COMP.SEC.300

PRECIOUS IDORO

TESTING THE SECURITY OF AN AI-BUILT APPLICATION: SECURE FILE ENCRYPTION/DECRYPTION TOOL IN A CLIENT-SERVER SETTING

INTRODUCTION

Generative Artificial Intelligence (GEN AI) has become a big part of everyday life for many people across varying fields, including programming. Their programming abilities have also grown tremendously over the past few years. However, considering the cyber threat landscape, it is not enough to write fully functional programs. It is essential to ensure that they are also secure.

Thus, this project aims to ascertain the security of an application built with Chatgpt 4o mini. The idea here is to give just enough context to the AI application to understand that security is expected while providing as minimal help as possible. The prompts used to achieve the project can be summarised as follows:

- Stating the context and requirements of the project, with emphasis on security, and requesting a project outline
- Building the application
- Troubleshooting errors
- Evaluation against OWASP criteria
- Usability and security tests

A comprehensive list of the relevant prompts used is provided in Appendix A.

APPLICATION OVERVIEW:

A secure client-server application for encrypting and decrypting files, ensuring the confidentiality, integrity, and safety of data in transit and at rest. The server handles encryption and decryption requests using strong cryptographic algorithms, and the client provides an interface for users to interact with the system securely. For instance, the client contains dialog boxes such as encryption, decryption, and password dialog boxes that allow the user to select the desired document and provide a password that serves as the encryption key. After encryption, the encrypted file is saved to a folder on the local machine. The user can then send the file to anyone or leave it saved if that is what is desired. In the same way, the file is decrypted using the same password, and the decrypted file is saved to a download folder on the local machine. The communication between client and server is protected using HTTPS/TLS.

Technology Stack:

- Programming Language: Python
- Backend Framework: Flask API
- Encryption Library: cryptography
- Hashing: HMAC (within AES-GCM)
- Communication: HTTPS/TLS
- Client: Python GUI (Tkinter)
- Authentication: API keys

System Architecture:

Client:

- Uploads files for encryption/decryption.
- Sends encryption keys or passwords securely.
- Downloads encrypted or decrypted files.

Server:

- Receives and processes file requests.
- Encrypts and authenticates files using AES-GCM.
- Derives encryption keys from passwords securely.
- Returns encrypted or decrypted files to the client.

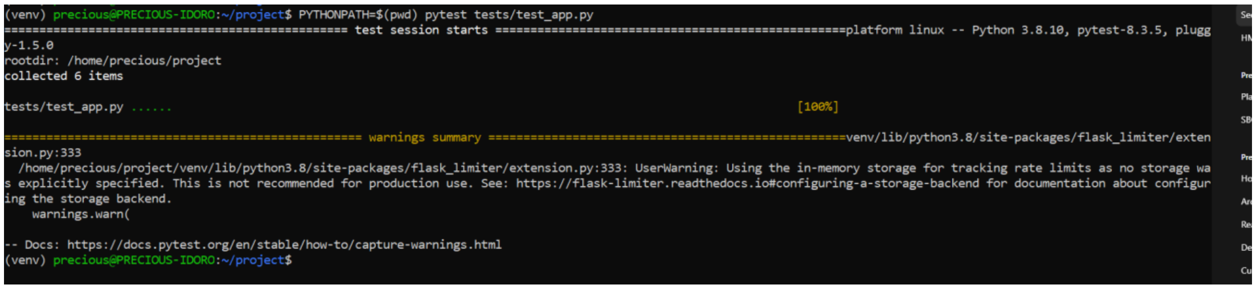
Security Measures Implemented:

- HTTPS/TLS for secure client-server communication.
- AES-GCM for authenticated encryption.
- PBKDF2 for key derivation.
- API key client authentication.
- Rate limiting to prevent brute force attacks.

TESTS AND RESULTS

Various tests were carried out, including unit and automated tests using Pytest, cyclonedx-py, pip-audit, pipdeptree, and Bandit. The manual unit tests include encryption and decryption of .txt files. The pytest script was used to test various endpoints, including encryption and decryption correctness, and Flask routes to ensure that the application logic performs as intended and that edge cases are handled to prevent logic bugs that could result in vulnerabilities. SBOM was generated using cyclonedx-py to determine all the application dependencies and enable vulnerability tracking. Pipdeptree was used to visualise the dependency hierarchy of the generated SBOM. Pip-audit was used to scan installed packages to highlight outdated or insecure dependencies. Finally, Bandit was used to analyse the code for common security issues such as insecure functions or unsafe data handling.

