I. Potential coding issues

1. Potential issues in file upload processing:

o The following issues exist in the file upload part:

file\_name = "file\_" + str(int(time())) + "\_" + str(uuid.uuid4().hex)

file\_path = os.path.join("tmp", file\_name)

 Issue 1: The uploaded file name is not verified, and the file type is not checked. An attacker may upload malicious files (such as script files) and access them through the file path.

 Issue 2: The file name is generated based on timestamp and UUID, but this cannot completely avoid file name conflicts, and there is a lack of verification and filtering of malicious files.

 Suggestion: Add verification logic for file names and file types to ensure that only specific types of files (such as pictures, text files, etc.) are allowed to be uploaded. At the same time, a more complex file path management strategy can be used to avoid file name conflicts.

2. The file path is at risk of path traversal attacks:

file\_name = file\_name.replace('\\', '/')

file\_path = os.path.normpath(os.path.join(uploads\_dir, file\_name))

if not file\_path.startswith(uploads\_dir):

return web.Response(text="Access denied.\n", status=403)

o Problem: Path traversal attacks cannot be completely avoided through simple string replacement and os.path.normpath processing. If an attacker constructs a malicious path, such as using ../, other sensitive files on the server may be accessed.

o Recommendation: Restrict file names more strictly and ensure that only files in the uploads\_dir directory are allowed to be accessed. You can further use a path whitelist strategy or regular matching verification of file names.

3. Inadequate WebSocket exception handling:

o In the WebSocket connection handling part:

async def handle\_first(self, websocket):

try:

message = await websocket.recv()

...

except Exception as e:

logging.error(f"Unestablished connection closed due to error: {e}")

▪ Problem: There is no clear handling of specific exceptions that may occur (such as WebSocket connection disconnection, timeout, etc.), and the error information is simply printed after all exceptions are caught. WebSocket connection exceptions may cause service instability or message loss.

▪ Suggestion: When catching exceptions, distinguish different types of exceptions (such as websockets.exceptions.ConnectionClosedError), and design a reasonable handling mechanism for specific exceptions, such as reconnecting or sending notifications to the client.

4. Lack of strict validation of encryption protocols:

o The code uses SSL context for encrypted communication:

self.ssl\_context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

...

auth\_context = ssl.create\_default\_context(ssl.Purpose.SERVER\_AUTH)

auth\_context.load\_verify\_locations(cafile="rootCA\_cert.pem")

▪ Problem: The SSL/TLS configuration uses the default trust chain and does not specify a client certificate validation mechanism. An attacker could use a man-in-the-middle attack to eavesdrop on the communication. In addition, there is a lack of strict validation of expired and untrusted certificates.

▪ Recommendation: Enable the mutual TLS validation mechanism and ensure that the client also provides a certificate for authentication. Further strengthen the certificate validation logic to ensure that the certificate comes from a trusted authority and is not expired.

5. Insufficient message signature and counter validation:

counter = int(message\_json.get('counter'))

if user\_data.counter >= counter:

logging.warning("Warning! Counter for this message has not been incremeneted.")

o Problem: The counter validation in signature validation is too loose. The code only checks whether the counter of the current message is greater than the counter of the previous message, but does not prevent the counter from being replayed or tampered with. In addition, once the counter is predicted, an attacker can forge messages or perform replay attacks.

o Recommendation: Ensure that the counter of the message is unique, and use unpredictable elements such as random numbers or timestamps to enhance message uniqueness. In addition, more security measures can be introduced for the counter, such as signature validation based on session ID or context.

6. Insecure public and private key storage:

o Private and public keys are stored directly in files:

with open("private\_key.pem", "rb") as key\_file:

self.private\_key = serialization.load\_pem\_private\_key(

key\_file.read(), password=None, backend=default\_backend())

▪ Problem: Private key is stored in plaintext file, attacker can easily obtain the file and crack the communication. File is stored in the server file system, lacking additional security measures (such as encrypted storage or using hardware security module).

▪ Recommendation: Encrypt private key or use the operating system's secure key storage mechanism, such as hardware security module (HSM) or keystore. Also, avoid loading unencrypted private keys directly in the code.

7. Hardcoded certificate path:

auth\_context.load\_verify\_locations(cafile="rootCA\_cert.pem")

o Problem: Certificate file path is hardcoded, if the file path is compromised or the certificate is replaced, attacker can forge legitimate communication.

o Recommendation: Manage the certificate path through environment variables or secure configuration files to avoid leaking hard-coded paths. At the same time, regularly update and check the validity of the certificate.

II.backdoor

1. Backdoor risk in neighbor server connection:

o The code automatically connects to other neighbor servers and trusts their public keys:

self.servers[hostname] = ServerData(hostname, public\_key)

server\_listeners.append(self.connect\_to\_server(hostname, auth\_context))

▪ Problem: Without a strict neighbor server authentication mechanism, attackers can forge neighbor servers and implant malicious code to eavesdrop or tamper with communication content. The current configuration relies on the public key neighborhood.olaf in the local file, and attackers can forge servers by replacing this file.

▪ Recommendation: Use two-way TLS authentication and ensure that the authentication mechanism of the neighbor server is reliable. In addition, adopt a certification authority (CA)-based authentication mode to verify the legitimacy of the neighbor server.

2. Possible man-in-the-middle attacks:

o The code does not enable client certificate verification during SSL connection:

auth\_context = ssl.create\_default\_context(ssl.Purpose.SERVER\_AUTH)

Problem: The identity of the client is not verified, which means that any attacker who can forge an SSL certificate can perform a man-in-the-middle attack and eavesdrop on the communication.

Suggestion: Enable mutual TLS authentication and ensure that the client also provides a valid certificate. This prevents man-in-the-middle attacks and guarantees the identity of both parties in the communication.

3. Counter prediction and message forgery:

o Signature verification relies on a counter, but the counter is easy to predict:

counter = int(message\_json.get('counter'))

Problem: If an attacker can predict the counter value in the message, they may be able to forge a legitimate signed message, thereby deceiving the server or client.

Suggestion: The counter should be combined with other context of the session, or an unpredictable random number should be used to increase the unpredictability of the message. In addition, use a stronger message signing mechanism to prevent forgery.

4. Backdoor risk of file upload interface:

o The file upload interface does not have strict file type restrictions:

file\_name = "file\_" + str(int(time())) + "\_" + str(uuid.uuid4().hex)

file\_path = os.path.join("tmp", file\_name)

 Problem: Attackers can implant backdoors by uploading malicious files. In some cases, the uploaded files may be executed by the server or other clients.

 Recommendation: Strictly limit the types of files allowed to be uploaded, filter executable files, script files, etc. In addition, restrict the execution permissions of files to ensure that uploaded files will not be executed on the server.