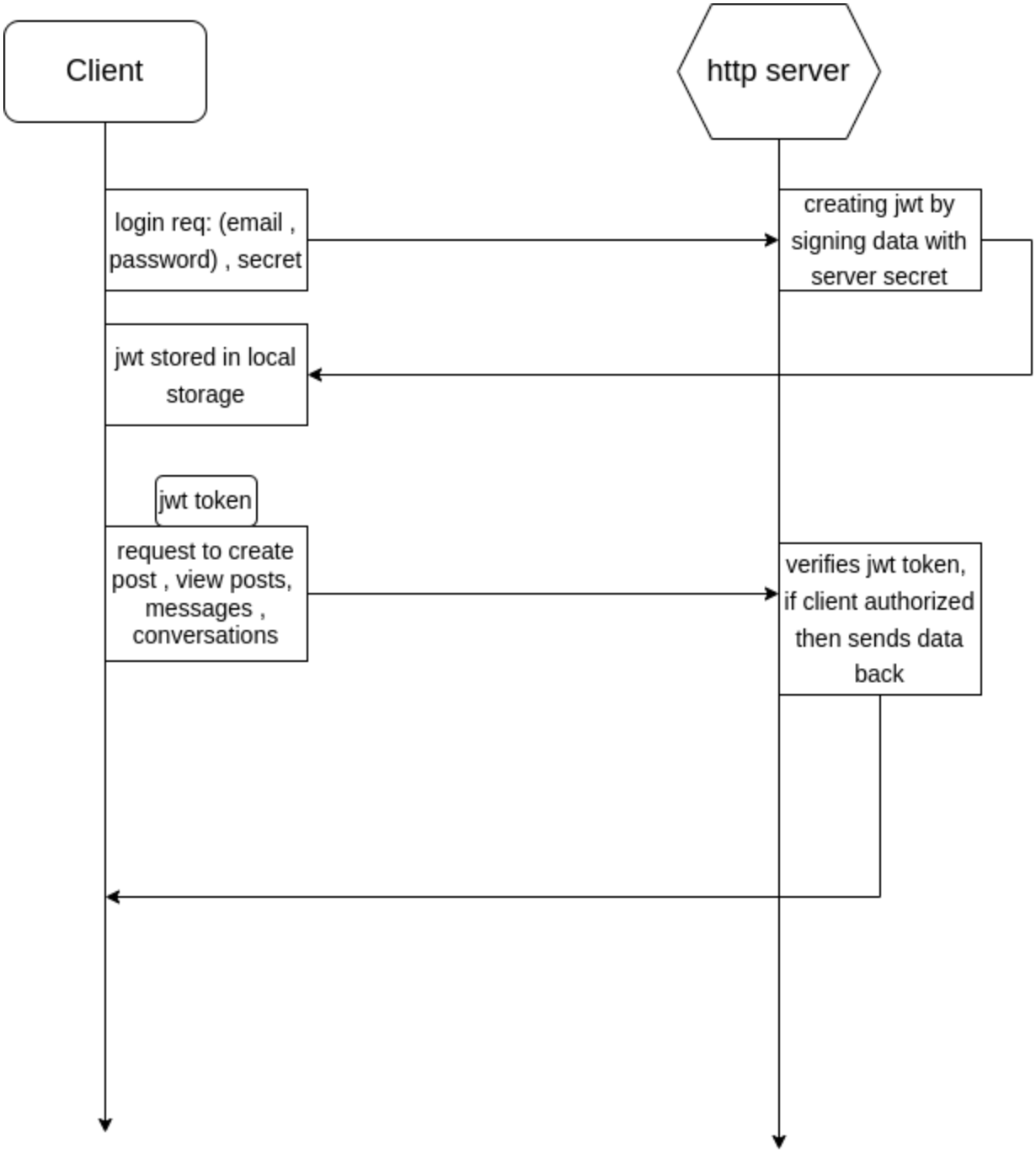


Figure 9. Working of JSON Web Tokens.

Node js http server attacks apart from MITM

NOSQL Injection

SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This

information may include any number of items, including sensitive company data, user lists or private customer details.

Sql Injection We Performed:

1. GENERIC SQL INJECTION

Commands:

‘ OR 1=1#

‘ OR 1=1/*

‘ OR ‘1’=’1

Phishing attacks

Phishing attacks are the practice of sending fraudulent communications that appear to come from a reputable source. It is usually done through email. The goal is to steal sensitive data like credit card and login information or to install malware on the victim’s machine. Phishing is a common type of cyber-attack that everyone should learn about in order to protect themselves.

Moreover, phishing is often used to gain a foothold in corporate or governmental networks as a part of a larger attack, such as an advanced persistent threat (APT) event. In this latter scenario, employees are compromised in order to bypass security perimeters, distribute malware inside a closed environment, or gain privileged access to secured data.

An organization succumbing to such an attack typically sustains severe financial losses in addition to declining market share, reputation, and consumer trust. Depending on scope, a phishing attempt might escalate into a security incident from which a business will have a difficult time recovering.

The following illustrates a common phishing scam attempt. A spoofed email ostensibly from myuniversity.edu is mass-distributed to as many faculty members as possible.The email claims that the user’s password is about to expire. Instructions are given to go to myuniversity.edu/renewal to renew their password within 24 hours.Several things can occur by clicking the link. For example,the user is redirected to myuniversity.edurenewal.com, a bogus page appearing exactly like the real renewal page, where both new and existing passwords are requested. The attacker, monitoring the page, hijacks the original password to gain access to secured areas on the university network.

Cross Site Scripting (Xss) Prevention:

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures.Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input.Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Results

Basic functionalities of chat app implemented using Node http server and the mongodb .The API is tested using Thunder Client to send requests to the server and get corresponding responses.

Users endpoint consists of functionality to update user information , delete user , get other users following the current user, follow or unfollow a user.

Thunder Client testing :

Register a user1 with username as client1 into the system by accessing the endpoint : http://192.168.201.219:8800/api/auth/register

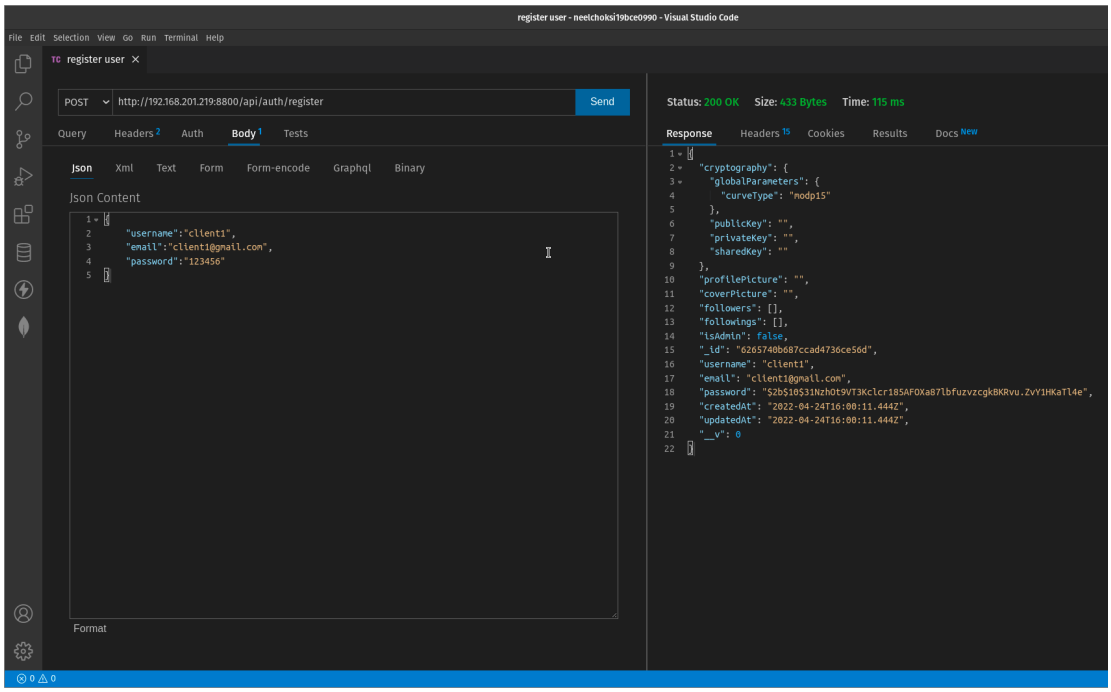


Figure 10. Register request to the server and response received.

client1 id : 6265740b687ccad4736ce56d

Registering another user named client2.

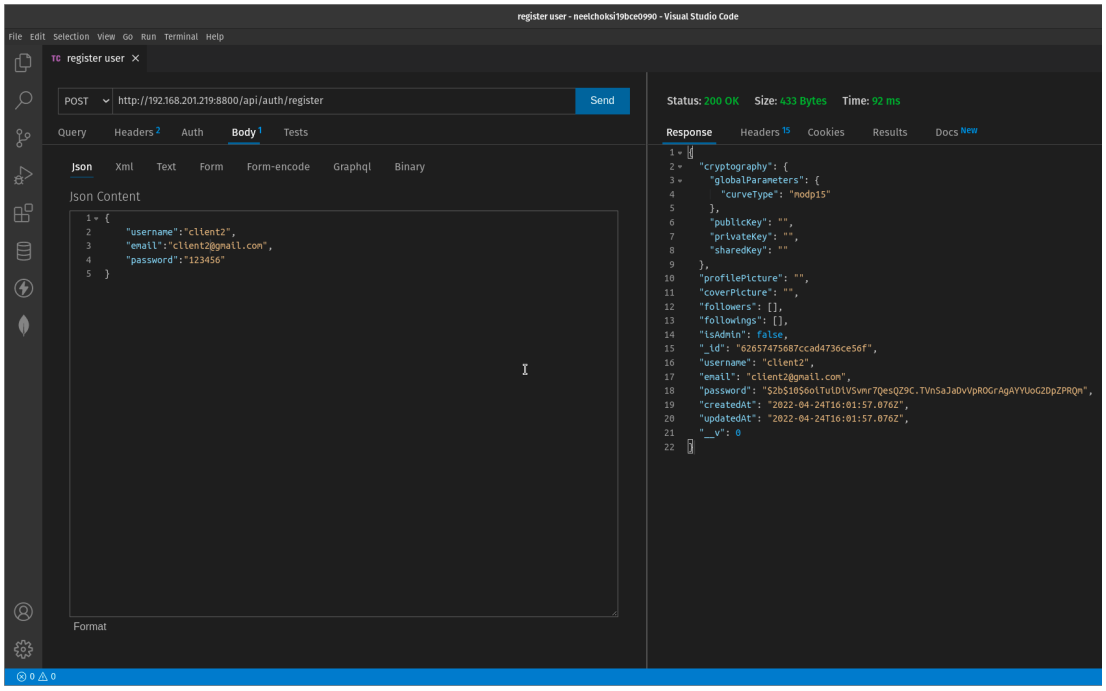


Figure 11. Register request to the server and response received.

client2 id : 62657475687ccad4736ce56f

Login a user: client1 into the system using the endpoint :

http://192.168.201.219:8800/api/auth/login

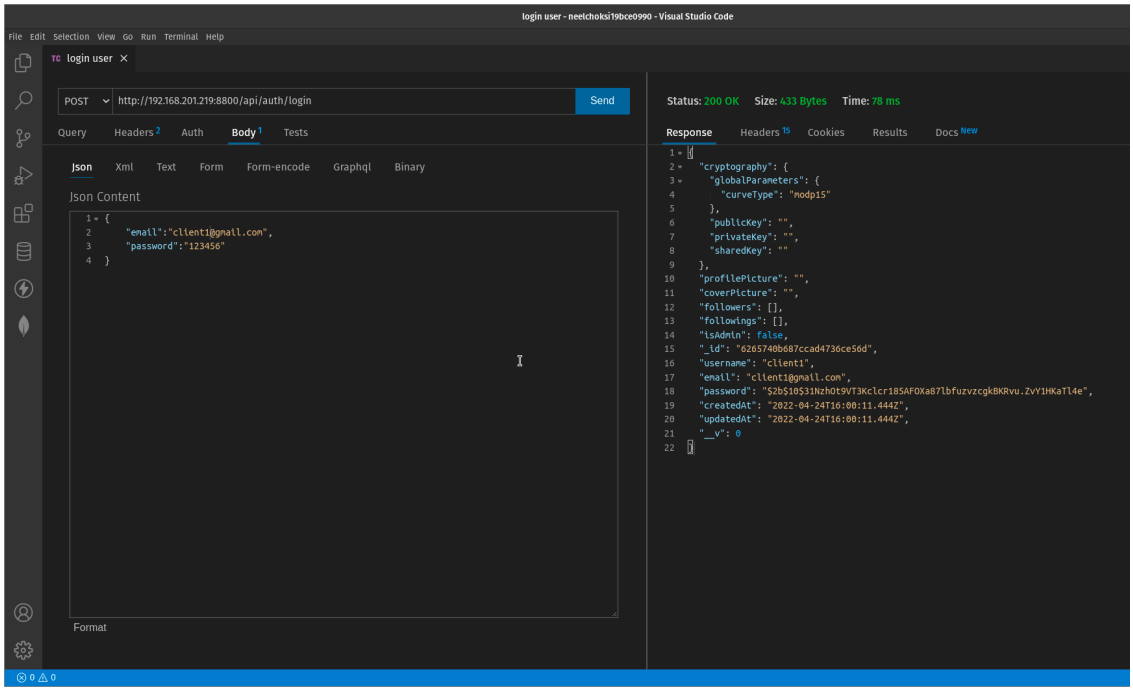


Figure 12. Login request to the server and response received.

Login a user client2 into the system

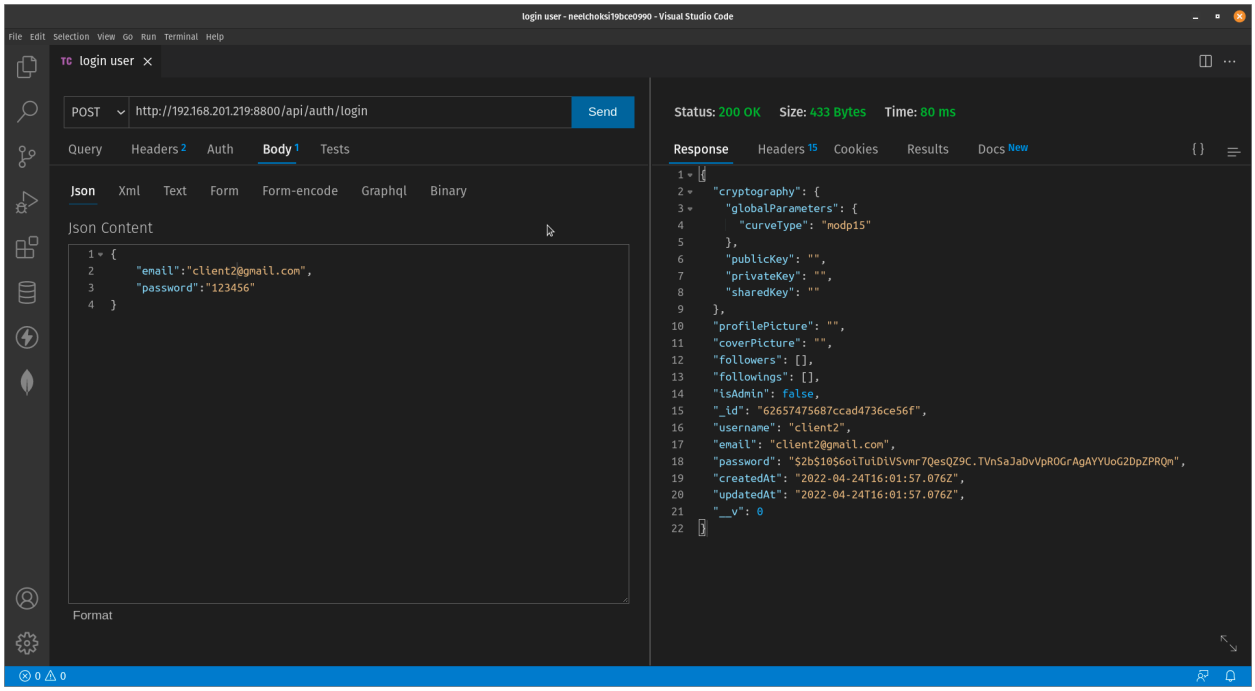


Figure 13. Login request to the server and response received.