The images below show some of the test results.



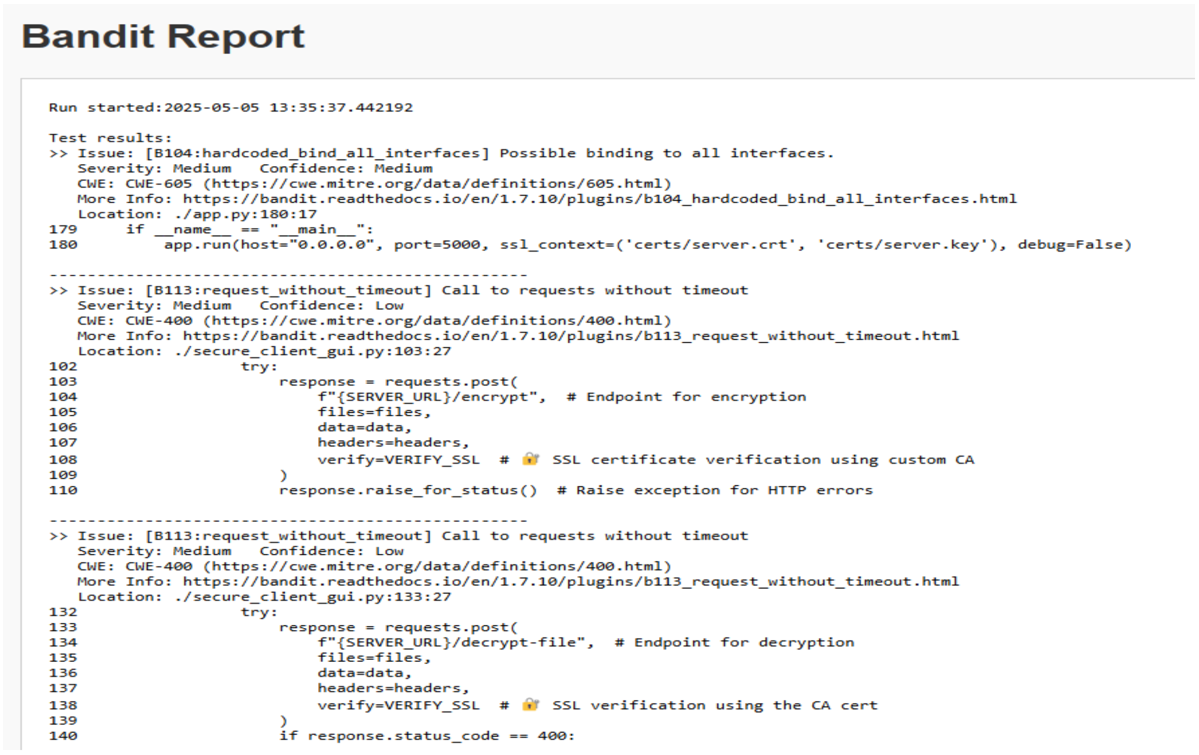
```
(venv) precious@PRECIOUS-IDORO:~/project$ PYTHONPATH=$(pwd) pytest tests/test_app.py
================================================ test session starts ==================================================platform linux -- Python 3.8.10, pytest-8.3.5, plugg
y-1.5.0
rootdir: /home/precious/project
collected 6 items

tests/test_app.py ......                                                                                         [100%]

=================================================== warnings summary ===================================================venv/lib/python3.8/site-packages/flask_limiter/exten
sion.py:333
  /home/precious/project/venv/lib/python3.8/site-packages/flask_limiter/extension.py:333: UserWarning: Using the in-memory storage for tracking rate limits as no storage wa
s explicitly specified. This is not recommended for production use. See: https://flask-limiter.readthedocs.io#configuring-a-storage-backend for documentation about configur
ing the storage backend.
    warnings.warn(

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
(venv) precious@PRECIOUS-IDORO:~/project$
```

