Sakshi Pawar (ID)

Secure Programming

Secure PhoneBook API - Project Report

1. INTRODUCTION

The Secure PhoneBook API is a RESTful application built using **FastAPI**, designed to demonstrate secure coding practices including:

- Proper input validation
- Safe authentication and authorization
- SQL injection protection
- Secure password comparison
- Audit logging for all actions
- Full automated testing
- Containerized deployment with Docker

This project simulates a small but security-sensitive application where users manage contact records within a protected environment. Two roles are supported, a **reader** who can list entries, and an **admin** who has full read/write access.

2. SYSTEM ARCHITECTURE

The system follows a clear modular design:

2.1 Components

- FastAPI → primary web framework
- SQLite → lightweight local database
- SQLAlchemy ORM → database layer
- Pydantic v2 → request validation
- HTTP Basic Authentication → reader / admin roles
- Docker → containerization
- pytest → test automation

2.2 Project Structure

- app.py # Main API application
- phonebook.db # SQLite database
- audit.log # Audit log for all actions
- requirements.txt # Dependency list
- Dockerfile # Container build instructions
- tests/test_api.py # 65 automated test cases

3. AUTHENTICATION & AUTHORIZATION

3.1 Authentication

The API uses HTTP Basic Auth with constant-time password comparison using:

secrets.compare_digest()

Users:

Username	Password	Role
reader	readerpass	read
admin	adminpass	readwrite

3.2 Authorization

Role-based policies ensure:

- **reader** → can only call /PhoneBook/list
- **admin** → can add or delete entries

Unauthorized attempts generate:

- HTTP 403 (Forbidden)
- An audit log entry of type AUTHZ_FAIL

4. INPUT VALIDATION

4.1 Name Validation

Names must match real-world patterns such as:

- Bruce Schneier
- Schneier, Bruce
- O'Malley, John F.
- John O'Malley-Smith
- Cher

Invalid inputs include:

- Double apostrophes (O''Henry)
- Digits (L33t Hacker)
- HTML tags (<script>)
- SQL injection strings (select * from users;)
- 4+ word names (e.g., *Brad Everett Samuel Smith*)

4.2 Phone Validation

A comprehensive set of validation patterns ensures only legitimate phone formats are accepted, including:

- 12345
- 123-1234

- (703)111-2121
- +1(703)111-2121
- +32 (21) 212-2324
- 011 701 111 1234
- 12345.12345

Rejected formats:

- 7031111234 (no separators)
- 1/703/123/1234 (slashes)
- +1234 (201) 123-1234 (invalid country code)
- ext 204
- XSS strings
- SQL injection payloads

Invalid inputs return HTTP 400 Bad Request, as required.

5. DATABASE DESIGN

A simple normalized schema:

Phonebook Table:

Field	Type	Description
id	INTEGER	Primary key
name	TEXT	Validated person name
phone_number	TEXT	Validated phone number

SQLAlchemy ORM ensures parameterized queries, protecting against SQL injection.

6. API ENDPOINTS

GET /PhoneBook/list

- Role: reader, admin
- Response: list of all phonebook entries
- Logged action: LIST

POST /PhoneBook/add

- Role: admin
- Validates: name + phoneNumber
- Rejects: duplicates
- Logged action: ADD

Example:

```
{
 "name": "Bruce Schneier",
 "phoneNumber": "123-1234"
}
```

PUT /PhoneBook/deleteByName

Parameter: ?name=<validated_name>

- Validates the provided name
- Logs either DELETE_BY_NAME or DELETE_BY_NAME_NOT_FOUND

PUT /PhoneBook/deleteByNumber

Parameter: ?number=<validated_phone>

- Validates phone number
- Logs DELETE_BY_NUMBER / NOT_FOUND

7. AUDIT LOGGING

Every action is logged to audit.log with:

- Timestamp
- Action type
- Username
- Details (name/number/error)

8. SECURITY CONSIDERATIONS

The following controls were implemented:

- **Input Validation:** Strict regex patterns to prevent malformed data and XSS.
- **Password Security:** Constant-time comparison prevents timing attacks.
- **SQL Injection Prevention:** ORM queries prevent manual string construction.
- **Least Privilege:** Two roles with minimal required permissions.
- **Error Handling:** No internal stack traces exposed to users.
- **Audit Logging:** Mandatory for all sensitive operations.
- **Containerization:** Eliminates dependency problems and isolates execution.

9. TESTING

9.1 Test Framework

- pytest
- Starlette TestClient
- 65 automated tests

9.2 What was tested

Category	Tests Performed
Authentication	Wrong password, unknown user
Authorization	Reader attempting admin actions
Name validation	All valid/invalid patterns
Phone validation	Many formats, edge cases
SQL injection	'DROP TABLE phonebook;--'
Duplicates	Prevent duplicate phone numbers
Delete operations	Success + not found cases

10. DOCKERIZATION

Dockerfile Highlights:

- Python 3.10-slim
- Non-root user
- Gunicorn + Uvicorn worker
- Exposes port 8000
- **Build Image:** docker build -t secure-phonebook .
- **Run API:** docker run -p 8000:8000 secure-phonebook
- **Run Tests Inside Container:** docker run --rm --entrypoint pytest secure-phonebook -v

11. Conclusion

The Secure PhoneBook API successfully demonstrates robust secure-coding practices:

- Full input validation
- Strong authentication & authorization
- Complete audit trail
- Safe ORM-based DB operations
- Full test automation

All security and functional requirements have been met.
The system is hardened, predictable, testable, and ready for deployment.