Follow the client1 from client2 :
http://192.168.201.219:8800/api/users/6265740b687ccad4736ce56d/follow
client1 id is in the url and client2 id is sent along with the request body.

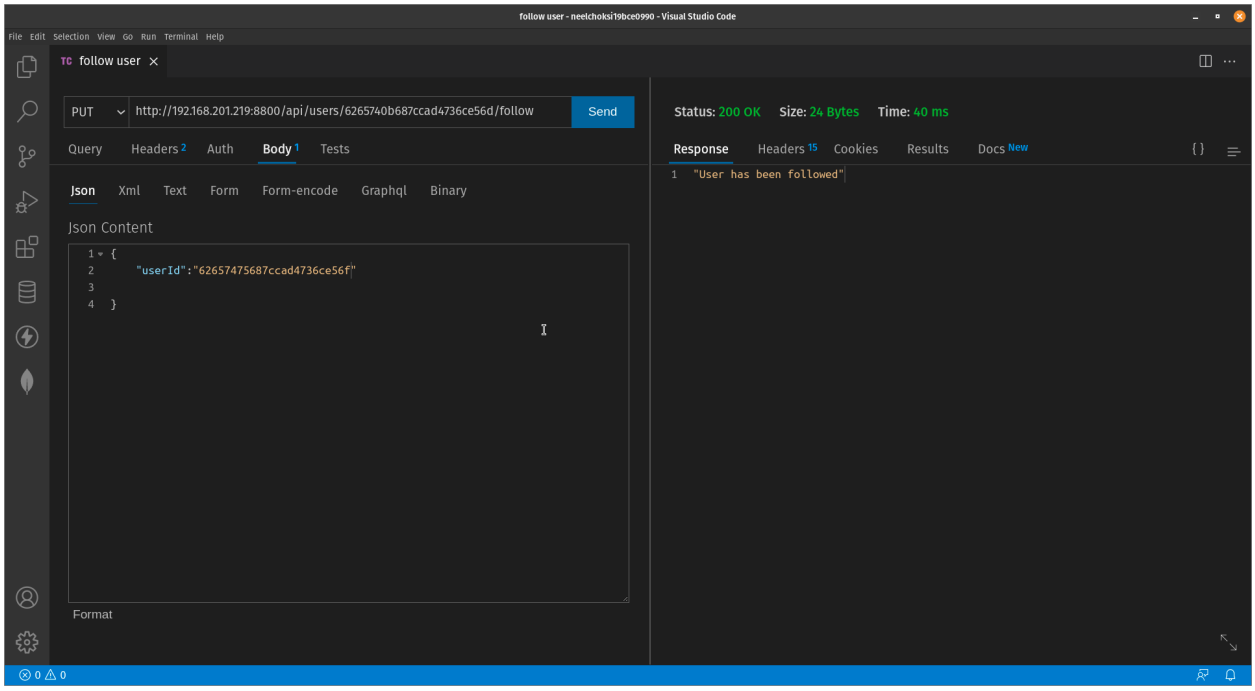


Figure 14. Follow request from client1 to client2 to the server and response received.

Create new conversation between the client1 and client2 :
http://192.168.201.219:8800/api/conversations
Client1 id is set as sender id and client2 id is set as receiver id in the request body.

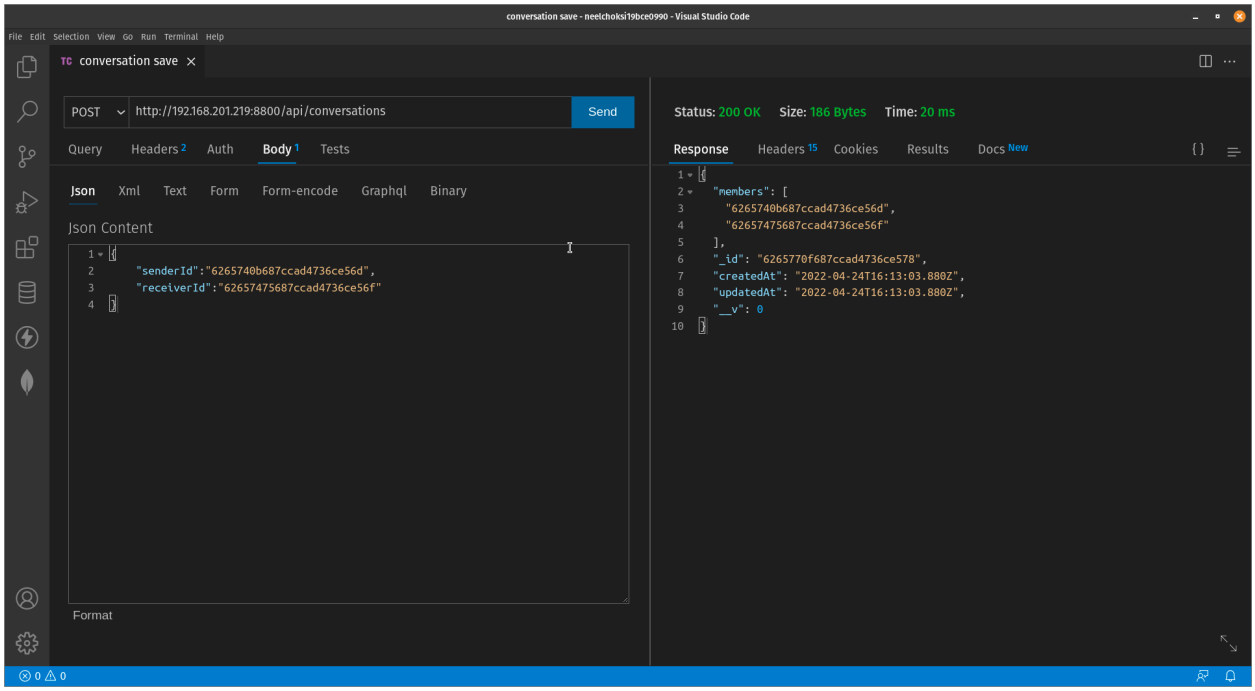


Figure 15. New Conversation from client1 to client2 to the server and response received.

conversation id : 6265770f687ccad4736ce578

Send message from client1 to client 2: http://192.168.201.219:8800/api/messages
Conversion created through the previous request is used to send a message from client1 to client2 using conversation id and id of client1.

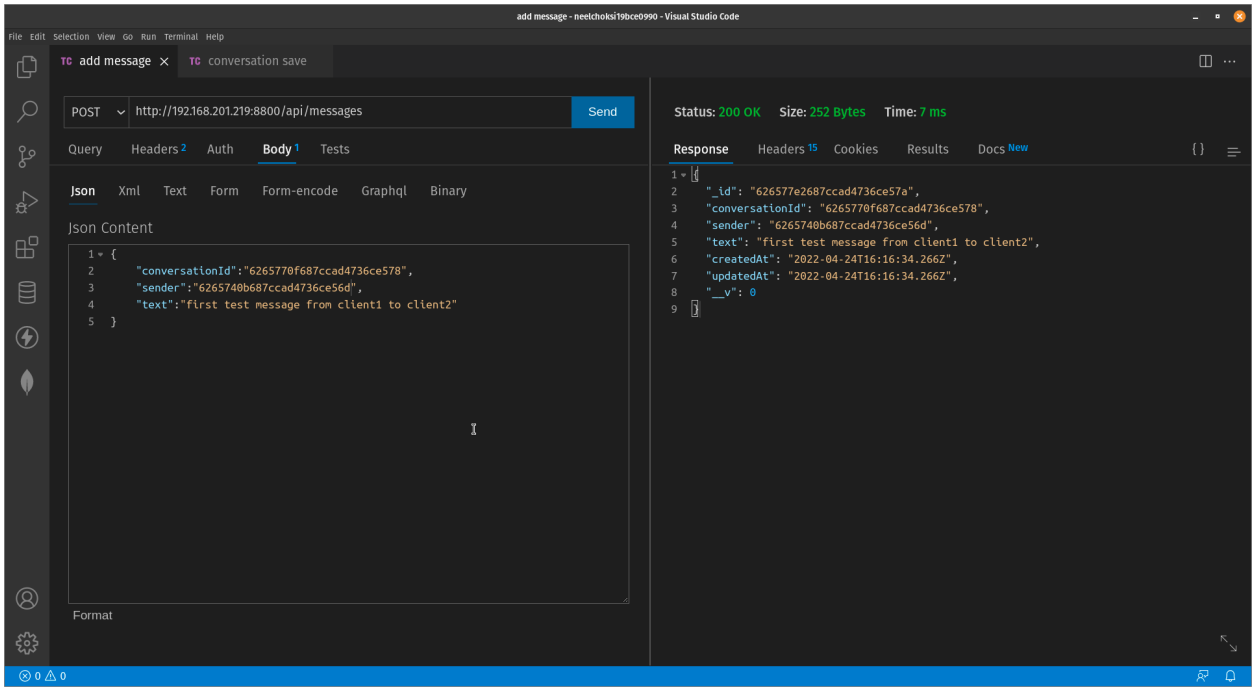


Figure 16. Send message request from client1 to client2 to the server and response received.

Getting the messages according to conversation id :

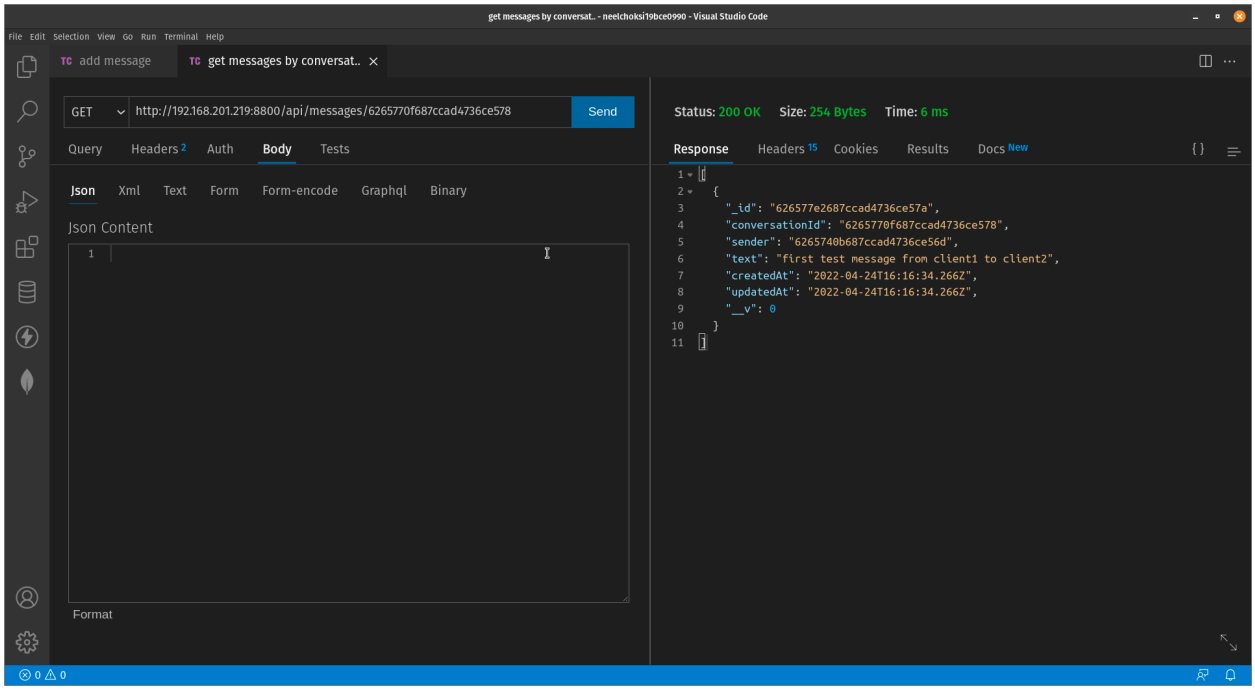


Figure 17. Getting messages request for conversation of client1 and client2 to the server and response received.

Nodejs server logs : Morgan is a logging middleware added to the nodejs server to log every request sent to the server.

```
neelchoksi19bce0990@neel:~/Documents/NeelVITCurrSem/sem6/proj/cse3502/ISM_Project_Sem6_VIT/chat-web-app-try1/server$ yarn start
yarn run v1.22.17
$ nodemon index.js
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Backend server is running
(node:54387) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Connected to mongodb
undefined
::ffff:127.0.0.1 - - [24/Apr/2022:15:59:46 +0000] "DELETE /api/users/62657348f7d34ad2371545df HTTP/1.1" 200 17
::ffff:192.168.201.219 - - [24/Apr/2022:16:00:11 +0000] "POST /api/auth/register HTTP/1.1" 200 433
::ffff:192.168.201.219 - - [24/Apr/2022:16:01:57 +0000] "POST /api/auth/register HTTP/1.1" 200 433
::ffff:127.0.0.1 - - [24/Apr/2022:16:05:17 +0000] "POST /api/auth/login HTTP/1.1" 200 433
::ffff:192.168.201.219 - - [24/Apr/2022:16:06:12 +0000] "POST /api/auth/login HTTP/1.1" 200 433
::ffff:192.168.201.219 - - [24/Apr/2022:16:07:07 +0000] "POST /api/auth/login HTTP/1.1" 200 433
::ffff:192.168.201.219 - - [24/Apr/2022:16:09:42 +0000] "PUT /api/users/6265740b687ccad4736ce56d/follow HTTP/1.1" 200 24
::ffff:192.168.201.219 - - [24/Apr/2022:16:13:03 +0000] "POST /api/conversations HTTP/1.1" 200 186
::ffff:192.168.201.219 - - [24/Apr/2022:16:16:34 +0000] "POST /api/messages HTTP/1.1" 200 252
::ffff:192.168.201.219 - - [24/Apr/2022:16:19:09 +0000] "GET /api/messages/6265770f687ccad4736ce578 HTTP/1.1" 200 254
```

Figure 18. Logs of API requests to the NodeJS server for creating messages, conversation , users, login users, creating posts, like and follow other users.

Client pages implemented using React

Register

Client 3 is registered using the email address, name , password and confirm password input field .