Figure 1 Pytest Scan of Endpoints



Bandit Report

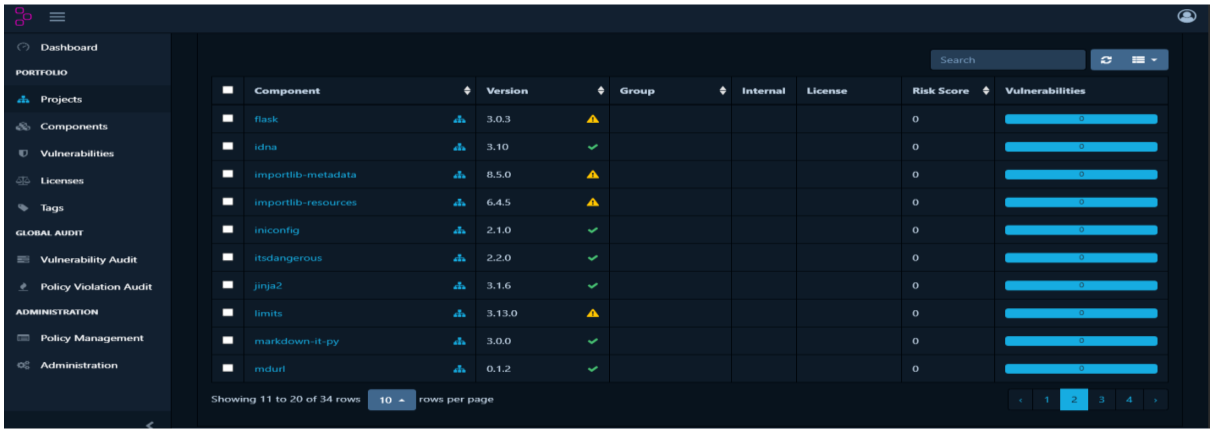
```
Run started:2025-05-05 13:35:37.442192

Test results:
>> Issue: [B104:hardcoded_bind_all_interfaces] Possible binding to all interfaces.
   Severity: Medium   Confidence: Medium
   CWE: CWE-605 (https://cwe.mitre.org/data/definitions/605.html)
   More Info: https://bandit.readthedocs.io/en/1.7.10/plugins/b104_hardcoded_bind_all_interfaces.html
   Location: ./app.py:180:17
179	    if __name__ == "__main__":
180	        app.run(host="0.0.0.0", port=5000, ssl_context=('certs/server.crt', 'certs/server.key'), debug=False)

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.10/plugins/b113_request_without_timeout.html
   Location: ./secure_client_gui.py:103:27
102	                try:
103	                    response = requests.post(
104	                        f"{SERVER_URL}/encrypt",  # Endpoint for encryption
105	                        files=files,
106	                        data=data,
107	                        headers=headers,
108	                        verify=VERIFY_SSL  # 🔐 SSL certificate verification using custom CA
109	                    )
110	                    response.raise_for_status()  # Raise exception for HTTP errors

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Call to requests without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.10/plugins/b113_request_without_timeout.html
   Location: ./secure_client_gui.py:133:27
132	                try:
133	                    response = requests.post(
134	                        f"{SERVER_URL}/decrypt-file",  # Endpoint for decryption
135	                        files=files,
136	                        data=data,
137	                        headers=headers,
138	                        verify=VERIFY_SSL  # 🔐 SSL verification using the CA cert
139	                    )
140	                    if response.status_code == 400:
```

Figure 2 Bandit Report in HTML



Component	Version		Group	Internal	License	Risk Score	Vulnerabilities
flask	3.0.3	⚠				0	0
idna	3.10	✓				0	0
importlib-metadata	8.5.0	⚠				0	0
importlib-resources	6.4.5	⚠				0	0
iniconfig	2.1.0	✓				0	0
itsdangerous	2.2.0	✓				0	0
jinja2	3.1.6	✓				0	0
limits	3.13.0	⚠				0	0
markdown-it-py	3.0.0	✓				0	0
mdurl	0.1.2	✓				0	0

Showing 11 to 20 of 34 rows 10 rows per page

Figure 3 Dependency Track Overview from SBOM

A summary of the issues discovered from the tests is shown below.

