表格

描述已自动生成

Coding Issues

Inadequate exception handling:

In several places, exception handling is very general and does not provide enough context.

In particular, try-except blocks only catch Exception without distinguishing between different types of exceptions.

Example:

except Exception as e:

print("Exception: ", {e})

traceback.print\_exc()

stop\_event.set()

break

Improvement suggestion: Specific exception types should be caught and handled appropriately according to different exception types.

Missing validation of user input:

When processing user input, such as sendPrivateMessage function and setFingerprint function, no input validation is performed.

Example:

participantsString = input("Who would you like to send the message to: ")

myFingerprint = input()

Improvement suggestion: Add input validation to ensure that the value entered by the user is valid and in the expected format. Especially when it comes to security-sensitive data (such as fingerprints, message targets, etc.), the security of the input should be ensured.

Sensitive data exposure:

The getPrivateKey function directly exports and sends the private key.

Example:

message = myPrivateKey.export\_key().decode()

Improvement suggestion: The private key should not be exposed or sent to any external entity. Ensure that the private key is always stored in a secure environment and that encryption operations are only performed when used locally.

Lack of verification of encryption details:

In the implementation of RSA and AES encryption and decryption in many places, the decryption results are not sufficiently verified, which may result in tampered messages passing.

Example:

receivedAESKey = myRSACipher.decrypt(receivedEncryptedKey)

Improvement suggestion: The decryption result should be verified to be in the expected format (such as signature verification, message structure integrity, etc.).

Backdoor:

Backdoor #1:

Location:

In the receiveMessages function, there is a piece of code

if data["message"] == "admin":

await asyncio.gather(getPrivateKey(websocket))

Problem: If the received message is "admin", the code will call the getPrivateKey function, which will transfer the private key to the server.

Risk: This is a very obvious backdoor implementation. An attacker can easily obtain the private key of the server or client by sending an "admin" message, resulting in a serious security vulnerability. Anyone who sends an "admin" message can trigger this mechanism.

Backdoor #2:

Location: setFingerprint function

async def setFingerprint():

print("Input fingerprint to change to: ")

myFingerprint = input()

Problem: This function allows users to arbitrarily change their fingerprints by entering a new fingerprint. This means that an attacker can pretend to be another client or user and bypass authentication.

Backdoor #3:

Location: In the message processing logic of the clientHandler function

elif data["type"] == "client\_update":

serverSockets = []

currentServerIndex = 0

for server in server\_list:

serverSockets.append(server["socket"])

if server["socket"] == activeSocket:

break

currentServerIndex += 1

if currentServerIndex == len(server\_list):

print("An unverified server is trying to send a message!!!")

continue

server\_list[currentServerIndex]["clients"] = data["clients"]

elif data["type"] == "client\_update\_request":

await asyncio.gather(sendClientUpdate(activeSocket))

else:

print("SERVER SHUTTING DOWN - BACKDOOR #3 FOUND")

break

Problem: If the message type does not match the expected processing path, the program will print "SERVER SHUTTING DOWN - BACKDOOR #3 FOUND" and shut down the server. This means that the server can be shut down remotely by sending certain types of messages.