**1.Secure Coding Review: Python Web Application**

**1.1 Introduction**

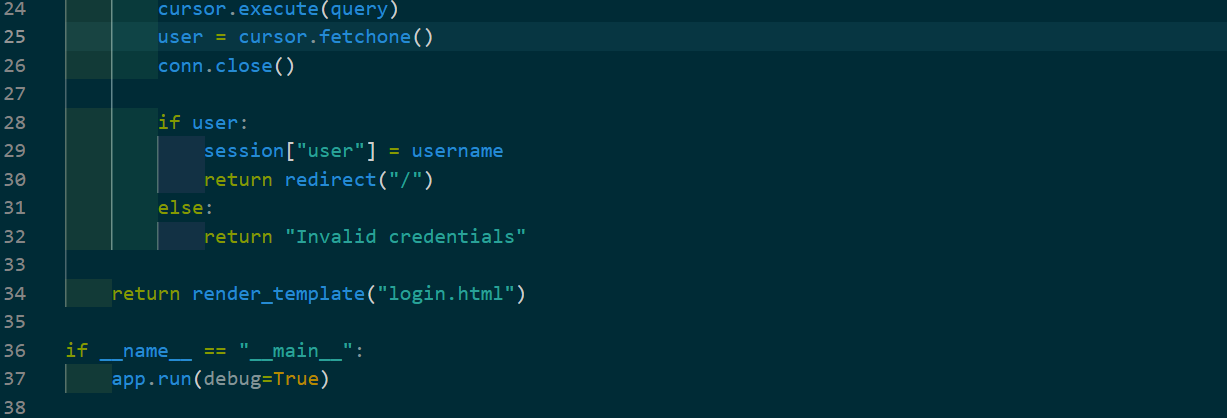
Secure coding is a crucial aspect of software development that ensures applications are protected against security threats. This review analyzes a **Python Flask web application** to identify vulnerabilities and provides recommendations for secure coding practices.

**1.2 Code Sample for Review**

The following is a **Python Flask web application** that allows users to log in.

Insecure System.



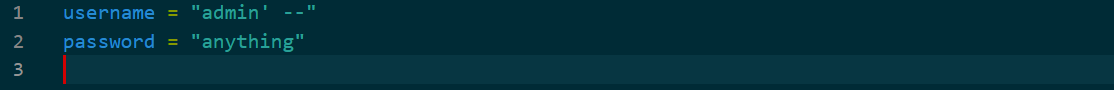


**Figure 1.2.1 Code of Insecure System**

**1.3 Security Vulnerabilities Identified**

**1.3.1 SQL Injection**

* The query f" SELECT \* FROM users WHERE username = '{username}' AND password = '{password}'" is vulnerable to **SQL injection**.
* An attacker can bypass authentication by entering:



* This will log them in without a valid password.

**1.3.2 Hardcoded Secret Key**

* The line app.secret\_key = "secret" uses a weak, hardcoded secret key.
* If exposed, attackers can manipulate session data.

**1.3.3 Plaintext Password Storage**

* The database stores passwords in plaintext, making them vulnerable if leaked.

**1.3.4. Missing Rate Limiting**

* The login form does not limit login attempts, allowing brute-force attacks.

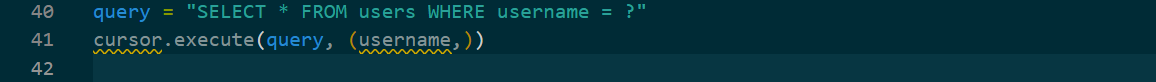
**1.3.5. Debug Mode Enabled**

* app.run(debug=True) in production exposes detailed error messages to attackers.

