**Group 9 members**

CHI KEI LAO

GUO YIN HE

HARRY HUNG JUN WONG

SIJIN YANG

ZEYU LIU

## # Feed Back

### ## User one feedback

Vulnerability 1 - Hardcoded AES key found in codebase, highly dangerous and easy to exploit just by looking up the string inside the binary. With this key, the attacker may decrypt any message between this user and any other users. A suggested remedy is, a random AES key should be exchanged via asymmetric encryption for each session that the user initiates.

Vulnerability 2 - The user with a special username called `backdoor\_admin` is able to perform admin tasks without further authentication, which is bad practice. Malicious actors might try to register this username and gain unlimited administration privilege. This feature should be patched and no user admin should exist from the ground up.

Anonymous User , 29 Jul at 14:28

### ## User two feedback

There’s usage of try/except blocks in various modules, which is good. Some error handling could be more specific (avoid broad except clauses).

Anonymous User , 30 Jul at 15:16

### ## User three feedback

Review:

Command-Line Args: Support CLI flags for config overrides (e.g., --port 1234).

Chat History: Implement persistent message storage/retrieval in the DB.

User Auth: Add basic authentication (even pre-shared keys) for clients.

Anonymous User , 30 Jul at 15:20

### ## User four feedback

Although the encryption mechanism uses AES-GCM, there is no field-level verification or additional identity signature.

The lack of client identity verification mechanism will allow the server to be attacked by forged connection.

HOST = "127.0.0.1" PORT = 65432

The port number is hardcoded and there is no environment isolation.

database\_manager.py / database\_manager\_cockroachdb.py

There is a local or remote database, but no link control or isolation processing is observed.

Overall, the functions are very comprehensive and the structure is quite complete. The backdoor design is also relatively concealed.

Anonymous User , 1 Aug at 9:37

### ## User five feedback

For Group 9-

Strengths-

1.Setup of login and register is good and making sure the duplicate usernames cannot register.

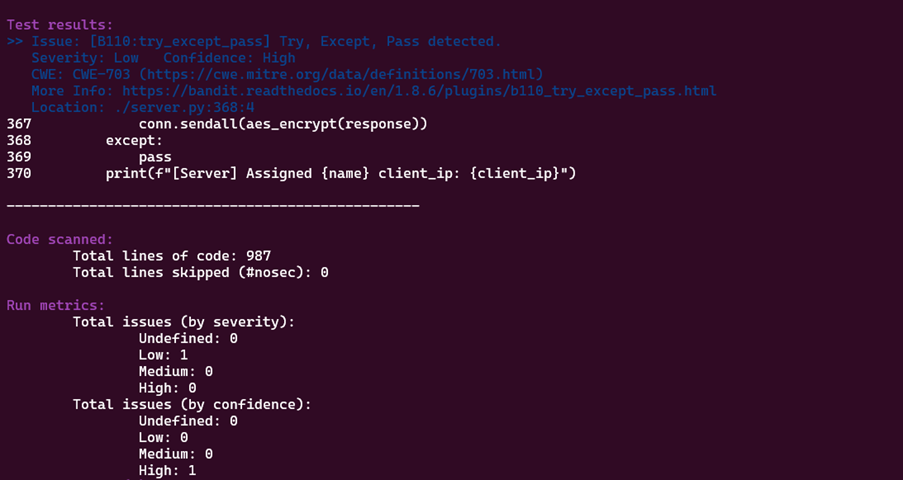
2.README.md instructions are truly clear and understandable, by reading that once, setup was easy and successful.

3.Files setup and structure is also good.

4.Server is also handling all edge cases like login failures, usernames not found and others.