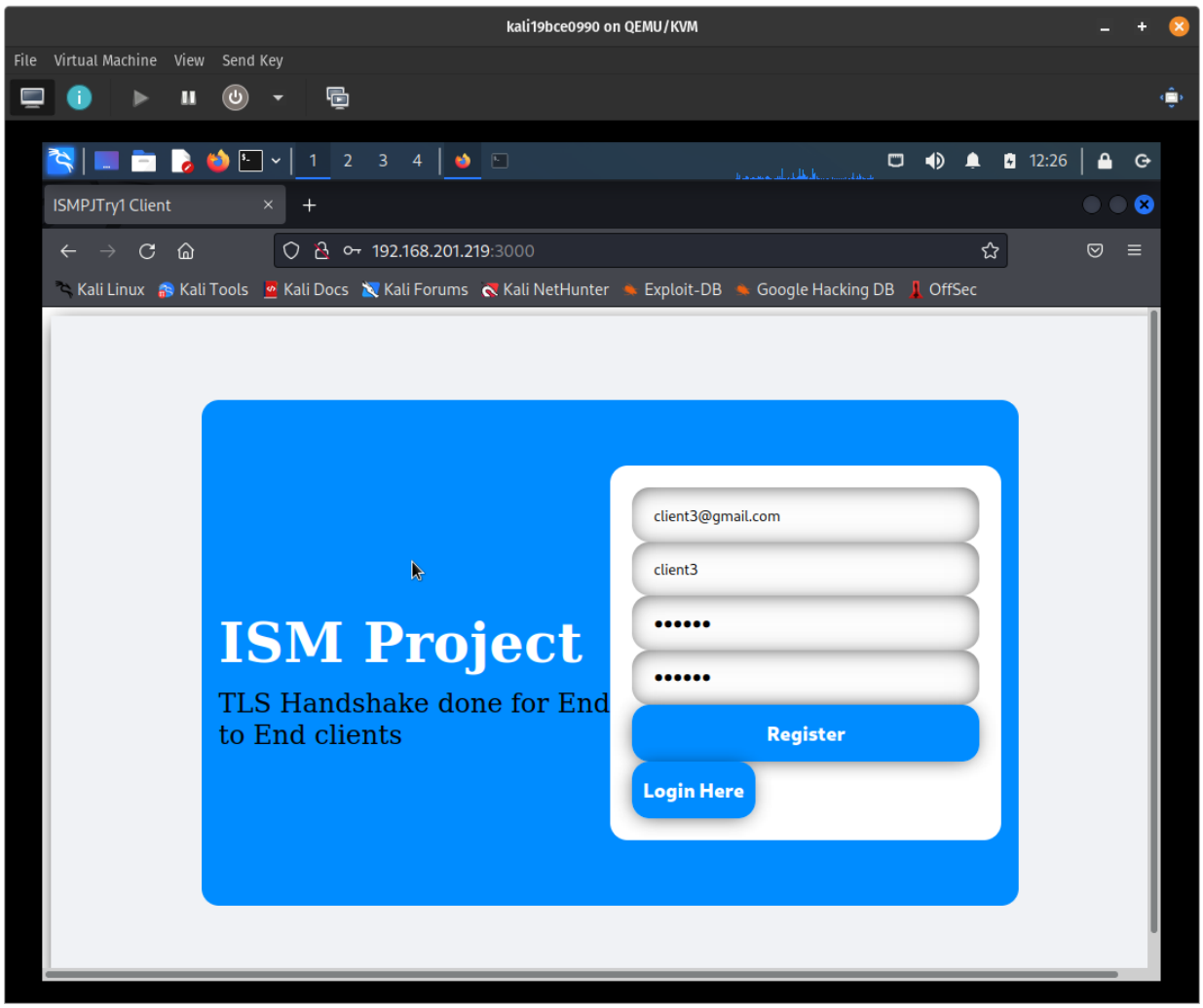


Figure 19. Register client3 from the frontend.

Login

Client3 is logged in using email address and password.

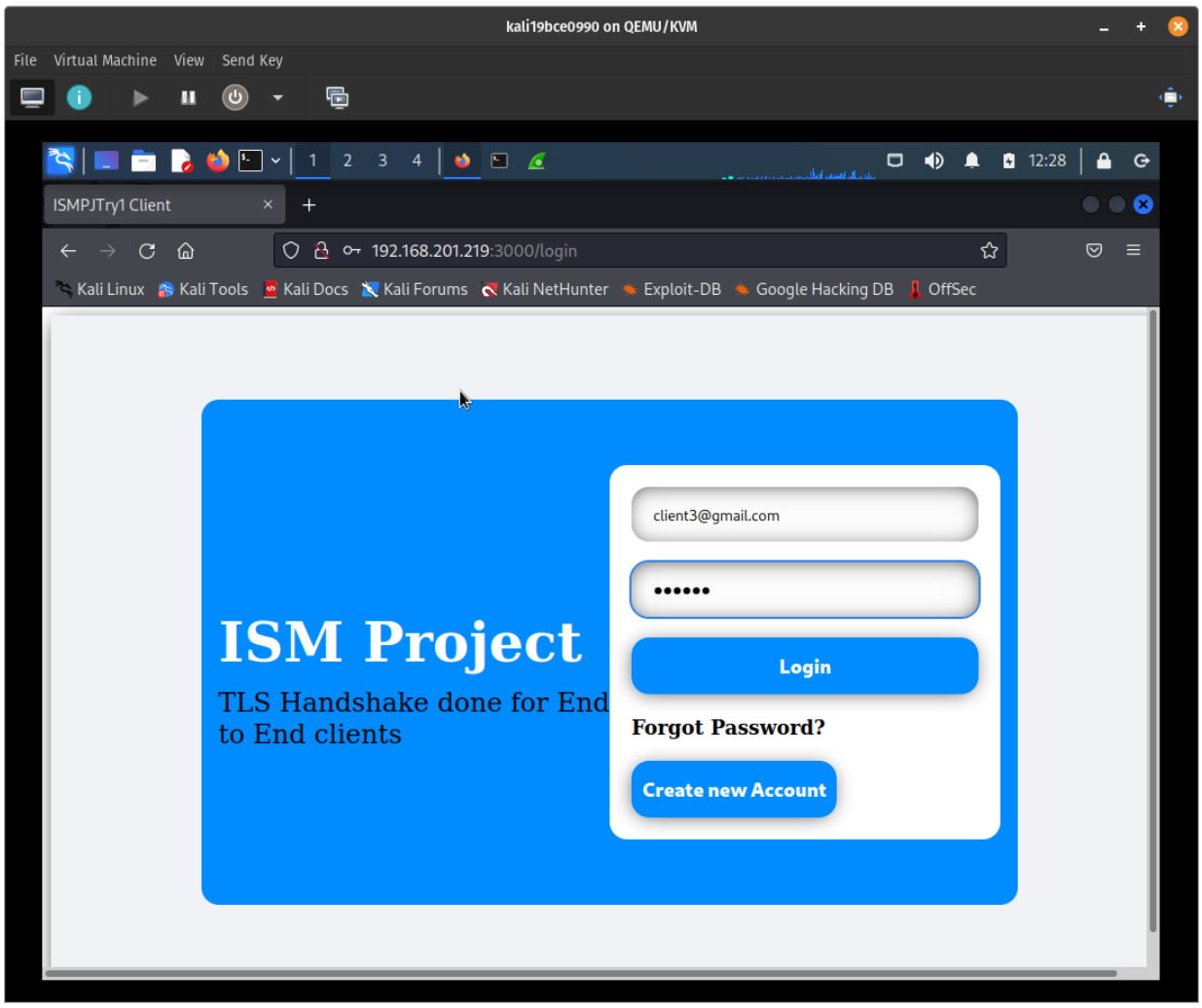


Figure 20. Login client3 from the frontend.

Posts

New post of client3 is created by adding description of post :"first post of client3"

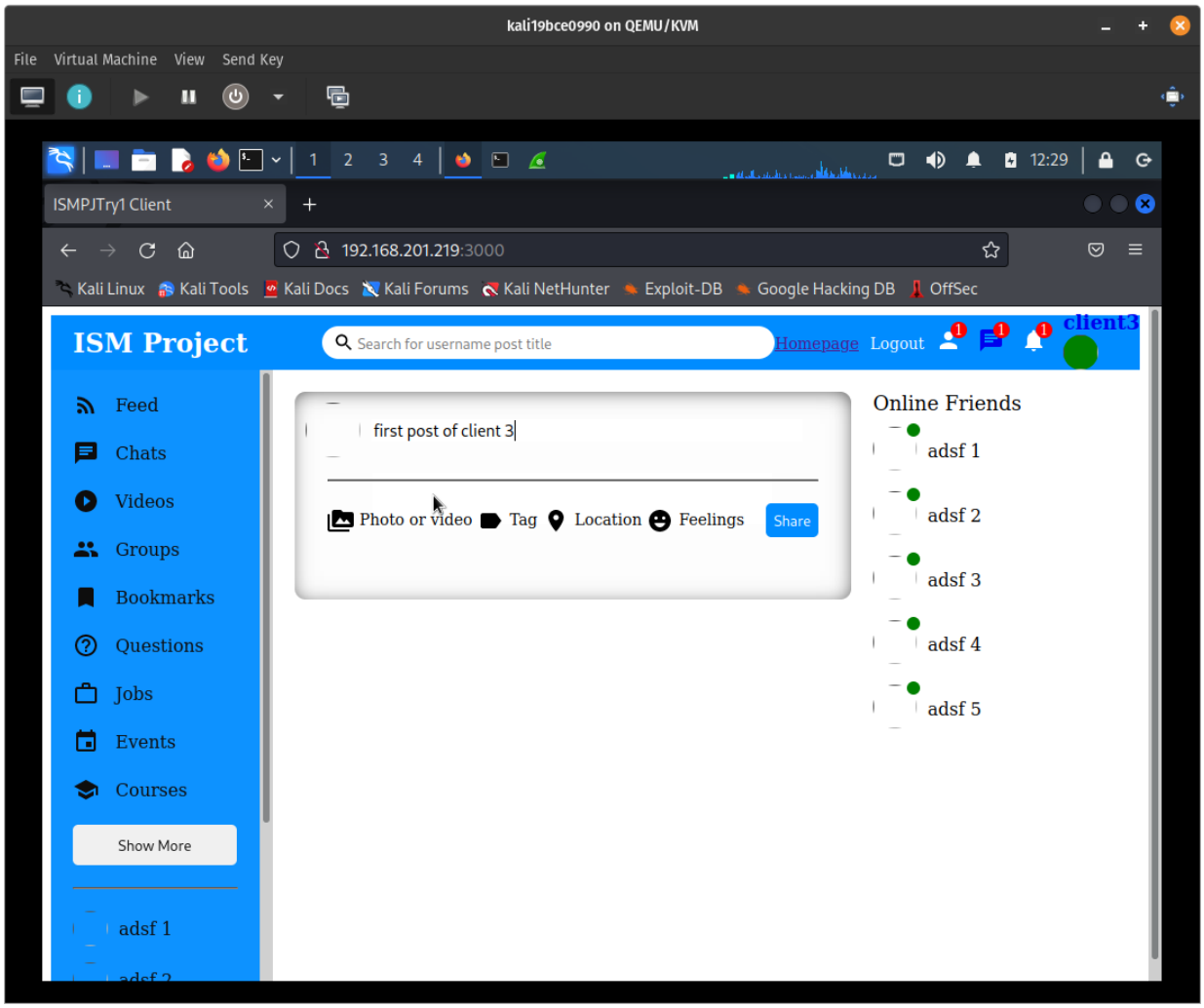


Figure 21. Create Post for client3 from the frontend.

client3 post is visible in the dashboard

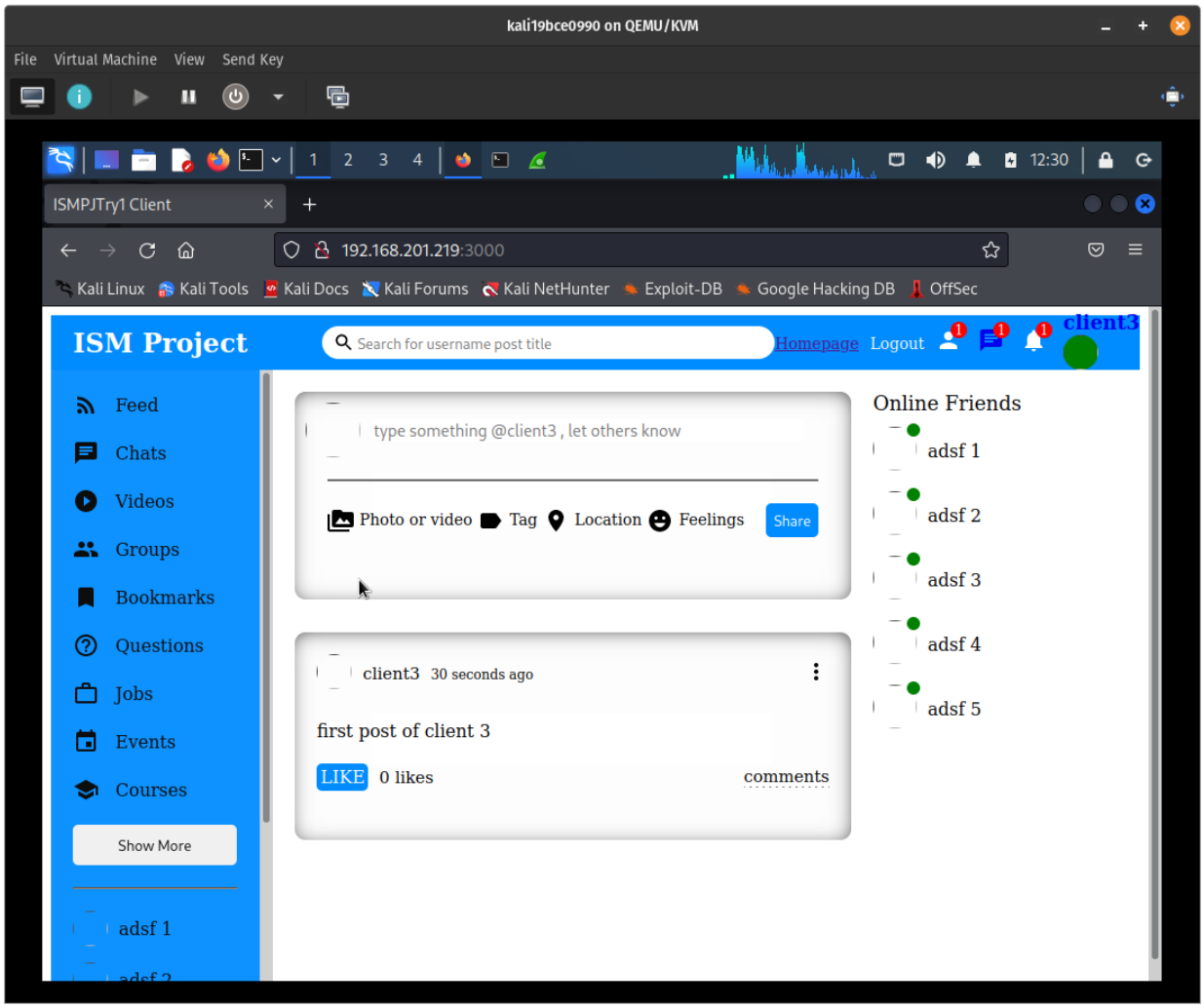


Figure 22. View posts for client3 from the frontend.

Profile

Client3 profile can be viewed by clicking on client3 image on top right of the screen or by visiting the url : http://192.168.201.219/client3 or http://192.168.201.219/<client3_id> . The username, email , description , all posts of client3 and users following client3 can be viewed.

