Peer Review Report for Group 47 – Ebony Bratton

Vulnerabilities:

1. **Backdoor 1 - mock encryption**
   Base64 encoding is used by both clients and servers. It is only an encoding system and not encryption. It means that the "encrypted" signals are simply decoded by everyone. Attackers do not require a decryption key in order to read messages in plain text.

2. **Backdoor 2 – Signature Verification**
   Without any cryptographic verification, the create_mock_signature function generates signatures that are simply a text concatenation. As a result, a malicious user may send unsigned messages and evade signature verification, leaving the system open to forging attacks.

```
80          return web.json_response({"File was not found"}, status=404)
81
82  def create_mock_signature(data, counter):
83      return f"mock_signature_for_{json.dumps(data)}_counter_{counter}"
84
85  def validate_signature(data, signature, counter):
86      expected_signature = create_mock_signature(data, counter)
87      return True
88
```

Strengths:

1. Clear Protocol Flow: The protocol flow of the chat system is straightforward and modular, featuring distinct message types for managing server greetings, client lists, and public and private chats.

2. By adding a counter with each message, the system protects against replay attacks. It helps in making sure that repeated or duplicate messages are not received.

Weaknesses:

1. Lack of True Encryption: As previously stated, the "encryption" in the code is a mock version that offers no actual security. This would make it possible for adversaries to read and intercept plain text messages.

2. There is no validation for the user to upload the file to the chatroom and it can allow an attacker to upload harmful files. It is easy to fix just add validation function for

allowed file types and size limits on both the client and server side.

```python
async with aiohttp.ClientSession() as session:
    try:
        with open(file_path, 'rb') as file:
            data = aiohttp.FormData()
            data.add_field('file', file, filename=os.path.basename(file_path))

            async with session.post("http://localhost:8081/upload", data=data) as response:
                if response.status == 200:
                    response_data = await response.json()
                    file_url = response_data.get('file_url')
                    if not file_url:
                        print("Bad response from server")
                        return

                    file_transfer_message = {
                        "body": {
                            "type": "file_transfer",
                            "sender": username,
                            "recipient": recipient,
                            "file_name": os.path.basename(file_path),
                            "file_url": file_url,
                        }
                    }
                    await websocket.send(json.dumps(file_transfer_message))
                    print(f" You successfully sent a file to {recipient}!.")
                else:
```

Overall, the code offers a strong basis and skillfully applies important security measures like encryption. It is a user-friendly and excellent implementation of the protocol. Good job!