Table 1 Summary of Identified Issues and Fixes

S/N	Issue	Application	Description	Severity	Fix
1	Hardcoded bind all interfaces (B104)	Server	Application configured to bind to 0.0.0.0, causing the server to accept connections from any network interface and potentially exposing it to external attacks.	Medium	Restricted host to local access only at 127.0.0.1
2	Requests without timeout (B113)	Client	Requests.post calls for encryption and decryption lack timeout parameters. This could lead to a denial of service in the event that the server does not respond to a call.	Medium	A timeout parameter of 10 seconds was added to requests.post.
3	Outdated packages and libraries	General	Packages and libraries such as pytest and werkzeug were outdated at the time of testing which could potentially lead to vulnerabilities.	Low	Updated to the latest version (pending)

EVALUATION AGAINST OWASP TOP 10

A01:2021-Broken Access Control

Access control mechanisms such as API key-based authentication were implemented to restrict access to encryption/decryption endpoints to unauthorized clients. Rate limiting was also added to prevent abuse and to mitigate denial of service attacks, but this was not tested.

A02:2021-Cryptographic Failures

Strong cryptographic algorithms were utilized. For instance, AES-GCM was used for authenticated encryption; PBKDF2 was used for secure key derivation from passwords; AES-GCM was utilized for authenticated encryption. Finally, all communication is over HTTPS. However, the client has issues verifying the server's certificate for some reason.

A03:2021-Injection

There are no typical injection risks. However, a potentially dangerous call, eval(), which was initially used for handling the encrypted data, was changed to JSON.loads().

A04:2021-Insecure Design

A secure-by-design approach, including strong cryptographic algorithms, was used. However, the tests revealed some issues, as shown in Table 1 above. No formal threat modeling was conducted.

A05:2021-Security Misconfiguration

Generally, this issue was avoided by ensuring no routes or headers were exposed, Flask debug mode was turned off, and no sensitive files were publicly accessible.

A06:2021-Vulnerable and Outdated Components

No libraries with known vulnerabilities were used in the code. The outdated components found by pip-bandit were essentially those used during the setup of the programming environment.

A07:2021-Identification and Authentication Failures

Since the project was not built to have multiple users and is just a simple client-server model, there is no provision for login mechanisms. However, API key authentication was implemented to verify clients, and TLS was implemented to ensure credentials are encrypted in transit.

A08:2021-Software and Data Integrity Failures

Some measures were taken to conform to this. For instance, AES-GCM was utilized for authenticated encryption. But not much else was done regarding this.

A09:2021-Security Logging and Monitoring Failures

Nothing was done to prevent this.

A10:2021-Server-Side Request Forgery

This is not relevant to the architecture of this application since the server does not make outbound requests on behalf of users.

OBSERVATIONS & LIMITATIONS

ChatGPT 4o mini can be trusted, to an extent, to generate codes that align with secure programming principles. However, this depends on the prompt provided by the user. The prompt must be detailed enough for ChatGPT to understand how critical secure programming is. It is also vital that the user knows different standards and frameworks that ChatGPT can reference while generating the code. For instance, the initial code for this project generated by ChatGPT was not secure, even though secure programming requirements were explicitly given (see prompt 1 in Appendix A). However, when asked to use the OWASP top 10 checklist, it began to implement each item on the list, relevant to the application, in subsequent generations.

ChatGPT 4o mini codes may not run on the first run or even multiple runs. If the user is an experienced programmer, then they might be able to easily spot the errors and fix them. Otherwise, providing the AI tool with the details of the errors is helpful but tedious. Unfortunately, getting this free version to fix the issue may take multiple tries.

Since ChatGPT 4o mini is free, it is limited in the amount of complex analysis it can do. Thus, this project took a long time to complete. For instance, fixing some bugs took many days because it sometimes takes over 24 hours for it to be able to do complex tasks again. Thus, a project such as this with multiple parts can drag on for weeks or months.