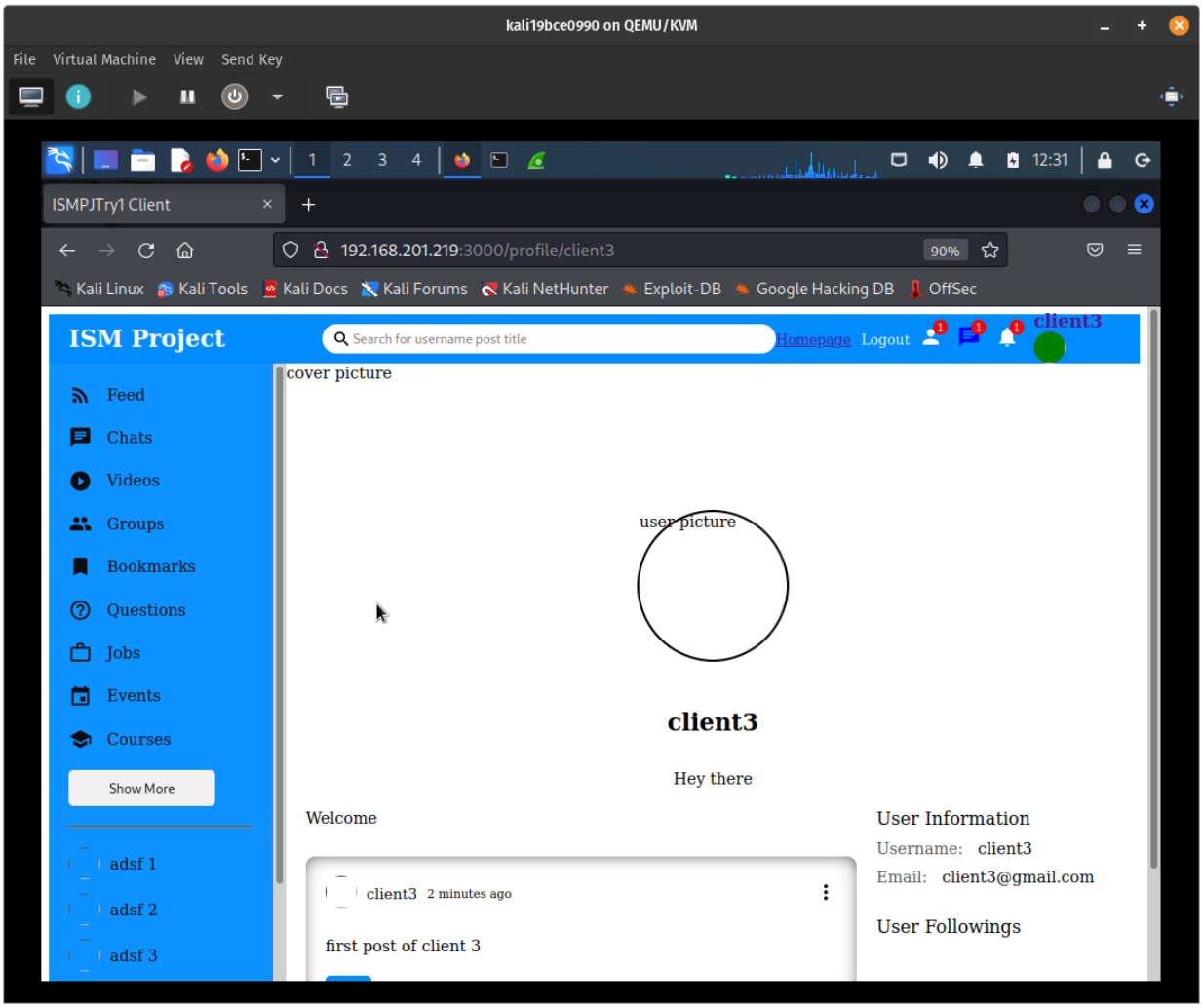


Figure 23. View profile of client3 from the frontend.

Real time Messenger

Client1 and Client2 messenger are opened by logging in as client1 from lubuntu2 vm (192.168.122.214) and as client2 from lubuntu20.04 vm (192.168.122.130).

Client1 messages dashboard is shown with client2 as a friend seen in the left bar and three test messages are seen . The socket server connected is the legitimate server.Messages sent from client1 to client2 are not modified by the hacker. The test message sent from client1 to client2 is “hello client2 “ and the message sent from client2 to client1 is “hello client1”.

Client1 : messenger dashboard

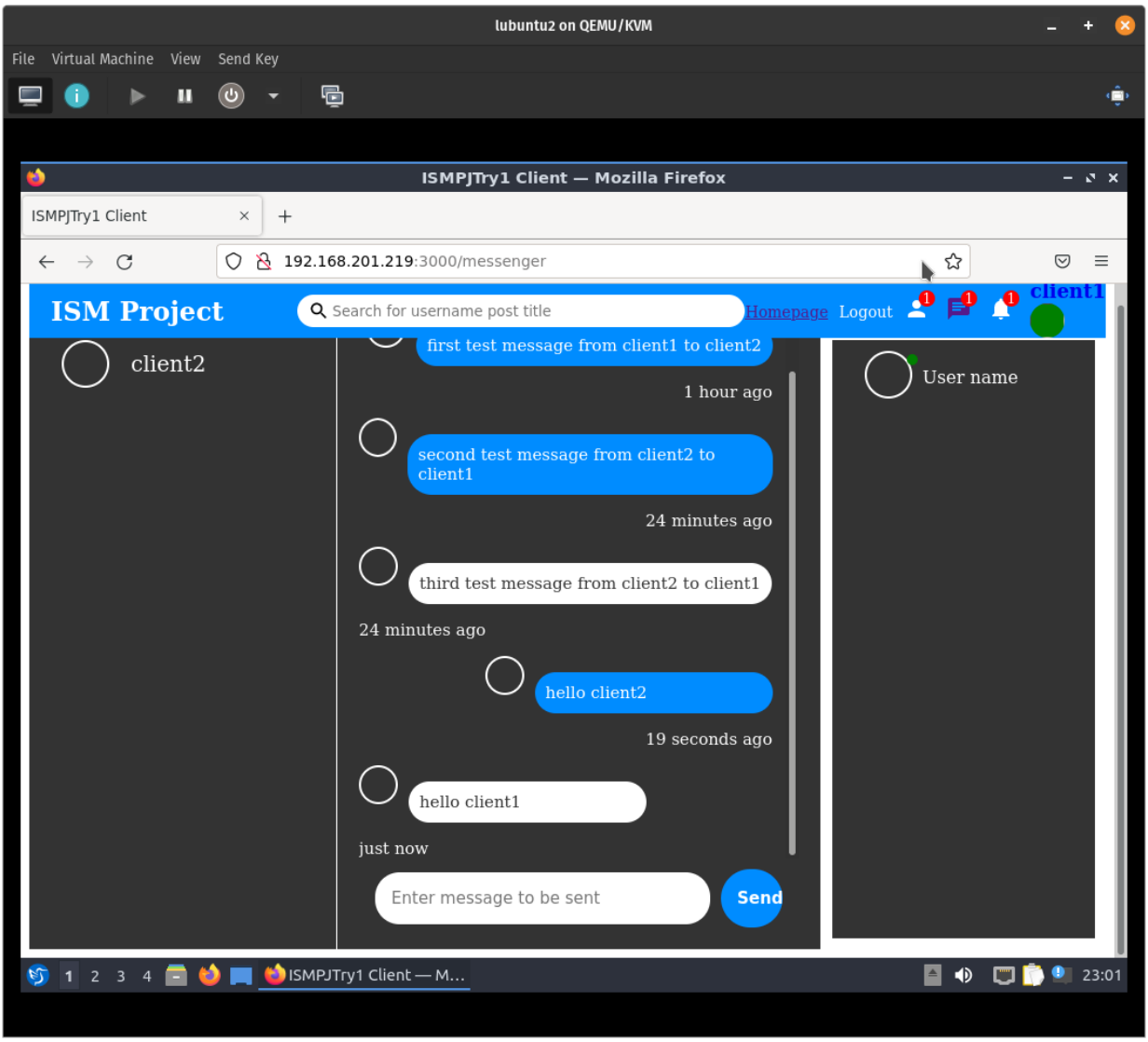


Figure 24. Sending hello message to client2 from client1 from the frontend.

Client2 : messenger dashboard

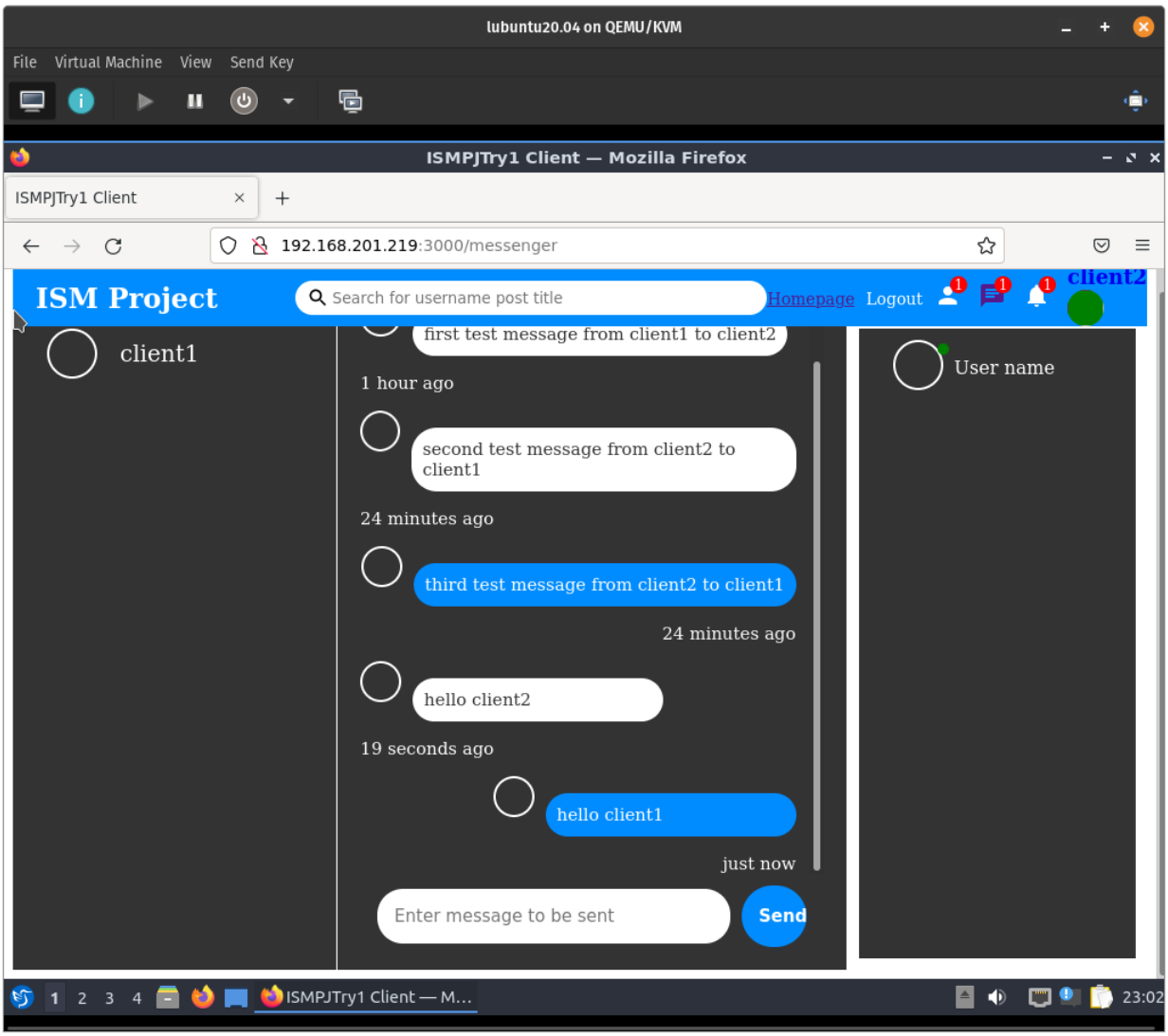


Figure 25. Sending hello message to client1 from client2 from the frontend.

legitimate socket server logs :

```
neelchoksi19bce0990@neel:~/Documents/NeelVITCurrSem/sem6/proj/cse3502/ISM_Project_Sem6_VIT/chat-web-app-try1/socket-legitimate$ yarn start
yarn run v1.22.17
$ nodemon index.js
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
user:6265740b687ccad4736ce56d connected
user:62657475687ccad4736ce56f connected
```

Figure 25. Logs generated by legitimate sockets server.

MITM client , host implementation

Host with the ip address of 192.168.201.219 is running the client server, node server , socket server.

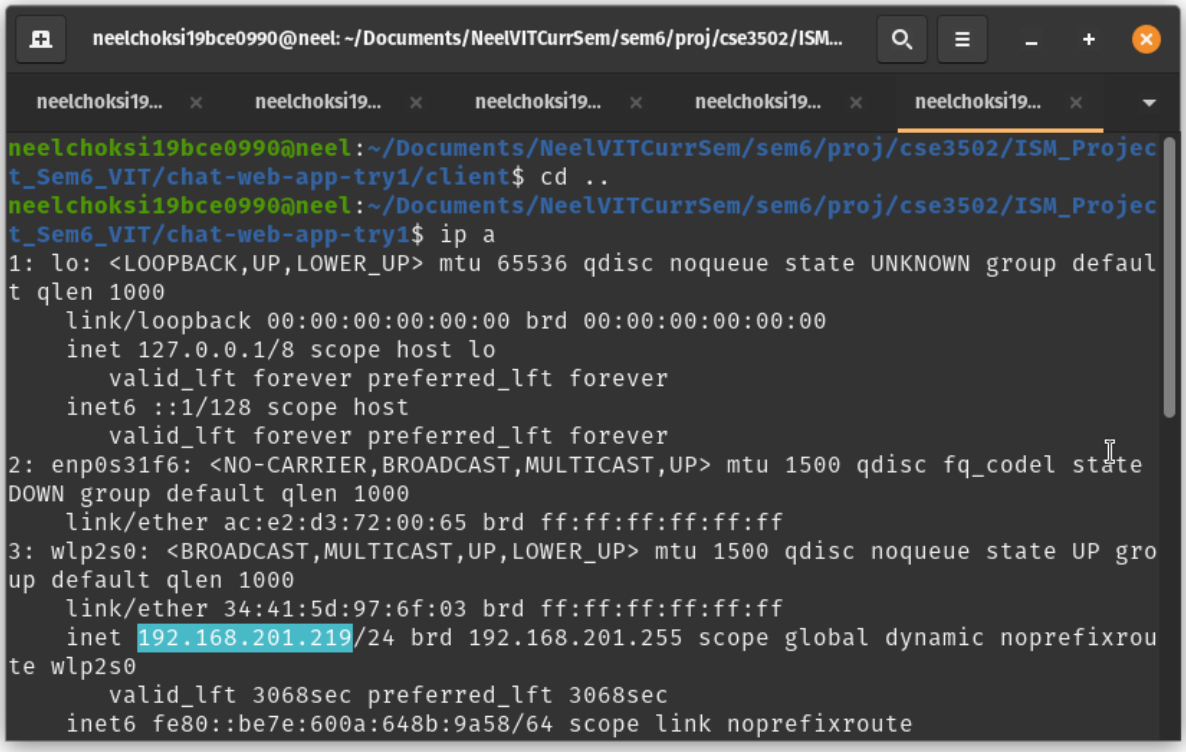


Figure 26. IP address of the host machine hosting React Frontend, Legitimate and mitm sockets server and NodeJS server.

Client 1 is running on lubuntu2 virtual machine with ip address of 192.168.122.214.

