TASK 3

Secure Coding Review

Let's choose a simple web application written in Python using the Flask framework. The application has the following functionality:

1. User registration
2. User login
3. Displaying a user profile

We'll review the code for security vulnerabilities and provide recommendations for secure coding practices.

Application Code

```
from flask import Flask, request, render_template, redirect, url_for,
session
from werkzeug.security import generate_password_hash, check_password_hash
import sqlite3

app = Flask(__name__)
app.secret_key = 'supersecretkey'
DATABASE = 'users.db'

def get_db():
    conn = sqlite3.connect(DATABASE)
    return conn

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        hashed_password = generate_password_hash(password)

        conn = get_db()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO users (username, password) VALUES (?,
?)', (username, hashed_password))
        conn.commit()
        conn.close()

        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = get_db()
        cursor = conn.cursor()
        cursor.execute('SELECT password FROM users WHERE username = ?',
(username,))
        user = cursor.fetchone()

        if user and check_password_hash(user[0], password):
            session['username'] = username
            return redirect(url_for('profile'))
        else:
            return 'Invalid credentials'
    return render_template('login.html')

@app.route('/profile')
def profile():
    if 'username' in session:
        username = session['username']
        return f'Welcome, {username}!'
    return redirect(url_for('login'))

if __name__ == '__main__':
    app.run(debug=True)
```

Security Vulnerabilities and Recommendations

1. **SQL Injection:**
    - **Issue:** The current code directly uses user input in SQL queries, which is vulnerable to SQL injection.
    - **Recommendation:** Use parameterized queries or ORM(Object-Relational

Mapping) to prevent SQL injection.

```
cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)',
(username, hashed_password))
cursor.execute('SELECT password FROM users WHERE username = ?',
(username,))
```

2. **Cross-Site Scripting (XSS):**
   - **Issue:** User input is not properly sanitized before rendering it in templates.
   - **Recommendation:** Use Flask's built-in escaping mechanisms by default when rendering user data in templates.

```
<!-- Example template escaping -->
<p>{{ username }}</p>
```

3. **Cross-Site Request Forgery (CSRF):**
   - **Issue:** The application does not protect against CSRF attacks.
   - **Recommendation:** Use Flask-WTF or a similar library to include CSRF tokens in forms.

```
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect(app)
```

4. **Session Management:**

   - **Issue:** Using a hardcoded secret key for session management is not secure.
   - **Recommendation:** Use environment variables or a configuration management tool to manage secret keys securely.

```
import os
app.secret_key = os.environ.get('SECRET_KEY', 'default_secret_key')
```

5. **Password Storage:**

   - **Issue:** Although passwords are hashed, the hashing algorithm (SHA256) might not be the most secure choice.
   - **Recommendation:** Use a stronger hashing algorithm like bcrypt provided by `werkzeug.security`.

```
from werkzeug.security import generate_password_hash, check_password_hash

```

```
hashed_password = generate_password_hash(password, method='pbkdf2:sha256',
salt_length=16)
```

6. **Error Handling:**
   - **Issue:** The application may expose stack traces and sensitive information in error messages.
   - **Recommendation:** Customize error pages and disable detailed error messages in production.

```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('500.html'), 500

if __name__ == '__main__':
    app.run(debug=False)
```

Tools for Static Code Analysis

1. **Bandit:**
   - Bandit is a tool designed to find common security issues in Python code.
   - Install and run Bandit:

```
pip install bandit
bandit -r path/to/your/code
```

2. **Flake8:**
   - Flake8 is a tool for checking the style and quality of Python code, which can also help identify some security issues.
   - Install and run Flake8:

```
pip install flake8
flake8 path/to/your/code
```

Running the Tools

To ensure the recommendations and fixes are effective, you can run Bandit on the code:

```
bandit -r path/to/your/code
```

Conclusion

By addressing these security vulnerabilities and following secure coding practices, you can enhance the security of your Flask web application. Regularly using static code analyzers like Bandit and Flake8, along with manual code reviews, will help maintain and improve the security posture of your applications.