Static Analysis and Manual Code testing

Found the vulnerabilities in server.py file, please find the screenshot of found vulnerability in the code by static analysis tool (Bandit)



Explanation and recommendation to fix the above vulnerability: The program has issues with silent failures, causing inconsistent behaviour when clients are not fully initialized due to network problems or socket closures. This can lead to resource leaks and odd behaviour. There is also a lack of logs, making it difficult to identify client failures and troubleshoot issues. The use of a bare except statement captures all kinds of errors, including programming mistakes, making it hard to shut down gracefully. To improve the situation, it is suggested to identify specific network exceptions like TimeoutError and others, log the errors with context, clean up the system state by removing clients and closing sockets, stop further processing, and avoid using bare except statements. Instead, a broad except statement should be used only to record errors and quit.

2.Explanation and recommendation to fix another vulnerability which is found in server.py file by manual code analysis:

AES key that is shared and hardcoded (total compromise if revealed)

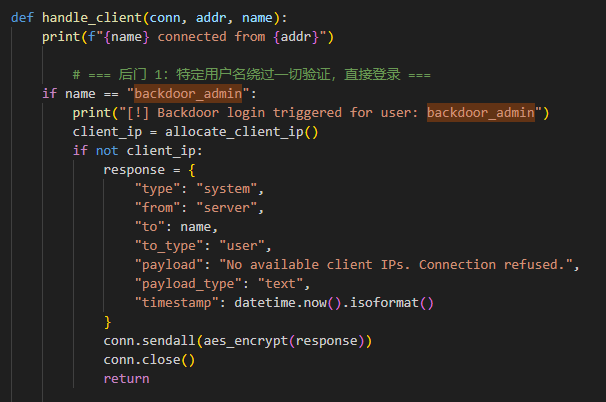
• Where in the code: At the top,

AES\_KEY = b"0123456789abcdef0123456789abcdef".

• Reason: The same fixed symmetric key is used for every client/server message. Anyone who gains access to it may imitate their peers, decrypt or create any traffic from the past and future, and move laterally. No key rotation or confidentiality between sessions.

• Fix: The ideal solution is to eliminate the custom AES and end TLS at the socket level, using mutual TLS for server-to-server connections. If it is necessary to keep AEAD, use temporary per-session keys, like an X25519 ECDH handshake followed by AES GCM, and rotate these keys frequently. Store long-term keys in a Key Management Service or secret store instead of placing them in code or in a. env file that is saved in version control systems.

3 Explanation and recommendation to fix another vulnerability which is found in server.py file by manual code analysis:  The comment even labels it as backdoor 1 and if any user will write as “backdoor\_admin”  as the username while register, then bypass authentication entirely and here no password is needed, you will log in straightaway. So, that user can have escalation privileges and even, if passwords are changed, the bypass will remain.

To fix this- Ensure the standard authentication for all the users and log all the login attempts clearly, furthermore, remove the backdoor code entirely.

Explanation and recommendation to fix another vulnerability which is found in server.py file by manual code analysis: Backdoor 2, which is identified in the comment, is introduced by this code. It determines whether a user has sent a message of the 'fake\_announce' type. The user is denied access if they are not "backdoor\_admin". To get around group ownership checks, however, a user with the username "backdoor\_admin" is permitted to send phony announcements to a group while posing as the group owner. This essentially creates a privilege escalation vulnerability that allows "backdoor\_admin" to write to group announcements without authorization.



To fix this- Remove the backdoor code entirely and validate the group ownership properly and log all privileged actions.

Snyk static tool – I used this tool too but did not get anything after analysation.

Dynamic Analysis

I used Valgrind for the dynamic analysis for the server and client.py files and got the output that is shown below,

A computer screen shot of white text

AI-generated content may be incorrect.

By analysing the output above, you can see definitely lost memory is 2040 bytes which is actually needs to be considered as it is actual memory leak and possibly lost bytes is 4608 which is also needs to be considered as in future can be memory leak. To fix this problem- first find the source of real leak by using valgrind -v with --track-origins=yes and then interpret the stack tree and try to free the memory you allocated with malloc() or new() , if memory leak is inside the libraries that you used the update them , moreover can use the python memory tools to check more about it.

Anonymous User , 1 Aug at 12:31