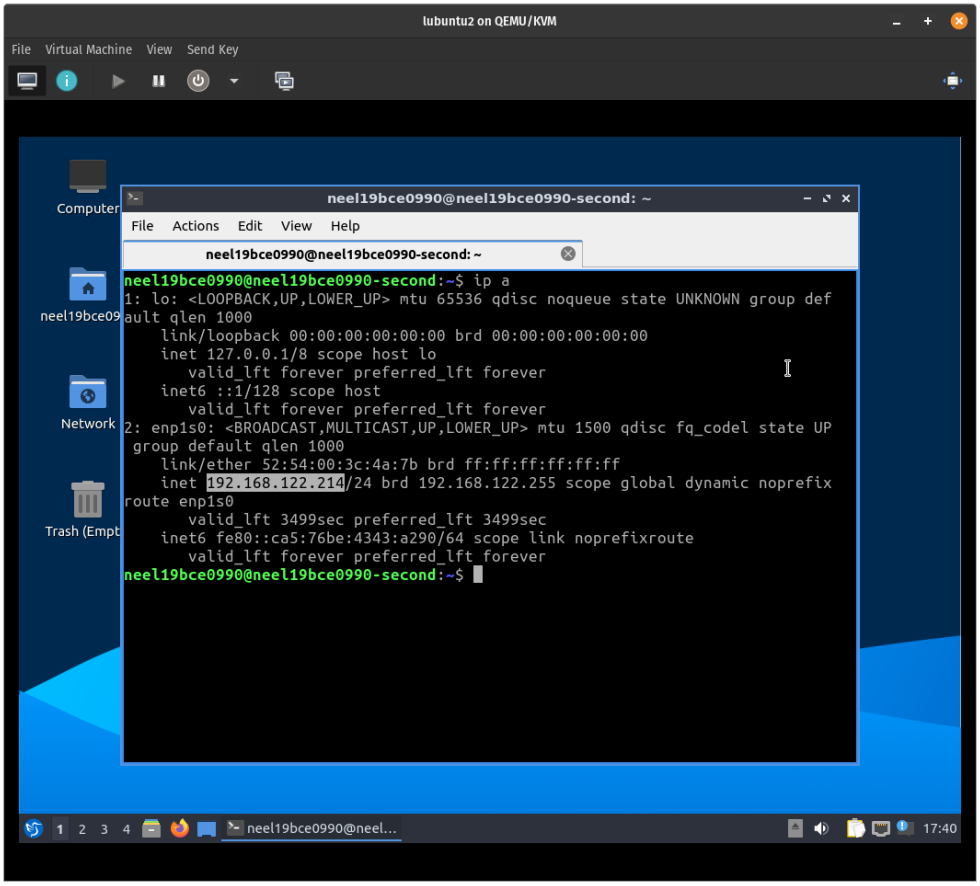


Figure 27. IP address of lubuntu2 vm .

Client 2 is running on lubuntu20.04 virtual machine with ip address of 192.168.122.130.

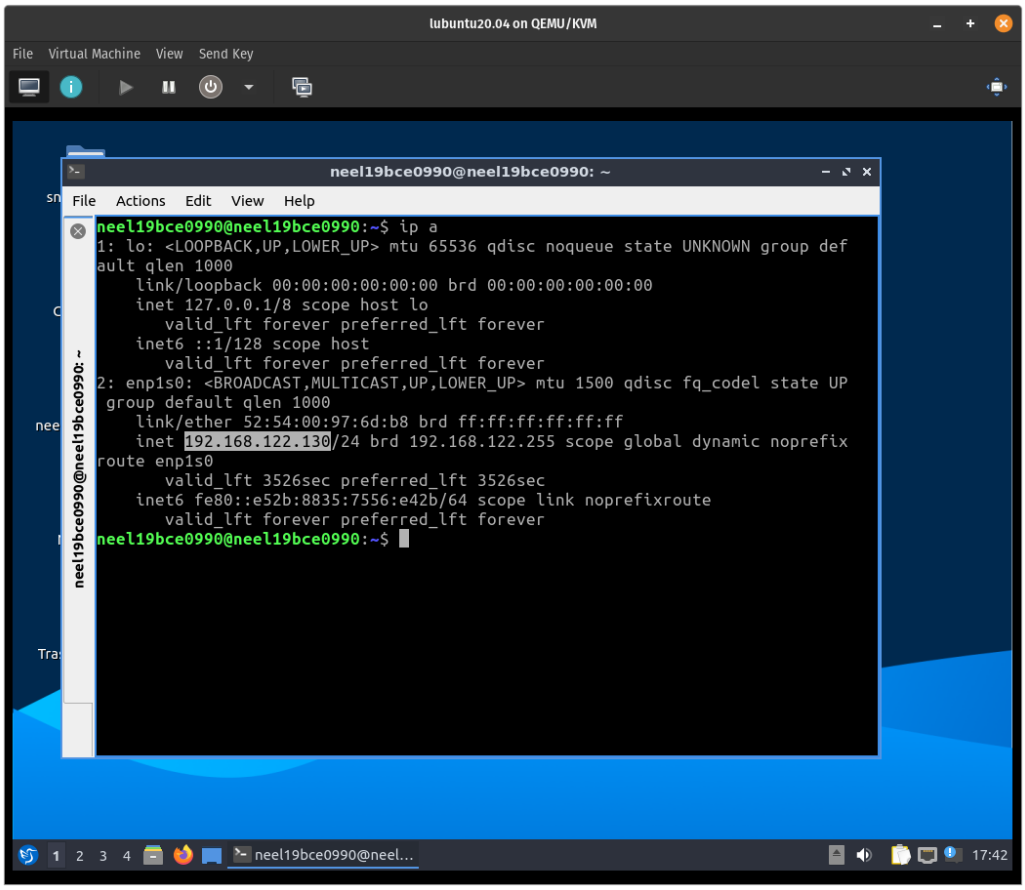


Figure 28. IP address of lubuntu20.04 vm.

Network map using nmap on kali linux : (192.168.122.73) . Network is depicted in Figure 5.

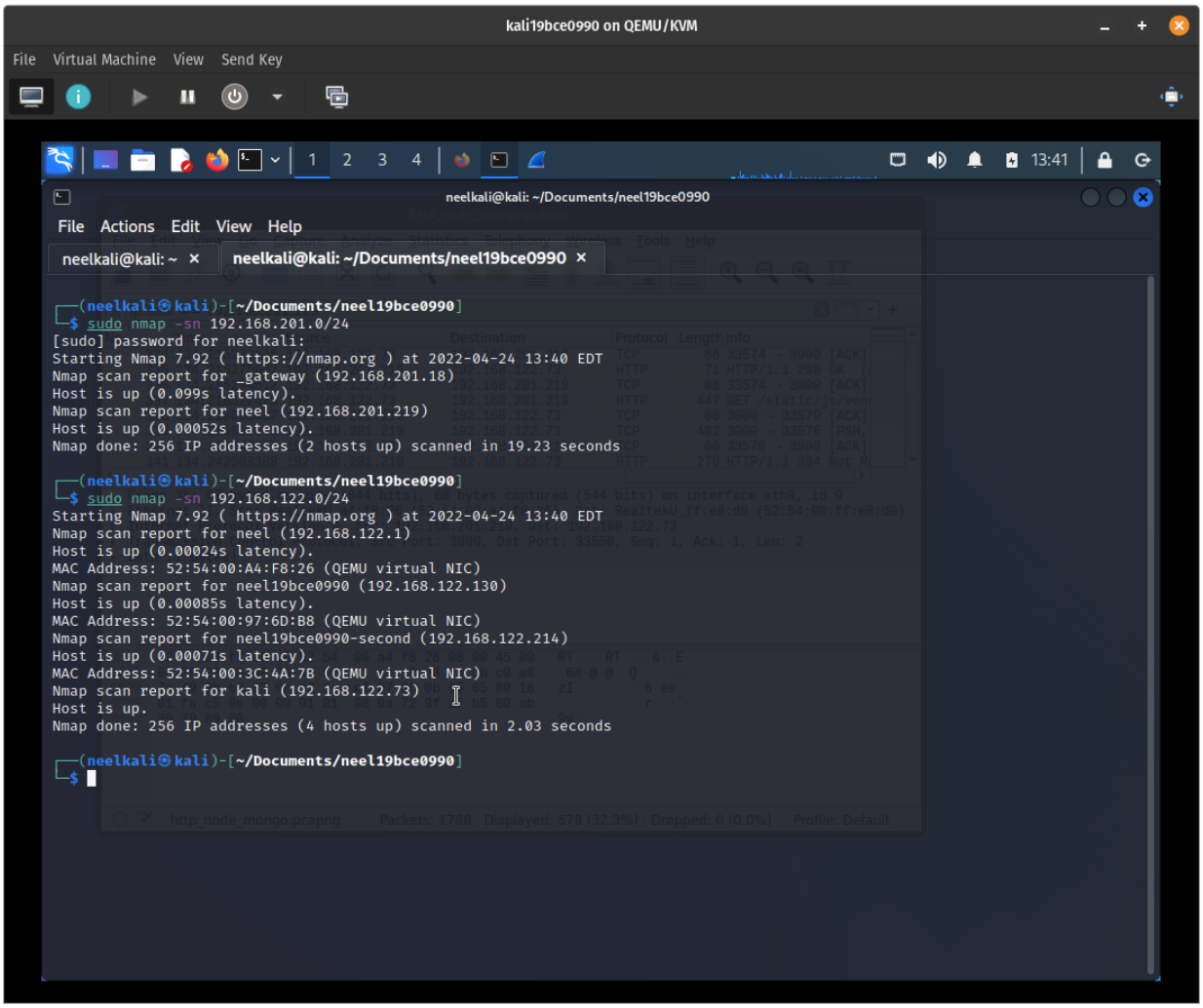


Figure 29. Running NMAP on kali linux to get devices connected in virtual machine network as well as host network .

intercepting the real time message packets sent from lubuntu2 vm-client1(192.168.122.214) to the virtual bridge interface(192.168.201.1) using kali linux virtual machine with ip address 192.168.122.73.

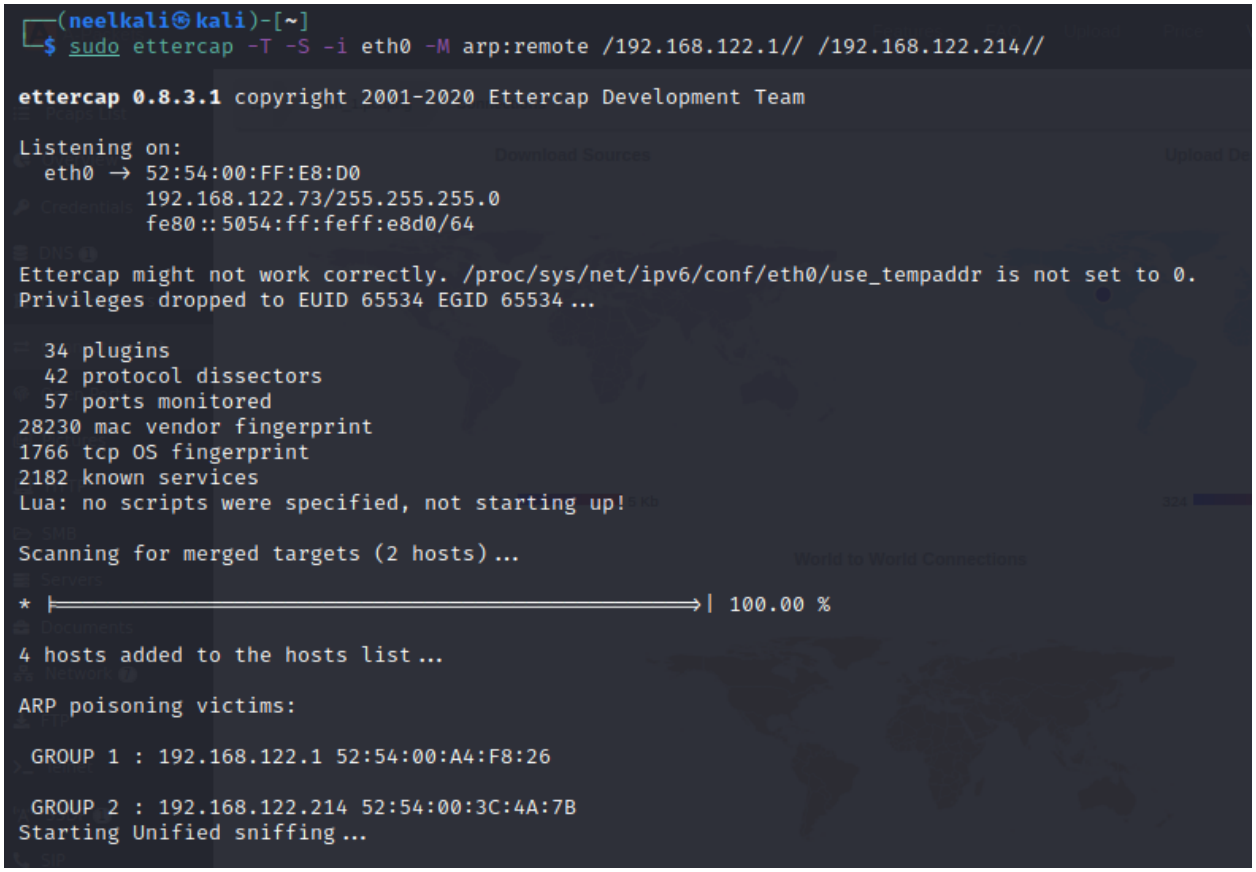


Figure 30. Starting Man in the middle attack between gateway of virtual machine network and lubuntu2 vm.

intercepting the real time message packets sent from lubuntu20.04 vm-client2(192.168.122.130) to the virtual bridge interface(192.168.201.1) using kali linux virtual machine with ip address 192.168.122.73.

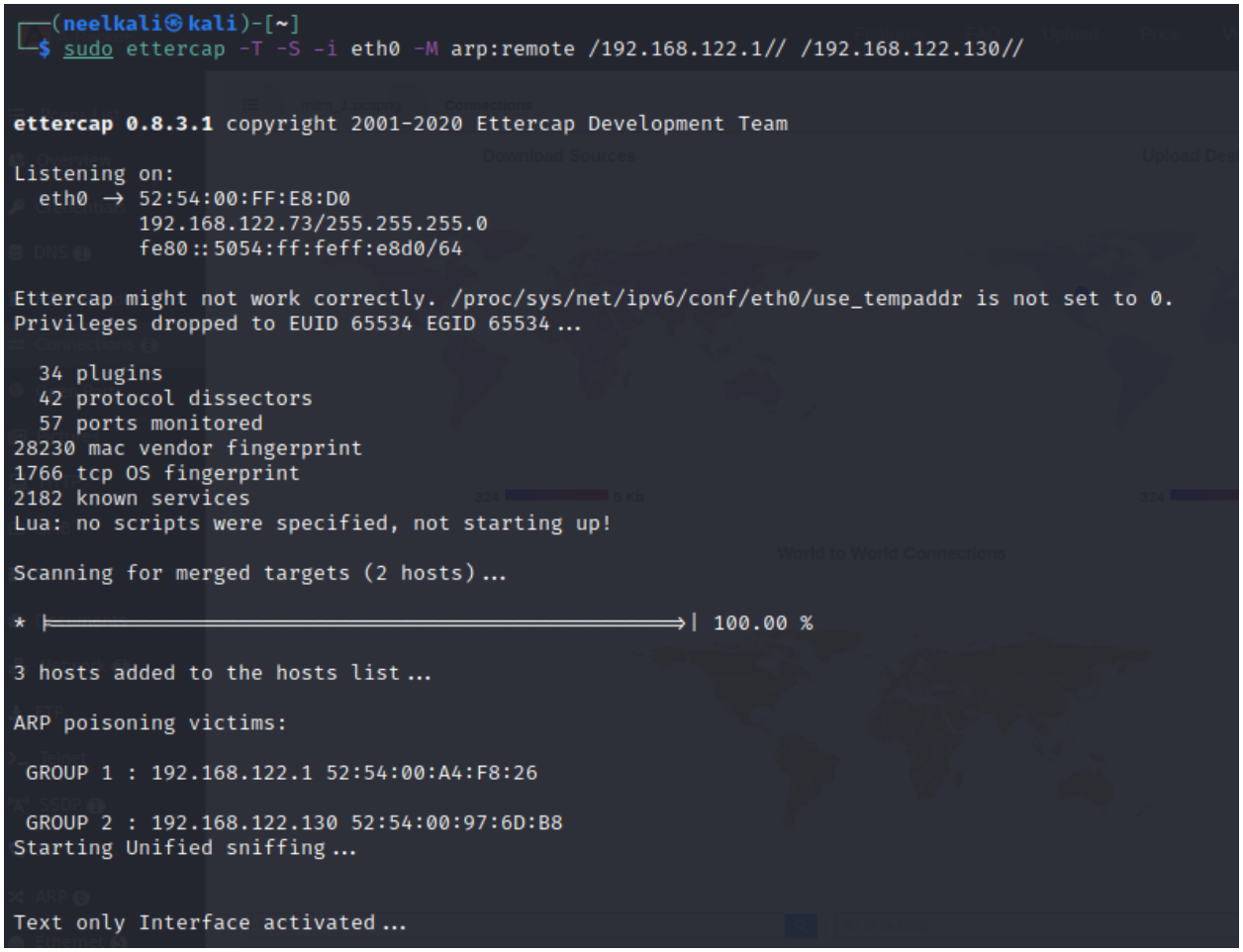


Figure 31. Starting Man in the middle attack between gateway of virtual machine network and lubuntu20.04 vm.

