SOHAIB KHAN
2022551
CYBER SECURITY
SSD WEEK : 4-6

Week 4-6: Secure Coding & Initial Implementation

In this phase, I will implement secure features for my application, focusing on **secure authentication** and **database security**. Below is a step-by-step guide with code.

---

1. Secure Authentication

Objective: Implement a secure authentication system to ensure only authorized users can access the application.

User Registration

Ensure users can securely register with the system by validating input, such as ensuring passwords meet minimum strength requirements.

Hash passwords before saving them to the database using **bcrypt** to prevent storing plain-text passwords.

User Login

Implement secure login by comparing the stored hashed password with the entered password.

Use **Flask-Login** to manage user sessions and store the necessary user data during a session

---

Step-by-Step Code

a. Create a `models.py` File

This file contains the models for **User** and **Note**. We use **SQLAlchemy** to define these tables.

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

# User model
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    password = db.Column(db.String(60), nullable=False)

    def __repr__(self):
        return f"User('{self.username}')"

# Note model
class Note(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

    def __repr__(self):
        return f"Note('{self.id}', '{self.content[:20]}...')"
```

---

b. app.py File

This file will implement secure authentication using **Flask-Login** and **Flask-Bcrypt** for password hashing.

```
from flask import Flask, render_template, redirect, url_for, flash, request
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user,
current_user
from flask_wtf.csrf import CSRFProtect
from forms import RegistrationForm, LoginForm, NoteForm  # You will create these forms later

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key'  # Use a secure secret key
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
csrf = CSRFProtect(app)

# User Loader for Flask-Login
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

# Registration Route
@app.route("/register", methods=['GET', 'POST'])
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, password=hashed_password)
        db.session.add(user)
        db.session.commit()
        flash('Account created successfully!', 'success')
        return redirect(url_for('login'))
    return render_template('register.html', form=form)

# Login Route
@app.route("/login", methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            login_user(user)
            return redirect(url_for('dashboard'))
        flash('Login Unsuccessful. Please check username and password.', 'danger')
    return render_template('login.html', form=form)

# Dashboard Route
@app.route("/dashboard")
@login_required
def dashboard():
    return render_template('dashboard.html')

# Logout Route
@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

# Run the app
if __name__ == "__main__":
    app.run(debug=True)
```

---

2. Secure Database (SQLAlchemy)

Objective: Ensure that the database is secured and sensitive information like passwords are hashed before being stored.

Steps:

Use **SQLAlchemy** for database interactions, which provides a higher level of abstraction and easier management of database connections.

Use **Flask-Bcrypt** to hash passwords when storing them in the database.

The **User** model has a `password` field that stores the hashed password.

The **Note** model stores user notes, and the notes are encrypted before saving them to the database (encryption is discussed later).

Key points:

Never store plaintext passwords in the database.

Always use **parameterized queries** to prevent SQL injection attacks.

---

3. CSRF Protection

To ensure that your forms are secure against **Cross-Site Request Forgery (CSRF)** attacks, **Flask-WTF** is used to automatically add a CSRF token to every form.

Steps for CSRF Protection:

Install Flask-WTF by running: pip install flask-wtf.

Add CSRF protection to your app as shown in the app.py code (csrf = CSRFProtect(app)).

---

4. Forms for Registration and Login

Create two forms using **Flask-WTF**: one for user registration and one for login.

Registration Form (forms.py)

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms.validators import DataRequired, Length, EqualTo

class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=20)])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
```

Login Form (forms.py)

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
```

---

5. Testing and Vulnerability Mitigation

Brute Force Attacks: Protect against brute force by implementing rate-limiting for login attempts.

SQL Injection: Make sure your database interactions use **parameterized queries** to prevent SQL injection attacks.

Session Management: Make sure that session management is handled securely by **Flask-Login**, including session expiration.

Conclusion

This implementation covers secure authentication and database security using **Flask**, **Flask-Login**, **Flask-Bcrypt**, and **SQLAlchemy**. It ensures that sensitive information such as passwords is stored securely and that users can register, log in, and access their notes securely.

I can further enhance this implementation by adding additional features, such as password reset functionality, user input validation, and implementing a logging mechanism for audit trails.
