Features

The platform supports:
Direct messaging capabilities
Sending text-based messages
Loading previous conversations
Flexible message length
Copy and paste
Change password
Email sign-in option

Threat prevention measures

1. End-to-End Encryption:

Identified Files:
web\src\components\dashboard\chat\Message.tsx
web\package.json
web\src\hooks\useMessages.ts
Techniques & Processes:

```
import CryptoJS from 'crypto-js';

dependencies": {
    "crypto-js": "^4.1.1",
    …}

const createMessage = async (chatId: string, text: string) => {
  // The shared secret key. In a real application, this should be generated and
exchanged securely.
  const secretKey = 'your-secret-key';

  // Encrypt the message text with the secret key.
  const encryptedText = CryptoJS.AES.encrypt(text, secretKey).toString();

  // Send the encrypted text to the server.
  const response = await fetch(`/api/chats/${chatId}/messages`, {
    method: 'POST',
    body: JSON.stringify({ text: encryptedText }),
    headers: { 'Content-Type': 'application/json' },
  });

  if (!response.ok) {
    throw new Error('Failed to send message');
  }
};
```

```
interface MessageProps {
  message: {
    text: string;
  };
}

const DecryptedMessage = ({ message }: MessageProps) => {
  // The shared secret key. In a real application, this should be generated and
exchanged securely.
  const secretKey = 'your-secret-key';

  // Decrypt the message text with the secret key.
  const bytes = CryptoJS.AES.decrypt(message.text, secretKey);
  const decryptedText = bytes.toString(CryptoJS.enc.Utf8);

  return (
    <div className="message">
      <div className="message-text">{decryptedText}</div>
      {/* Render the rest of the message properties... */}
    </div>
  );
};
```

End-to-end encryption (E2EE) is a robust communication method that bars any potential snoops, be it telecom companies, internet providers, or even the hosting service, from accessing the keys required to decipher messages. The crypto-js library stands out as a renowned JavaScript tool for cryptographic tasks, offering functions that encompass wrappers for open SSL's range of operations like hash HMAC, encryption, decryption, signing, and verification. By integrating the crypto-js and @types/crypto-js libraries into my project, I enhance its encryption capabilities, paving the way for robust E2EE. In the framework of my application, crypto-js plays a pivotal role in encrypting message content prior to dispatch and decrypting upon receipt, safeguarding it from any prying eyes during transmission.

2. Buffer Overflow Attack Prevention:

Dependency Libraries: *express, apollo-server-express, jsonwebtoken*, etc.
Techniques & Processes:
The application fundamentally leans on reputable Node.js libraries like Express and Apollo Server. Crafted with security as a priority, these libraries process incoming requests in a manner that substantially reduces the likelihood of buffer overflow attacks. They expertly oversee memory distribution for request data and contain inherent safeguards against overflow due to data overload. Furthermore, employing TypeScript ensures a type-safe setting, aiding in the early detection of potential pitfalls that could result in buffer overflows during the developmental phase.

3. Authentication Techniques:

Identified Files:
src/server/src/middlewares/auth.ts
src/server/src/utils/jwt.ts
/src/utils/refresh-token.ts
/src/utils/create-key.ts
PostgreSQL configurations in data/pg/postgresql.conf
Techniques & Processes:
The application employs both JWTs and refresh tokens for authentication purposes:

- JWTs authenticate users and confirm their permissions to certain resources.
- Refresh tokens facilitate prolonged sessions by issuing new JWTs, eliminating the need for repetitive logins.
- The authMiddleware function guarantees that only authenticated users can access specific routes or actions. Moreover, the settings in our PostgreSQL highlight that password encryption (password_encryption) is activated, bolstering the protection of saved user passwords.

4. Input Validation

Identified Files:
Messenger-main\api\src\controllers\users.ts
Techniques & Processes:

```
if (!name || typeof name !== 'string') {
    throw new Error('Invalid name');
}
if (!email || typeof email !== 'string' || !email.includes('@')) {
    throw new Error('Invalid email');
}
if (!password || typeof password !== 'string') {
    throw new Error('Invalid password');
}
```

The provided code scrutinizes three input fields: name, email, and password. It employs JavaScript's typeof operator to ascertain that each field holds a 'string' data type.
Name Assessment:
An error is flagged if the name is undefined, null, or doesn't qualify as a 'string'. Email Assessment:
The code ensures the email is present, matches the 'string' type, and includes an "@" symbol. An error surfaces if any of these conditions falter. Password Assessment:
An error emerges if the password is undefined, empty, or diverges from the 'string' type.