MITM at the application layer

Client1 and Client2 messenger are opened by logging in as client1 from lubuntu2 vm (192.168.122.214) and as client2 from lubuntu20.04 vm (192.168.122.130).

Client1 messages dashboard is shown with client2 as a friend seen in the left bar and three test messages are seen . The socket server connected is the malicious server and a scenario has been emulated wherein the hacker has managed to replace the legitimate socket server with the malicious sockets server. The message is being modified when sent from client1 to client2 and vice versa. Here the hacker has appended the message : “hello from hacker “.The updated message is seen in the chat windows of both client1 and client2.

Client1 : messages dashboard

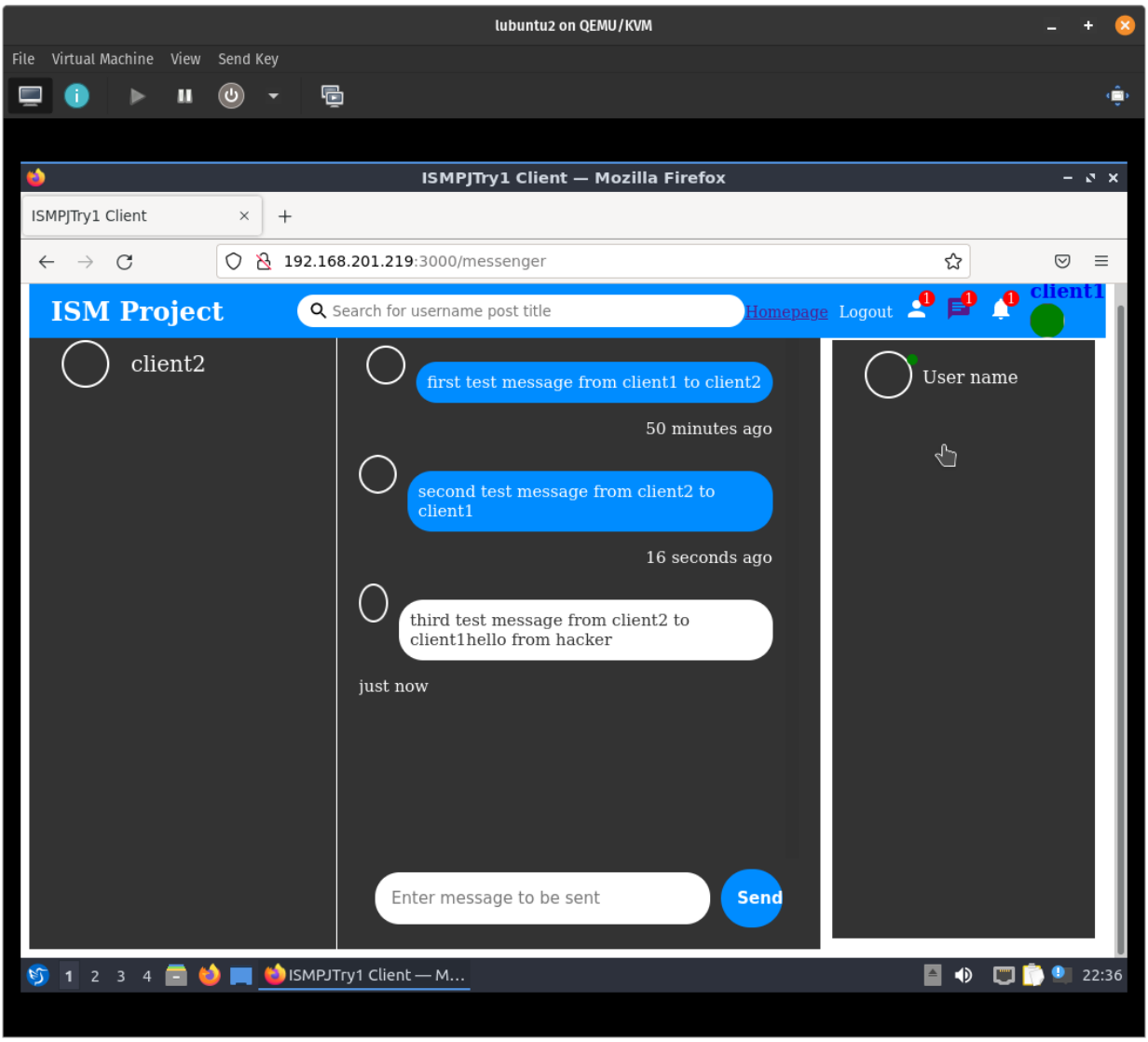


Figure 32. Application Layer Man in the middle attack displayed by sending message from client2 to client1 and “hello from hacker” is appended.

Client2 : messages dashboard

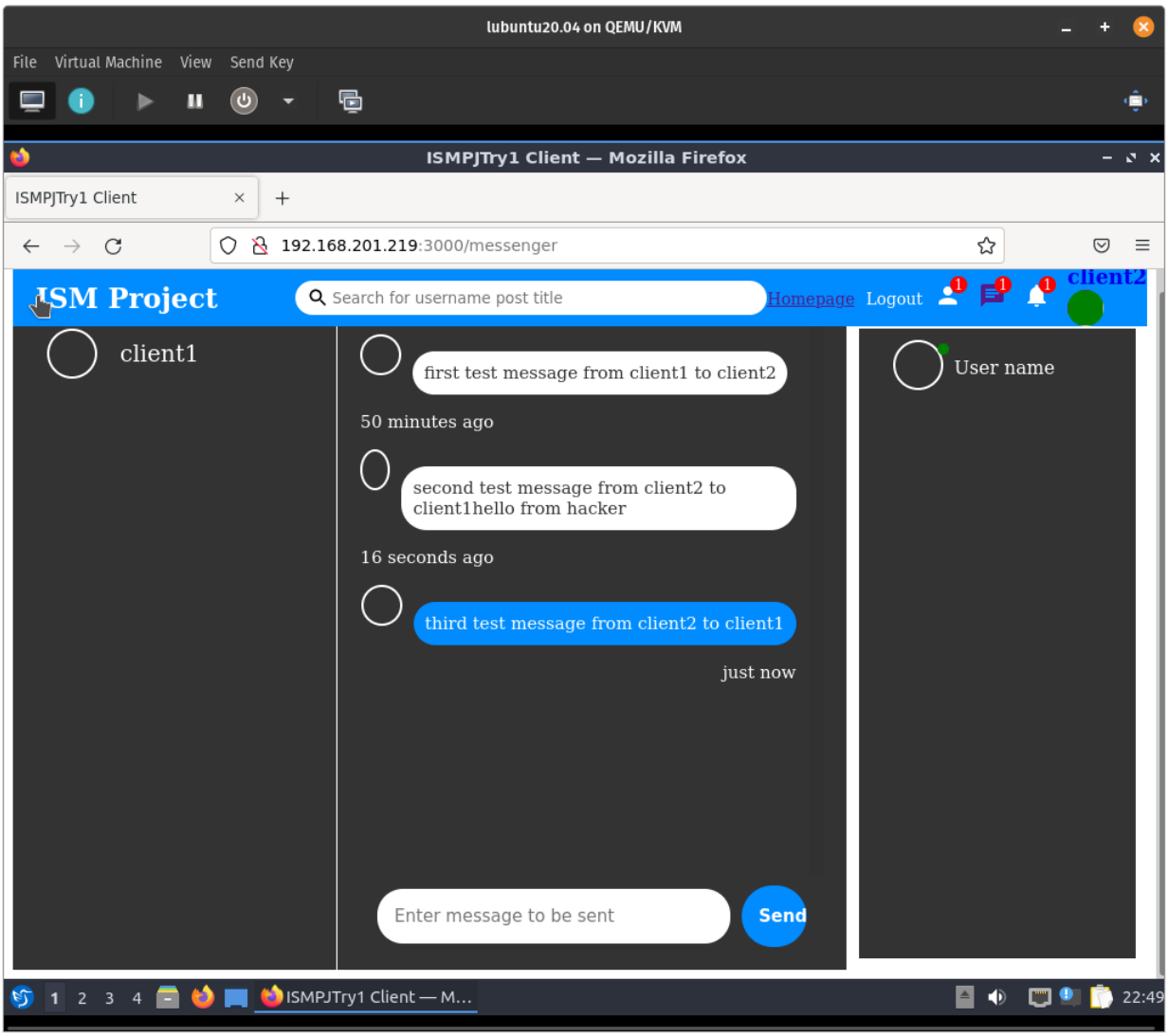


Figure 33. Application Layer Man in the middle attack displayed by sending message from client1 to client2 and “hello from hacker” is appended.

Here the success rate of client to host man in the middle attack is very low since the client is hosted on the react server which is secure and stops sending packets if the arp poisoning is detected.

Analysis

Packet flow of http packets with mongodb and node js.

The wireshark packet analyzer is run on kali linux vm (192.168.122.73) . The client web page is accessed (hosted at : 192.168.201.219) by the firefox browser in the kali linux vm . A new user with username client2 is registered and logged in using the client graphical user interface. A new post is created for the user and the post is viewed for client3. Profile page of

client3 is visited and logged out of the system using the graphical user interface. The packet file gave the following analysis.

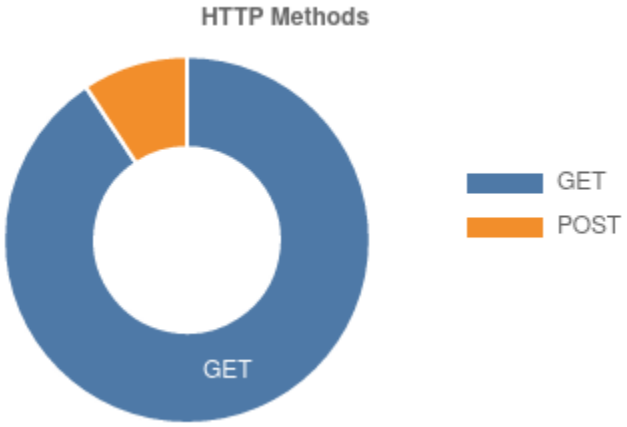


Figure 34. Client3 Activity Packet file methods used to send requests.

Most of the requests sent were get requests

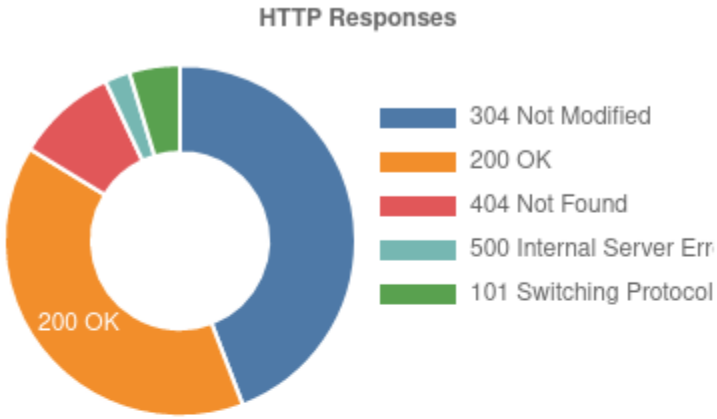


Figure 35. Client3 Activity Packet file success rate of packets while sending requests.

There were some internal server errors during the interaction of the client and 45% of the responses were successful indicated by 200 OK .

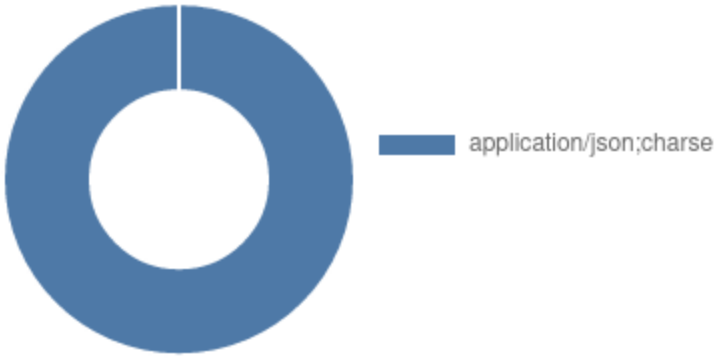


Figure 36.Client3 Activity Packet file type of data used to send requests.

The form of data transferred between the client and the server included application/json .

Client 3 registration HTTP follow up

192.168.122.73:33552 ↔ 192.168.201.219:3000 (POST)

```
POST /auth/register HTTP/1.1
Host: 3000
Accept:  application/json, text/plain, */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Content-Length:  70
Content-Type:  application/json;charset=utf-8
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

{"username":"client3","email":"client3@gmail.com","password":"123456"}

HTTP/1.1 200 OK
Connection: close
Content-Length: 433
Content-Security-Policy: 'unsafe-inline';upgrade-insecure-requests
Content-Type: application/json; charset=utf-8
Date:33 GMT
Etag: W/"1b1-vmffkrCrs6T1iJ/L6fLXjAjb0yU"
Expect-Ct: max-age=0
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Powered-By: Express
X-Xss-Protection: 0

{"cryptography":{"globalParameters":{"curveType":"modp15"},"publicKey":"","privateKey":"","sharedKey":""},"profilePicture":"","coverPicture":"","followers":[],"followings":
[],"isAdmin":false,"_id":"62657a75687ccad4736ce57d","username":"client3","email":"client3@gmail.com","password":"$2b$10$XztiiZA1clzpP8s1lWCw3u8XshfmO6RxjgWpW9nKoYZm4F.W45CeG",
"createdAt":"2022-04-24T16:27:33.443Z","updatedAt":"2022-04-24T16:27:33.443Z","__v":0}
```

Figure 37.Client3 Registration TCP stream.

