1. Static analysis:

a. Throwing exception problem: In the coding parts, this team widely uses “try-except” to catch the exception. For example. When the server is starting up, some codes use “except Exception: pass”. This will cover the real cause of the error. And this will provide an attacker with an opportunity to attack due to the lack of rigorous checking of important operations.

b. Skip the validation of the digital signature: a problem also exists in the code when you can skip the message signature check via a backdoor. For instance, the environment variable BACKDOOR_TRUST_GOSSIP is defined in the server.py, and when it is set to 1, the server will not verify the digital signature of incoming Gossip messages. When the backdoor is open, it will print the BACKDOOR log even though the signature is missing.

c. Allowance of weak RSA key: The design of the protocol recommends the need for RSA-4906; however, the code allows and accepts the weak RSA key(1024) and it will cause the problem of fake security.

2. Dynamic analysis:

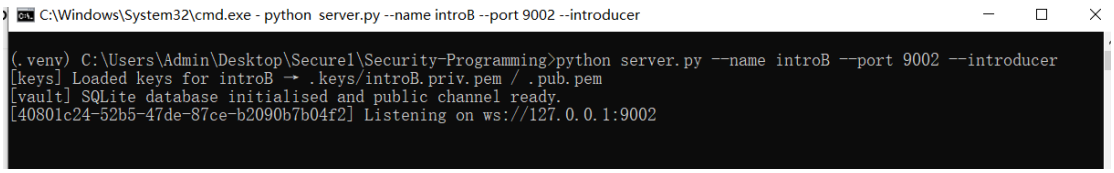
a. Wireshark capturing Packet network analysis: I use Wireshark to verify the behaviour of the backdoor. When the server is starting up and tampered messages are sent, we can tell that unsigned forged messages propagate through the network, and the server still accepts them for processing.

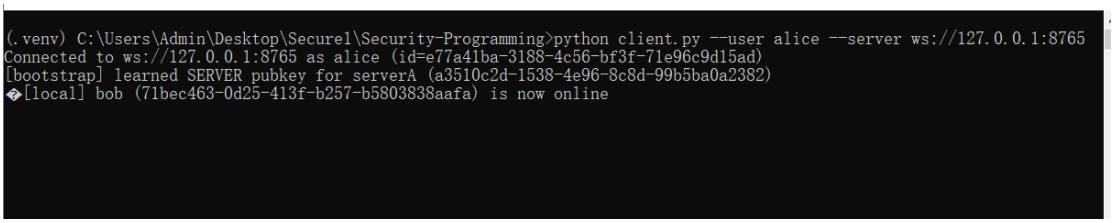
3. Backdoors and vulnerability

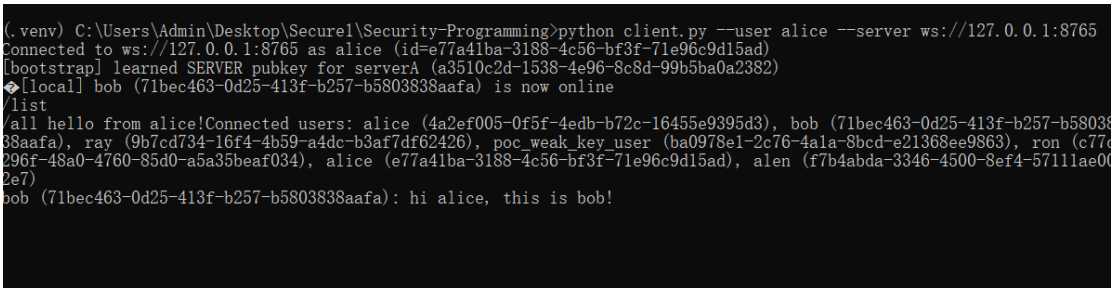
a.Weak key acceptance backdoor: This system intentionally injects a backdoor. This backdoor is triggered via an environment variable to release the key requirements, and the 1024-bit RSA key could be computed in modern technology.

b.Unsigned gossip accepts backdoors: The system will allow the server to accept cross-server advertised messages without digital signature authentication. An attacker can send fake user advertisements or takedown messages that will be accepted by other servers in backdoor mode.

4. Output:







```
(.venv) C:\Users\Admin\Desktop\Secure1\Security-Programming>python client.py --user bob --server ws://127.0.0.1:8765
Connected to ws://127.0.0.1:8765 as bob (id=71bec463-0d25-413f-b257-b5803838aafa)
[bootstrap] learned SERVER pubkey for serverA (a3510c2d-1538-4e96-8c8d-99b5ba0a2382)
[remote] alice (e77a41ba-3188-4c56-bf3f-71e96c9d15ad) has joined the network via server a3510c2d
/tell alice hi alice, this is bob!
```

```
(.venv) C:\Users\Admin\Desktop\Secure1\Security-Programming>python server.py --name serverA --port 8765
[keys] Loaded keys for serverA → .keys/serverA.priv.pem / .pub.pem
[bootstrap] OK via ws://127.0.0.1:9001 → assigned_id=a3510c2d-1538-4e96-8c8d-99b5ba0a2382
[bootstrap] Using server ID: a3510c2d-1538-4e96-8c8d-99b5ba0a2382
[vault] SQLite database initialised and public channel ready.
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] Listening on ws://127.0.0.1:8765
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] New connection received.
[vault] Registered user alice in DataVault.
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] User alice (e77a41ba-3188-4c56-bf3f-71e96c9d15ad) connected locally.
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] Sent USER_ADVERTISE (server pubkey) to e77a41ba-3188-4c56-bf3f-71e96c9d15ad
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] pubkey_b64u length = 734
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] New connection received.
[vault] Registered user bob in DataVault.
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] User bob (71bec463-0d25-413f-b257-b5803838aafa) connected locally.
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] Sent USER_ADVERTISE (server pubkey) to 71bec463-0d25-413f-b257-b5803838aafa
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] pubkey_b64u length = 734
[a3510c2d-1538-4e96-8c8d-99b5ba0a2382] Delivered LOCAL message bob -> alice
```