Finally, ChatGPT 4o mini has a poor memory. It sometimes forgets where it’s at. For example, I asked it to reference the project outline it generated many times, but it kept veering off course and giving unrelated answers. I had to provide the outline and codes many times during the chat so that it would recall what stage of the project we were at.

CONCLUSION

ChatGPT 4o mini does not apply secure programming principles by default. It does, however, do so if adequately prompted and guided by someone who is knowledgeable of these principles and software development standards.

Ultimately, this work can be made more robust with more time. The scope can be expanded to test the free/paid versions of various Generative AI applications, using predefined and unified prompts across the board. It can then be presented as a study or research.

USE OF AI IN REPORT

I hereby declare that the AI-based applications used in this work are as follows:

Application	**Version**
Chat GPT	GPT-4o mini

Purpose of the use of AI: Idea generation.

Parts of this work, where AI was used: Application overview

APPENDIX A

1. *I am taking a course called Secure Programming. The idea is to ensure that software and applications are safe and secure and meet secure programming standards. Can you build a good application that aligns with these principles? Make suggestions.*
2. *Expand on the first topic*
3. *How can this be implemented in a server-client setting?*
4. *Write a project outline.*
5. *What system requirements are needed for this project?*
6. *Yeah. Let's start.*
7. *Let's do the next thing*
8. *This isn't working. Let's start again. We will write the entire code in one file. Including encryption and decryption. (chatgpt was putting the functions in separate files but wasn't working).*
9. *It is running now. (moved on to testing encryption and decryption)*
10.

```
curl -X POST http://localhost:5000/encrypt-file \
     -F "file=@testfile.txt" \
     -F "password=strongpassword123"
   <!doctype html> <html lang=en> <title>404 Not Found</title>   <h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually,
please check your spelling and try again.</p>
```
(error due to chatgpt including encrypt-file endpoint instead of encrypt in the curl request)

11. *But there is no route called encrypt-file:*

```
flask routes
Endpoint Methods Rule
------------------------------------------------------------
decrypt POST /decrypt
encrypt POST /encrypt
home GET /
static GET /static/path:filename
precious@PRECIOUS-IDORO:~/secure_file_tool$ curl -X POST http://localhost:5000/encrypt-file \

 -F "file=@testfile.txt" \

 -F "password=strongpassword123"
```

12. *let's try something else entirely. This is not working (after having consistent issues with encrypting the files. This prompted chatgpt to start over)*

13. *curl -X POST http://localhost:5000/decrypt-file \ > -F "file=@/home/precious/project/encrypted/testfile.txt.enc" \ > -F "password=strongpassword123"*

<!doctype html> <html lang=en> <title>404 Not Found</title> <h1>Not Found</h1> <p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p> (New issues when it turned out that Chatgpt included decryption routes but did not include functions for it)

14. *Alright. It is done. It works now.*
15. *Let's move on to the next phase of this project*
16. *Look at the original plan for this work. What was supposed to be the next phase? (At this point, chatgpt was derailing and didn't know the next phase of the project)*

# I provided the entire plan again to get it back on track and then it wrote an automated unit test, with instructions on how to run it.

17. *So, the tests just passed. (after the test codes failed several times)*
18. *All others worked, only the test_dowmload_decrypted_file didn't work. OS not found*
19. *Can you bring up the original plan you drafted for this project? (it was derailing again)*
20. *No, this is not it. (brought a different plan. Eventually gave it the plan again)*
21. *Let's work on the security enhancements (Shown in the provided plan)*
22. *Let's move to the next phase*
23. *No. The idea is not to have users and logins. Let's skip Jwt auth (proceeded to use API-based authentication instead)*
24. *I would like you to use some kind of OWASP checklist on this. (proceeded to create a simple checklist based on **OWASP Top 10** plus some ASVS principles applicable to this specific application, based on this chatgpt had to remove the eval() method from the code, replacing it with json.loads())*
25. *Let's move to rate limiting*
26. *Okay. Let's move on to the next thing on our original schedule*
27. *This is something new. Now I have to give you the plan again*
28. *Build a GUI client using only secure libraries and with all the features stated in the plan.*
29. *So, include different warnings in the client code, for example, when a wrong password is used for decryption. Include different warnings for likely situations. Also, include the certificate line in the code. Here's the code again…*