**1.4 Secure Coding Recommendations**

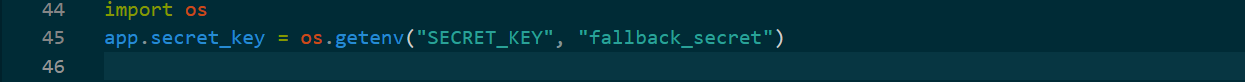
**1.4.1. Prevent SQL Injection with Parameterized Queries**

Use **prepared statements** to prevent SQL injection:



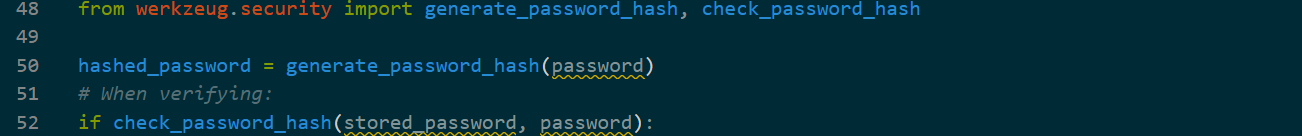
**1.4.2 Use a Strong, Environment-Based Secret Key**

Store the secret key in an **environment variable**:



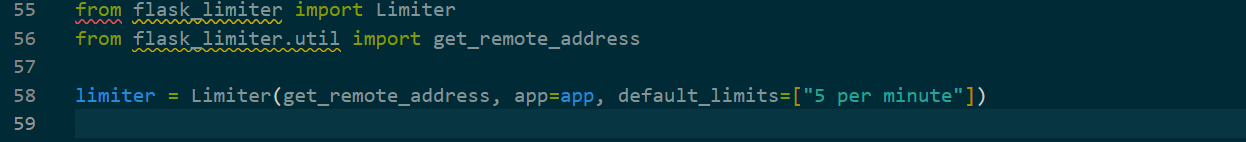
1.4.**3 Hash Passwords Before Storing**

Use **bcrypt** to store passwords securely:



**1.4.4 Implement Rate Limiting**

Prevent brute-force attacks with **Flask-Limiter**:

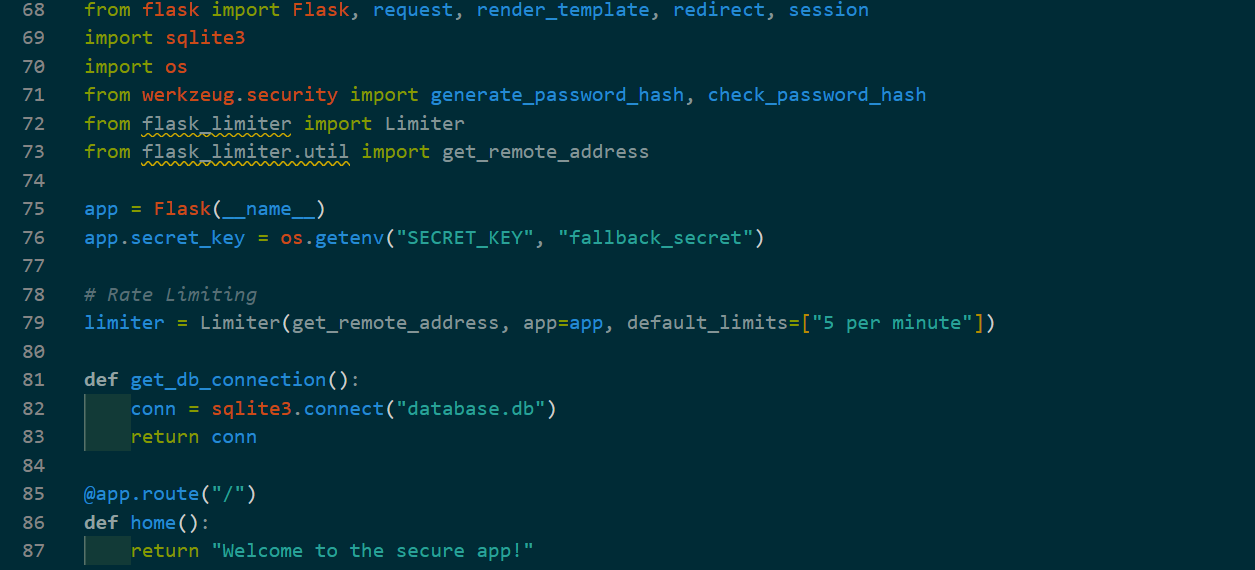


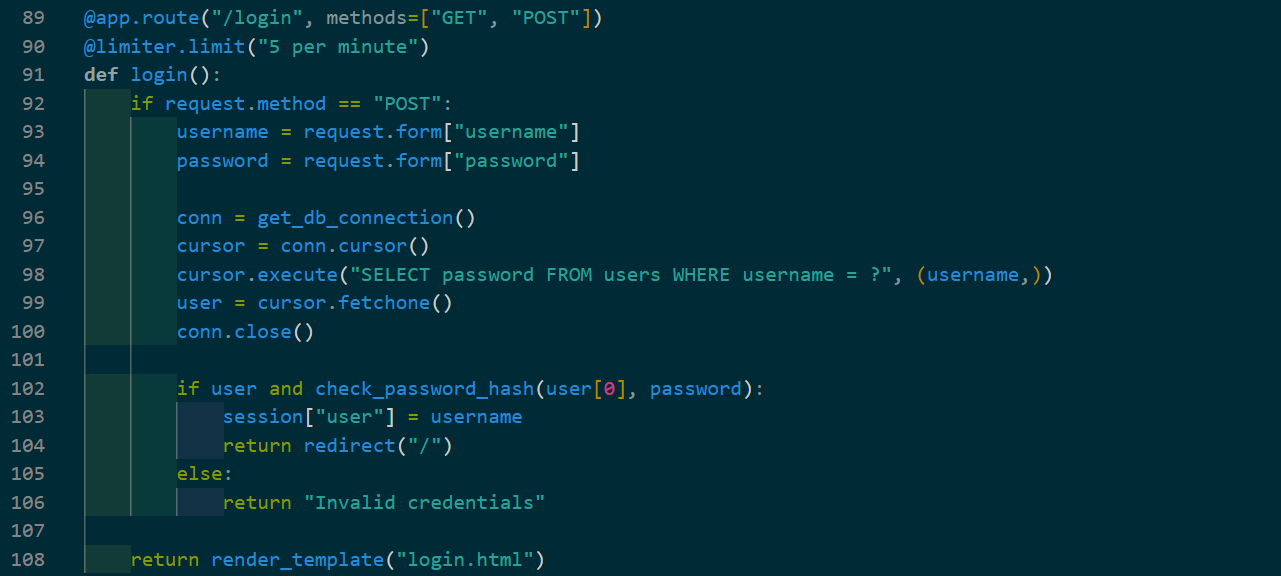
**1.4.5 Disable Debug Mode in Production**

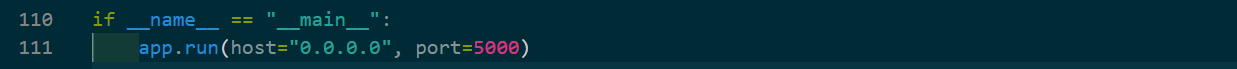
Change:



Secure Version of the Code.







**Figure 1.2.2 Code of Secure System**

**Tools Used for Code Review**

1. **Bandit** – A static code analyzer for Python security issues.  
   **Run:** bandit -r app.py
2. **Flake8** – Checks for code quality and security.  
   **Run:** flake8 app.py
3. **SQLMap** – Tests for SQL injection vulnerabilities.  
   **Run:** sqlmap -u "http://localhost:5000/login" --data "username=admin&password=test"

**Conclusion**

The original Python Flask web application contained several security vulnerabilities, including **SQL injection, plaintext password storage, hardcoded secrets, missing rate limiting, and debug mode exposure**.

By implementing **parameterized queries, hashed passwords, strong secret keys, rate limiting, and disabling debug mode**, the application's security is significantly improved.

**Following secure coding practices ensures that applications are protected against common cyber threats, making them safer for users.**