Clietn3 login HTTP follow up

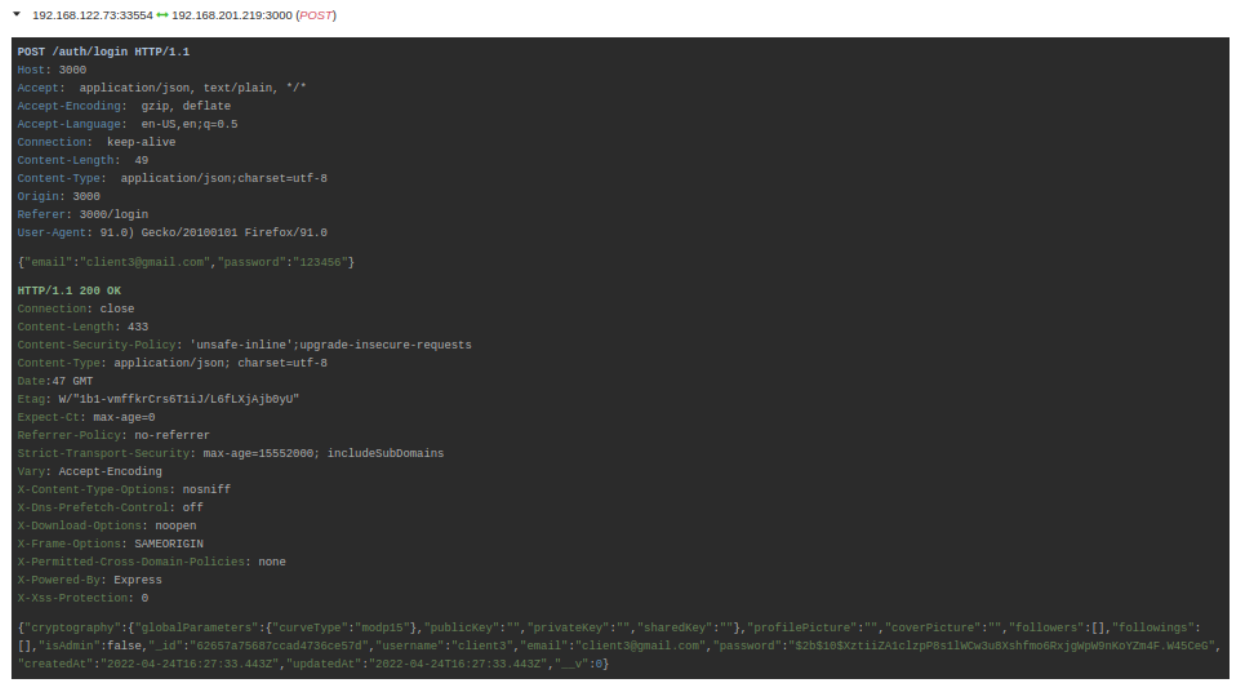


Figure 38.Client3 user login TCP stream.

Getting user information of client3:

Figure 39.Client3 getting user information TCP stream.

Getting friends of client3, failed since client3 was newly registered

192.168.122.73:33596 ↔ 192.168.201.219:3000 (GET)

```
GET /users/friends/undefined HTTP/1.1
Host: 3000
Accept:  application/json, text/plain, */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Referer: 3000/profile/client3
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 500 Internal Server Error
Connection: close
Content-Length: 243
Content-Security-Policy: 'unsafe-inline';upgrade-insecure-requests
Content-Type: application/json; charset=utf-8
Date:40 GMT
Etag: W/"f3-QwPF952Voc9t51n5DouR7Qm0M00"
Expect-Ct: max-age=0
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Powered-By: Express
X-Xss-Protection: 0

{"stringValue":"\"undefined\"","valueType":"string","kind":"ObjectId","value":"undefined","path":"_id","reason":{},"name":"CastError","message":"Cast
to ObjectId failed for value \"undefined\" (type string) at path \"_id\" for model \"User\""}
```

Figure 40.Client3 get friends TCP stream.

Posting a post of client3 :

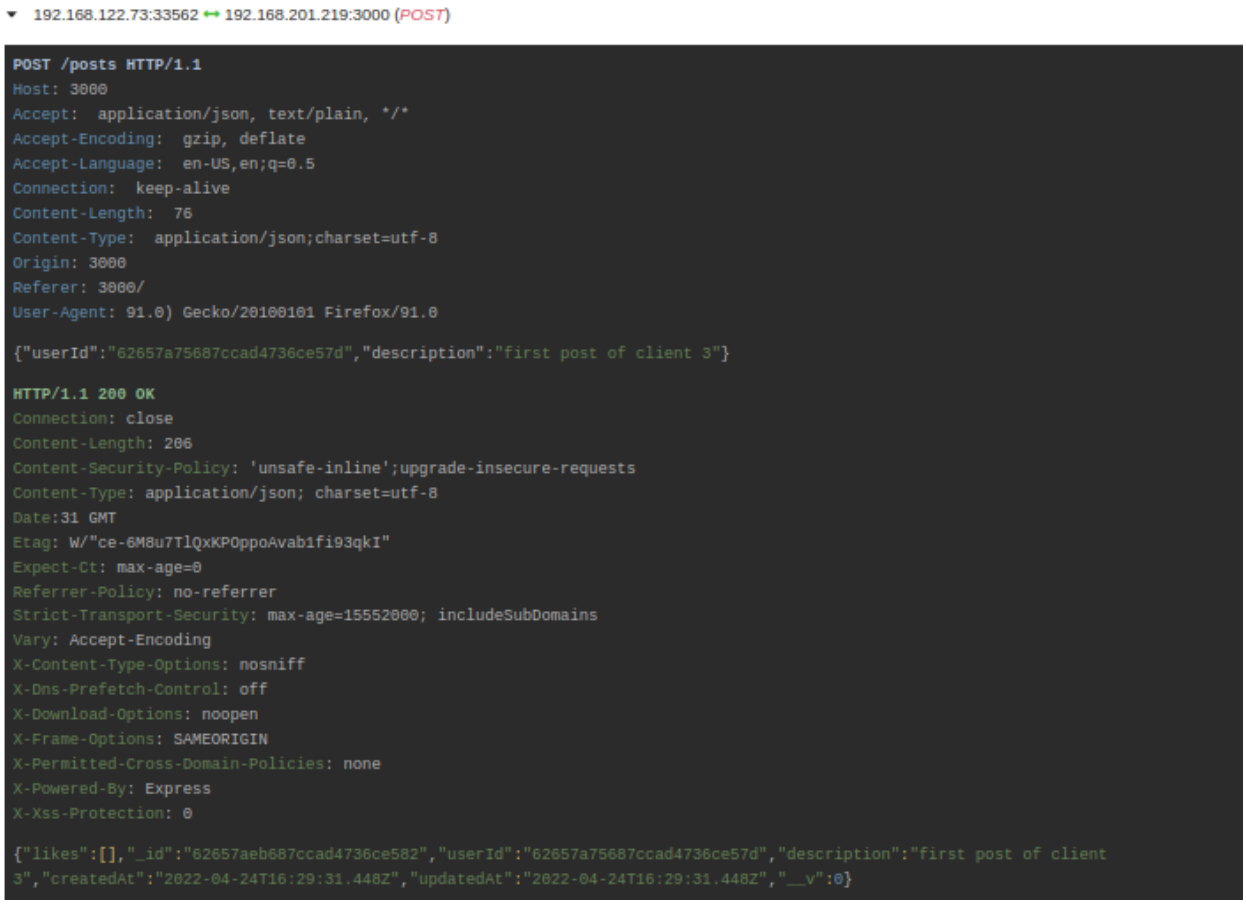


Figure 41.Client3 posting a post TCP stream.

Getting all posts of client3 :

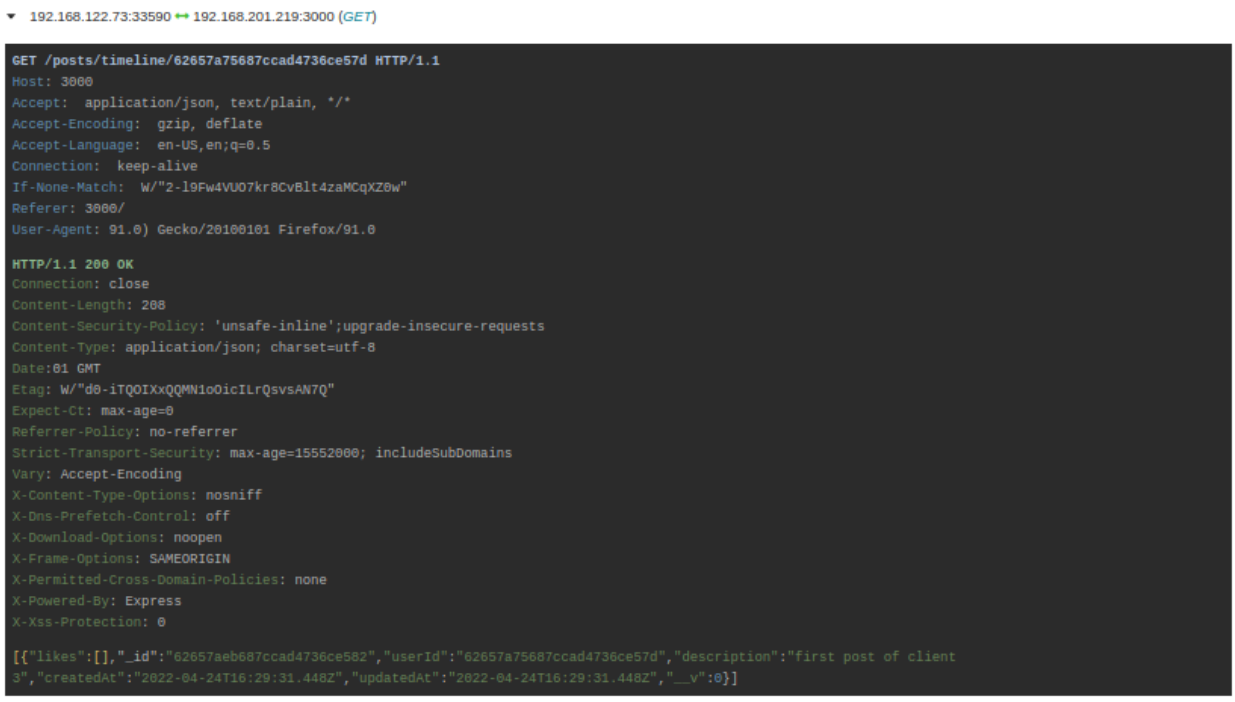


Figure 42.Client3 get all posts TCP stream.

Message post and get by client 1 :

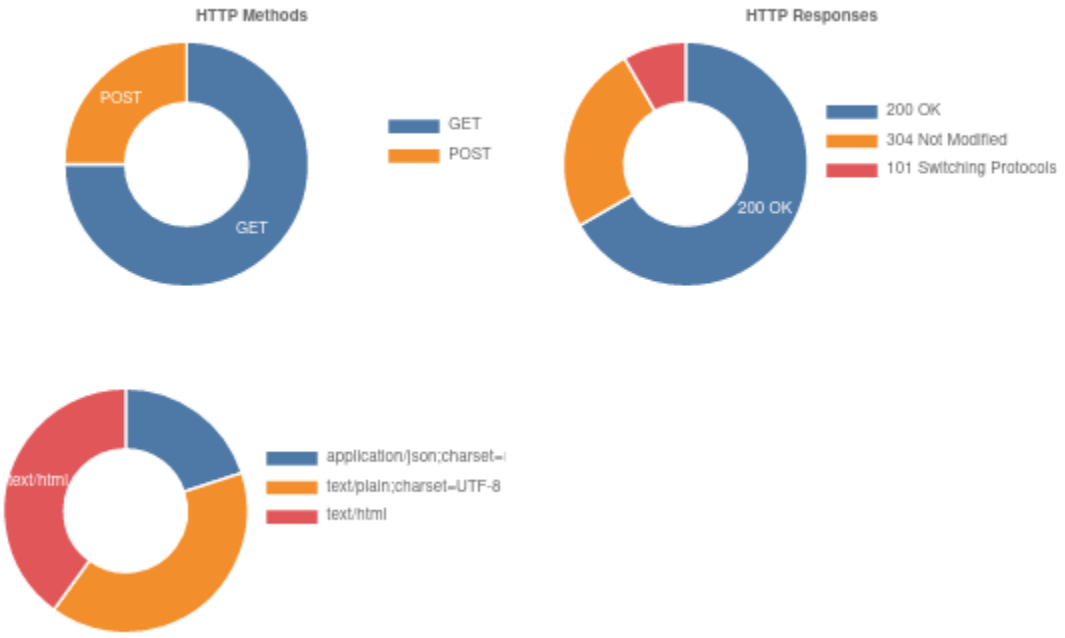


Figure 42.Client1 post message types of methods, responses, data.

▼ 192.168.122.73:33646 ↔ 192.168.201.219:3000 (*POST*)

```
POST /messages/ HTTP/1.1
Host: 3000
Accept:  application/json, text/plain, */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Content-Length:  118
Content-Type:  application/json;charset=utf-8
Origin: 3000
Referer: 3000/messenger
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

{"sender":"6265740b687ccad4736ce56d","text":"nodejs message server test2","conversationId":"6265770f687ccad4736ce578"}

HTTP/1.1 200 OK
Connection: close
Content-Length: 237
Content-Security-Policy: 'unsafe-inline';upgrade-insecure-requests
Content-Type: application/json; charset=utf-8
Date:05 GMT
Etag: W/"ed-jMdtMdQ8/vwqQC6fXQUQk/5YOHE"
Expect-Ct: max-age=0
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Powered-By: Express
X-Xss-Protection: 0

{"_id":"62659ba1d96ab5f050de2aaa","sender":"6265740b687ccad4736ce56d","text":"nodejs message server
test2","conversationId":"6265770f687ccad4736ce578","createdAt":"2022-04-24T18:49:05.953Z","updatedAt":"2022-04-24T18:49:05.953Z","__v":0}
```

▼ 192.168.122.73:33638 ↔ 192.168.201.219:3000 (*GET*)

```
GET /conversations/6265740b687ccad4736ce56d HTTP/1.1
Host: 3000
Accept:  application/json, text/plain, */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
If-None-Match:  W/"bc-Ob9I9dqw1F4cdT+LqOSa1lRmb1k"
Referer: 3000/messenger
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 304 Not Modified
Connection: close
Content-Security-Policy: 'unsafe-inline';upgrade-insecure-requests
Date:32 GMT
Etag: W/"bc-Ob9I9dqw1F4cdT+LqOSa1lRmb1k"
Expect-Ct: max-age=0
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Powered-By: Express
X-Xss-Protection: 0
```

192.168.122.73:36336 ↔ 192.168.201.219:8900 (GET)

```
GET /socket.io/?EIO=4&transport=websocket&sid=jZEGNdB3Ek90kqhtAAAG HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Cache-Control:  no-cache
Connection:  keep-alive, Upgrade
Origin: 3000
Pragma:  no-cache
Sec-Websocket-Extensions:  permessage-deflate
Sec-Websocket-Key:  H1ReKfiUb7n47ZEkyGS59Q==
Sec-Websocket-Version:  13
Upgrade:  websocket
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Sec-Websocket-Accept: xEC0Ur/3YWbpfgV7lzgWFz9sqDo=
Upgrade: websocket
```

192.168.122.73:33644 ↔ 192.168.201.219:3000 (GET)

```
GET /messages/6265770f687ccad4736ce578 HTTP/1.1
Host: 3000
Accept:  application/json, text/plain, */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
If-None-Match:  W/"ba5-cgWk1TsWA9wY+tVft1c6zgk55x4"
Referer: 3000/messenger
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 200 OK
Connection: close
Transfer-Encoding: chunked
Content-Encoding: gzip
Content-Security-Policy: 'unsafe-inline';upgrade-insecure-requests
Content-Type: application/json; charset=utf-8
Date:43 GMT
Etag: W/"d97-RVvD9HsdARNHT1m5JJ1CIzM7X8Y"
Expect-Ct: max-age=0
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Dns-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-Powered-By: Express
X-Xss-Protection: 0

[{"_id":"626577e2687ccad4736ce57a","conversationId":"6265770f687ccad4736ce578","sender":"6265740b687ccad4736ce56d","text":"first test message from
client1 to client2","createdAt":"2022-04-24T16:16:34.266Z","updatedAt":"2022-04-24T16:16:34.266Z","__v":0},
{"_id":"6265838a687ccad4736ce5a8","sender":"6265740b687ccad4736ce56d","text":"second test message from client2 to
client1","conversationId":"6265770f687ccad4736ce578","createdAt":"2022-04-24T17:06:18.774Z","updatedAt":"2022-04-24T17:06:18.774Z","__v":0},
{"_id":"6265839a687ccad4736ce5aa","sender":"62657475687ccad4736ce56f","text":"third test message from client2 to
client1","conversationId":"6265770f687ccad4736ce578","createdAt":"2022-04-24T17:06:34.817Z","updatedAt":"2022-04-24T17:06:34.817Z","__v":0},
{"_id":"62658949d96ab5f050de2a2e","sender":"6265740b687ccad4736ce56d","text":"hello
client2","conversationId":"6265770f687ccad4736ce578","createdAt":"2022-04-24T17:30:49.267Z","updatedAt":"2022-04-24T17:30:49.267Z","__v":0},
{"_id":"6265895
```

Figure 43.Client1 post message TCP stream.

