Task 3

SECURE FILE SHARING SYSTEM

Table of Contents

1. Introduction

In today’s digital era, organizations increasingly rely on electronic data storage and secure transfer mechanisms to protect sensitive information, including personal records, corporate assets, and government data. Any unauthorized disclosure, manipulation, or hijacking of such information can result in financial loss, reputational damage, regulatory penalties, and diminished trust. Ensuring the confidentiality, integrity, and availability of files has therefore become a critical pillar of cybersecurity.

This project addresses these challenges by developing a secure file upload and download portal that safeguards data both at rest and in transit. The system employs AES-256 in Galois/Counter Mode (AES-GCM) for strong encryption, ensuring confidentiality and providing built-in integrity verification. Secure transmission is achieved using HTTPS (TLS/SSL), preventing eavesdropping or tampering during communication. Additionally, an API token-based authentication mechanism is implemented to simulate modern security practices such as OAuth2 or multi-factor authentication.

Other security measures include filename sanitization, database-backed metadata storage, and exclusion of sensitive files via .gitignore to prevent unintended data leakage. Beyond technical safeguards, the project serves as a learning model in areas such as secure web development, cryptography, and key management, demonstrating how theoretical security principles can be applied in practical implementations.

By replicating real-world attack surfaces like unauthorized uploads, weak authentication, and insecure storage, this project provides a defense-in-depth approach, incorporating principles of least privilege, layered security, and fail-safe defaults. It acts not only as a practical security solution but also as an educational tool for students and professionals in cybersecurity and software engineering.

2. Objectives

- Design and implement a web-based portal for secure file upload and download.
- Apply AES-256-GCM encryption to protect all stored files.
- Ensure secure communication channels using HTTPS.
- Enforce token-based authentication to restrict unauthorized access.
- Demonstrate secure key management and safe file handling practices.
- Maintain a GitHub repository with full documentation, including system architecture, walkthrough, and security design.

3. Implementation

1. Environment Setup

- cd future_task_03
- python3 -m venv venv
- source venv/bin/activate
- pip install requirements.txt

2. **AES Key Setup**

- python3 - <<'PY'
- import os, base64
- print(base64.b64encode(os.urandom(32)).decode())
- PY
- export MASTER_KEY_B64=" ST5XyjEMdv//0Mpq9dtH448/bU36qEFvGWwAnA18/BE="
- export API_TOKEN="supersecrettoken"
- export FLASK_SECRET="somesecret"

3. **Application Development**

- Create app.py
- templates/layout.html
- templates/index.html

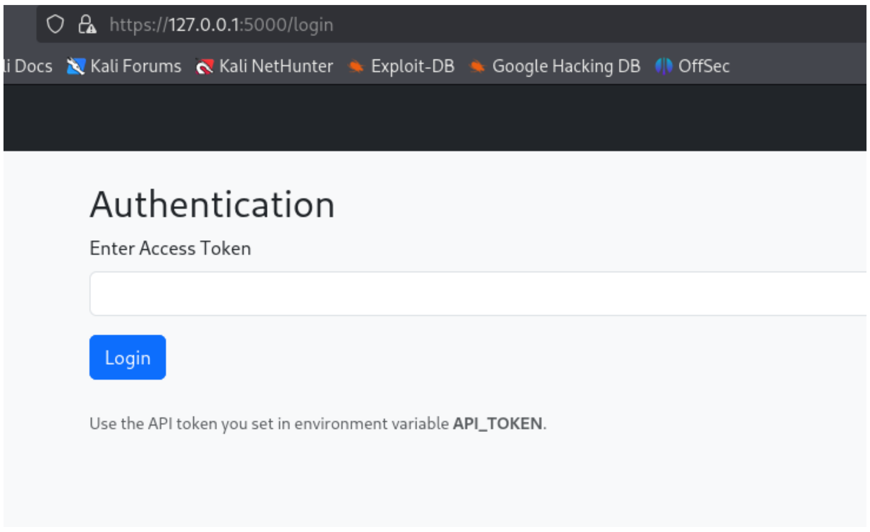
4. **HTTPS Setup**

- openssl req -x509 -newkey rsa:2048 -nodes \ -keyout key.pem -out cert.pem -days 365 \ -subj "/C=IN/ST=Karnataka/L=Bangalore/O=Internship/OU=IT/CN=localhost"

4. Run the application

- python3 app.py
- https://127.0.0.1:5000

5. Screenshots

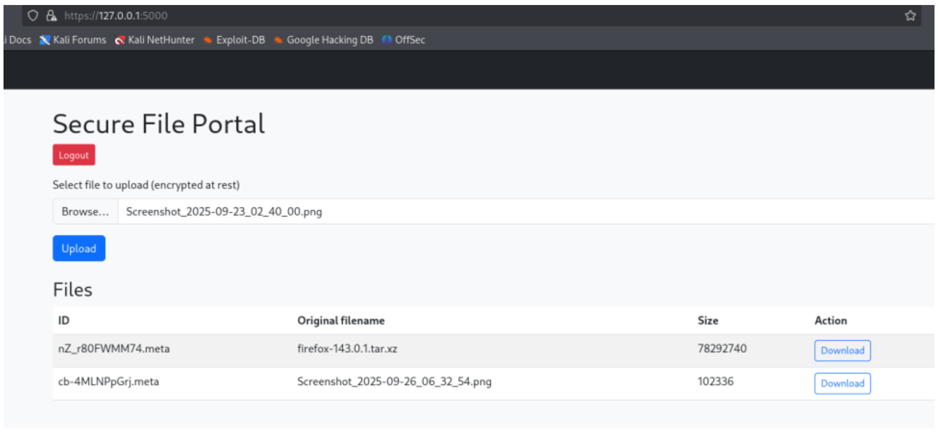


Authentication

Enter Access Token

Use the API token you set in environment variable **API_TOKEN**.

Figure 1: Authentication



Secure File Portal

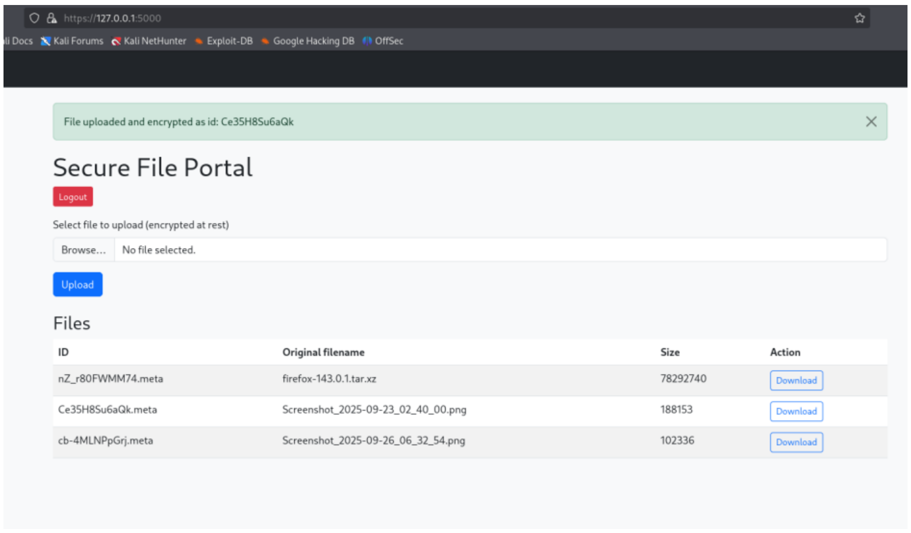
Select file to upload (encrypted at rest)

Screenshot_2025-09-23_02_40_00.png

Files

ID	Original filename	Size	Action
nZ_r80FWMM74.meta	firefox-143.0.1.tar.xz	78292740	Download
cb-4MLNPpGrj.meta	Screenshot_2025-09-26_06_32_54.png	102336	Download

Figure 2: File Uploading and Downloading page



File uploaded and encrypted as id: Ce35H8Su6aQk

Secure File Portal

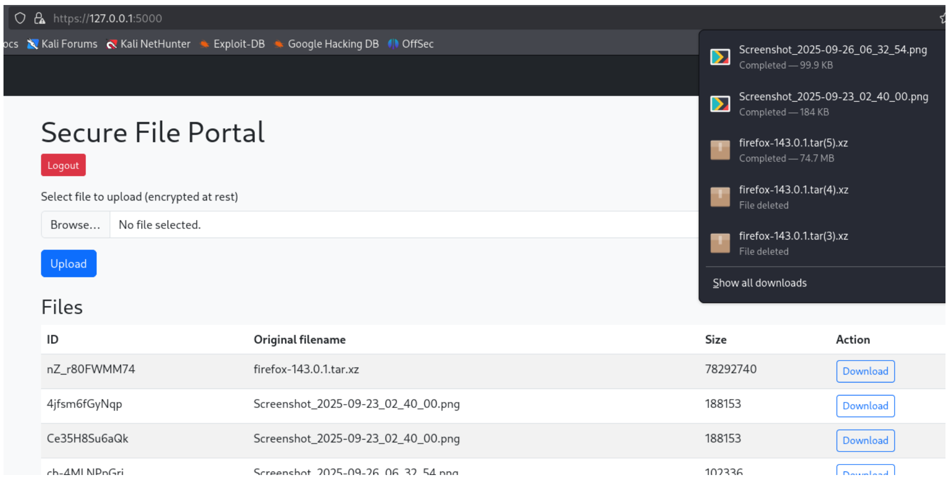
Select file to upload (encrypted at rest)

No file selected.