Real time client and socket server connection :

▼ 192.168.122.73:36334 ↔ 192.168.201.219:8900 (*GET*, *POST*)

```
GET /socket.io/?EIO=4&transport=polling&t=O1Svsz- HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 200 OK
Content-Length: 97
Access-Control-Allow-Origin:3000
Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Date:32 GMT
Keep-Alive: timeout=5
Vary: Origin

0{"sid":"jZEGNdB3Ek90kqhtAAAG","upgrades":["websocket"],"pingInterval":25000,"pingTimeout":20000}

POST /socket.io/?EIO=4&transport=polling&t=O1Svs-R&sid=jZEGNdB3Ek90kqhtAAAG HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Content-Length:  2
Content-Type:  text/plain;charset=UTF-8
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

40

HTTP/1.1 200 OK
Content-Length: 2
Access-Control-Allow-Origin:3000
Connection: keep-alive
Content-Type: text/html
Date:32 GMT
Keep-Alive: timeout=5
Vary: Origin
```

```
ok

GET /socket.io/?EIO=4&transport=polling&t=O1Svs-U&sid=jZEGNdB3Ek90kqhtAAAG HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 200 OK
Content-Length: 32
Access-Control-Allow-Origin:3000
Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Date:32 GMT
Keep-Alive: timeout=5
Vary: Origin

40{"sid":"SD1slbp9JISOuST1AAAH"}

POST /socket.io/?EIO=4&transport=polling&t=O1Svs_M&sid=jZEGNdB3Ek90kqhtAAAG HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Content-Length:  40
Content-Type:  text/plain;charset=UTF-8
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

42["addUser","6265740b687ccad4736ce56d"]

HTTP/1.1 200 OK
Content-Length: 2
Access-Control-Allow-Origin:3000
Connection: keep-alive
Content-Type: text/html
Date:33 GMT
Keep-Alive: timeout=5
Vary: Origin
```

```
ok

GET /socket.io/?EIO=4&transport=polling&t=O1Svs_M.0&sid=jZEGNdB3Ek90kqhtAAAG HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 200 OK
Content-Length: 160
Access-Control-Allow-Origin:3000
Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Date:33 GMT
Keep-Alive: timeout=5
Vary: Origin

42["getUsers",[{"userId":"62657475687ccad4736ce56f","socketId":"y69FBGtcPXWAX3S9AAAF"},
{"userId":"6265740b687ccad4736ce56d","socketId":"SD1slbp9JISOuST1AAAH"}]]

GET /socket.io/?EIO=4&transport=polling&t=O1Svs_i&sid=jZEGNdB3Ek90kqhtAAAG HTTP/1.1
Host: 8900
Accept:  */*
Accept-Encoding:  gzip, deflate
Accept-Language:  en-US,en;q=0.5
Connection:  keep-alive
Origin: 3000
Referer: 3000/
User-Agent: 91.0) Gecko/20100101 Firefox/91.0

HTTP/1.1 200 OK
Content-Length: 1
Access-Control-Allow-Origin:3000
Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Date:33 GMT
Keep-Alive: timeout=5
Vary: Origin

6
```

Figure 44.Client1 real time chat connection TCP stream.

Packet flow during man in the middle attack of client and host

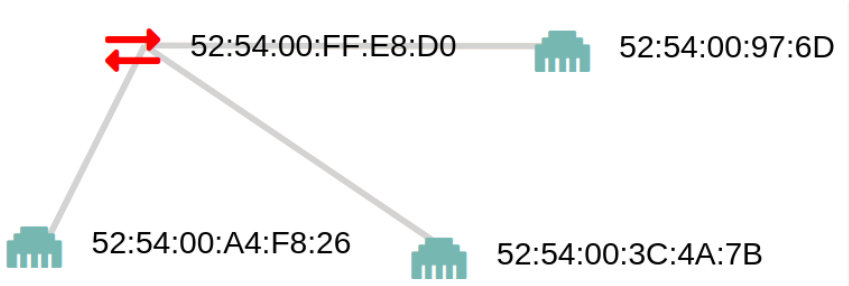


Figure 45.ARP Poisoning Man In The Middle Attack.

Conclusion and Future Work

A real time chat web application was developed using a NodeJS http server, sockets server using sockets.io and a client graphical user interface using ReactJS. The application was deployed in a test environment with 2 virtual machines as two clients and a man in the middle attacker vm. The web app front end, node http server, legitimate sockets server and man in the middle sockets server were deployed on the host machine behind a virtual bridge . MITM attack was demonstrated using the kali linux machine, malicious mitm sockets server was shown and an architecture to improve end to end application layer security for sockets server implementation was proposed.Although the ARP poisoning man in the middle attack did not succeed due to the secure implementation of ReactJS development server, we were successful in displaying the application layer man in the middle attack by appending the message hello from hacker when client1 sent message to the client2. The difficulties faced by us were to implement real time chat while running all the servers and clients in one machine using a virtual machine network . Our future work will be aimed at integrating the proposed architecture to prevent application layer man in the middle attack and test it by using advanced penetration testing techniques. We would also like to integrate jwt tokens into the system and analyse the corresponding packets between client and server.

References

[1] V. Sachdeva and S. Gupta, "Basic NOSQL Injection Analysis And Detection On MongoDB," 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), 2018, pp. 1-5, doi: 0.1109/ICACAT.2018.8933707.https://ieeexplore.ieee.org/document/8933707

[2] A. Algarni, F. Alsolami, F. Eassa, K. Alsubhi, K. Jambi and M. Khemakhem, "An Open Tool Architecture for Security Testing of NoSQL-Based Applications," 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), 2017, pp. 220-225, doi: 10.1109/AICCSA.2017.44.https://ieeexplore.ieee.org/document/8308289

[3] Gupta, Shashank & Sharma, Lalitsen. (2012). Exploitation of Cross-Site Scripting (XSS) Vulnerability on Real World Web Applications and its Defense. International Journal of Computer Applications (0975 – 8887). 60. 28-33. 10.5120/9762-3594. https://www.researchgate.net/publication/299040084_Exploitation_of_Cross-Site_Scripting_XSS_Vulnerability_on_Real_World_Web_Applications_and_its_Defense

[4]R. Banerjee, A. Baksi, N. Singh and S. K. Bishnu, "Detection of XSS in web applications using Machine Learning Classifiers," 2020 4th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), 2020, pp. 1-5, doi: 10.1109/IEMENTech51367.2020.9270052. .https://ieeexplore.ieee.org/document/9270052

[5] What Is The MERN Stack? Introduction & Examples | MongoDB .https://www.mongodb.com/mern-stack

[6] React.js + Node.js + Express + MongoDB example: MERN stack CRUD App - BezKoder .https://www.bezkoder.com/react-node-express-mongodb-mern-stack/

[7] Jinwei Zhu, Kun Cheng, Jiayang Liu, and Liang Guo. 2021. Full encryption: an end to end encryption mechanism in GaussDB. Proc. VLDB Endow. 14, 12 (July 2021), 2811–2814. DOI:https://doi.org/10.14778/3476311.3476351 http://dl.acm.org/doi/10.14778/3476311.3476351

[8] Cristian Borcea, Arnab “Bobby” Deb Gupta, Yuriy Polyakov, Kurt Rohloff, Gerard Ryan, PICADOR: End-to-end encrypted Publish–Subscribe information distribution with proxy re-encryption,Future Generation Computer Systems,Volume 71,2017,Pages 177-191,ISSN 0167-739X,https://doi.org/10.1016/j.future.2016.10.013.(https://www.sciencedirect.com/science/article/pii/S0167739X16303983)

[9] Benfano Soewito, Christian, Fergyanto E. Gunawan, Diana, I Gede Putra Kusuma, Websocket to Support Real Time Smart Home Applications,Procedia Computer Science, Volume 157,2019,Pages 560-566,ISSN 1877-0509,https://doi.org/10.1016/j.procs.2019.09.014 . 20 (https://www.sciencedirect.com/science/article/pii/S187705091931172X)

[10] Introduction | Socket.IO .https://socket.io/docs/v4/ .

[11] Asish Kumar Dalai and Sanjay Kumar Jena. 2011. Evaluation of web application security risks and secure design patterns. In Proceedings of the 2011 International Conference on Communication, Computing & Security (ICCCS '11). Association for Computing Machinery, New York, NY, USA, 565–568. DOI:https://doi.org/10.1145/1947940.1948057 http://dl.acm.org/doi/10.1145/1947940.1948057

[12]Eye on the End User: Application Layer Security.https://securityboulevard.com/2020/06/eye-on-the-end-user-application-layer-security/

[13]A3:2017-Sensitive Data Exposure | OWASP Foundation . https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure

[14]Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. 2021. Efficient Online-friendly Two-Party ECDSA Signature. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21). Association for Computing

Machinery, New York, NY, USA, 558–573. DOI:https://doi.org/10.1145/3460120.3484803 http://dl.acm.org/doi/10.1145/3460120.3484803

[15]X. Li, J. Xu, Z. Zhang, D. Feng and H. Hu, "Multiple Handshakes Security of TLS 1.3 Candidates," 2016 IEEE Symposium on Security and Privacy (SP), 2016, pp. 486-505, doi: 10.1109/SP.2016.36.https://ieeexplore.ieee.org/document/7546519

[16]Sahi, Aqeel & Lai, D.. (2015). Preventing Man-In-The-Middle Attack in Diffie-Hellman Key Exchange Protocol. 10.1109/ICT.2015.7124683. https://www.researchgate.net/publication/280722113_Preventing_Man-In-The-Middle_Attack_in_Diffie-Hellman_Key_Exchange_Protocol

[17] O. Ahmedova, Z. Khudoykulov, U. Mardiyev and A. Ortiqboyev, "Conversion of the Diffie-Hellman Key Exchange Algorithm Based on Elliptic Curve Equations to Elliptic Curve Equations with Private Parameters," 2021 International Conference on Information Science and Communications Technologies (ICISCT), 2021, pp. 1-4, doi: 10.1109/ICISCT52966.2021.9670074.https://ieeexplore.ieee.org/document/9670074

[18]Roland van Rijswijk-Deij, Mattijs Jonker, and Anna Sperotto. 2016. On the Adoption of the Elliptic Curve Digital Signature Algorithm (ECDSA) in DNSSEC. In Proceedings of the 12th Conference on International Conference on Network and Service Management (CNSM 21
2016). International Federation for Information Processing, Laxenburg, AUT, 258–262.http://dl.acm.org/doi/10.5555/3375069.3375104

[19] Mohamed N. Hassan, Ahmed S. Abo-Taleb, and Mohamed Shalaby. 2020. Securing Digital Signature Algorithm against Side Channel Attacks. In Proceedings of the 2020 International Conference on Industrial Engineering and Industrial Management (IEIM 2020). Association for Computing Machinery, New York, NY, USA, 30–34. DOI:https://doi.org/10.1145/3394941.3394947 http://dl.acm.org/doi/10.1145/3394941.3394947

[20] Alessandro Barenghi, Guido Marco Bertoni, Luca Breveglieri, Gerardo Pelosi, and Andrea Palomba. 2011. Fault attack to the elliptic curve digital signature algorithm with multiple bit faults. In Proceedings of the 4th international conference on Security of information and networks (SIN '11). Association for Computing Machinery, New York, NY, USA, 63–72. DOI:https://doi.org/10.1145/2070425.2070438 http://dl.acm.org/doi/10.1145/2070425.2070438

[21]S. Pallickara, M. Pierce, H. Gadgil, G. Fox, Y. Yan and Y. Huang, "A Framework for Secure

End-to-End Delivery of Messages in Publish/Subscribe Systems," 2006 7th IEEE/ACM International Conference on Grid Computing, 2006, pp. 215-222, doi: 10.1109/ICGRID.2006.311018. https://ieeexplore.ieee.org/document/4100475

[22]M. A. Rajan et al., "Security and Privacy for Real Time Video Streaming Using Hierarchical Inner Product Encryption Based Publish-Subscribe Architecture," 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2016, pp. 373-380, doi: 10.1109/WAINA.2016.101.https://ieeexplore.ieee.org/document/7471229

[23]J. Doerner, Y. Kondi, E. Lee and A. Shelat, "Threshold ECDSA from ECDSAAssumptions: The Multiparty Case," 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 1051-1066, doi: 10.1109/SP.2019.00024.https://ieeexplore.ieee.org/document/8835354

[24]J. Doerner, Y. Kondi, E. Lee and A. Shelat, "Secure Two-party Threshold ECDSA from ECDSAAssumptions," 2018 IEEE Symposium on Security and Privacy (SP), 2018, pp. 980-997, doi: 10.1109/SP.2018.00036.https://ieeexplore.ieee.org/document/8418649

[25]R. Lychev, S. Jero, A. Boldyreva and C. Nita-Rotaru, "How Secure and Quick is QUIC? Provable Security and Performance Analyses," 2015 IEEE Symposium on Security and Privacy, 2015, pp. 214-231, doi: 10.1109/SP.2015.21.https://ieeexplore.ieee.org/document/716302822

[26]J. A. Berkowsky and T. Hayajneh, "Security issues with certificate authorities," 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2017, pp. 449-455, doi: 10.1109/UEMCON.2017.8249081.https://ieeexplore.ieee.org/document/8249081

[27]N. Mehibel and M. Hamadouche, "A new approach of elliptic curve Diffie-Hellman key exchange," 2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B), 2017, pp. 1-6, doi: 10.1109/ICEE-B.2017.8192159.https://ieeexplore.ieee.org/document/8192159

Appendix

Appendix - A : Code for Real Time Chat application is hosted on github at this url.https://github.com/NeelChoksi/Sem6_ISM

Neel Rakesh Choksi (19BCE0990) :

Real Time Web Chat Application development using NodeJS, ReactJS, MongoDB, Sockets.io, Deployment on Virtual machine Network , Man in the middle attack using Kali linux, proposed structure for protection against Man in the middle attack at the application layer. Proposed JWT integration for authorization of requests.

Amara Hemanth Kumar (19BCT0126) :

Research and analysis on NoSQL Injection , SQL Injection , Phishing Attacks, Cross Site Scripting prevention , Man in the middle attack points