Files

ID	Original filename	Size	Action
nZ_r80FWMM74.meta	firefox-143.0.1.tar.xz	78292740	Download
Ce35H8Su6aQk.meta	Screenshot_2025-09-23_02_40_00.png	188153	Download
cb-4MLNPpGrj.meta	Screenshot_2025-09-26_06_32_54.png	102336	Download

Figure 3: Uploading File



Screenshot_2025-09-26_06_32_54.png
Completed — 99.9 KB

Screenshot_2025-09-23_02_40_00.png
Completed — 184 KB

firefox-143.0.1.tar(5).xz
Completed — 74.7 MB

firefox-143.0.1.tar(4).xz
File deleted

firefox-143.0.1.tar(3).xz
File deleted

Secure File Portal

Select file to upload (encrypted at rest)

No file selected.

Files

ID	Original filename	Size	Action
nZ_r80FWMM74	firefox-143.0.1.tar.xz	78292740	Download
4jfsm6fGyNqp	Screenshot_2025-09-23_02_40_00.png	188153	Download
Ce35H8Su6aQk	Screenshot_2025-09-23_02_40_00.png	188153	Download
cb-4MLNPpGrj	Screenshot_2025-09-26_06_32_54.png	102336	Download

Figure 4: Downloading File

6. Security Measures Implemented

1. AES-256-GCM encryption provides confidentiality and integrity of stored files.
2. Token-based authentication limits access to valid users.
3. Self-signed HTTPS encrypts files during transit.
4. secure_filename() safeguards against directory traversal attacks when uploading.
5. .gitignore prevents sensitive files (storage/, files.db, certs, .env) from being pushed to GitHub.
6. Rate limiting & malware scanning recommended for production environments.

7. Testing and Verification

1. **Upload Test**
   - Uploaded test files via the web interface.
   - Verified ciphertext stored in /storage/.
2. **Download Test**
   - Downloaded files through UI.
   - Confirmed AES decryption restored original content.

8. Observation

- Unauthorized access without token → rejected.
- Corrupted ciphertext → decryption failure with integrity error.
- Browser showed SSL warning (expected for self-signed cert).

9. Conclusion

The development of this secure upload and download portal highlights how cybersecurity principles can be integrated into web applications from the ground up. By implementing encryption, authentication, and secure coding practices, this project demonstrated how sensitive data can be protected and risks reduced in real-world scenarios. AES-256-GCM encryption ensured file confidentiality and integrity at rest, while HTTPS safeguarded data in transit from interception and tampering.

A key aspect of this project was the use of token-based authentication, restricting access to authorized users and reflecting real-world practices such as MFA, RBAC, and identity management systems. Additional safeguards—like filename sanitization, exclusion of sensitive files from version control, and centralized monitoring—illustrated a holistic, defense-in-depth approach that goes beyond encryption alone.

From a learning perspective, this project provided valuable exposure to secure web development, encryption methods, and access control strategies. It reinforced essential security concepts such as least privilege, layered defenses, and secure system design. Ultimately, it demonstrated that even relatively small applications can be made highly resilient when built with strong security principles, practices, and mindset.

10. Future Enhancements

To further strengthen and scale this project, the following improvements are recommended:

- **Advanced Authentication**: Upgrade from static tokens to OAuth2, MFA, or adaptive authentication for stronger user verification.
- **Enhanced Key Management**: Integrate AWS KMS, HashiCorp Vault, or similar tools for automated key rotation and secure storage.
- **Improved File Security**: Add malware scanning, file integrity checks, and restrictions on file types and sizes to prevent malicious uploads.
- **Cloud Integration & Scalability**: Shift storage to cloud platforms (e.g., AWS S3, Azure Blob) and optimize databases for handling larger workloads.
- **Centralized Monitoring**: Implement SIEM systems for real-time logging, anomaly detection, and incident response.
- **User Experience Upgrades**: Support drag-and-drop uploads, resumable transfers, progress indicators, and expiring download links for improved usability.
- **Backup & Disaster Recovery**: Introduce automated backup policies and restore mechanisms to ensure availability during system failures.