# Choose a Programming Language and Application

Consider creating a Python-based web application with Flask. This program could perform basic functions such as user registration and login.

# Review the Code for Security Vulnerabilities

Some common vulnerabilities that could exist in a Python online application:

* **SQL Injection**: If raw SQL queries are used instead of parameterized queries or an ORM (Object Relational Mapper), attackers may be able to modify the database.
* **Cross-Site Scripting (XSS)**: Malicious scripts could be introduced into websites that other people visit if user input is not adequately cleaned up.
* **Insecure Password Storage**: In the event that the system is compromised, passwords that are kept in plain text or that employ weak hashing techniques may be vulnerable.
* **Cross-Site Request Forgery (CSRF)**: Attackers could falsify requests on behalf of authenticated users if tokens are not used to protect authentication.
* **Insecure Direct Object References (IDOR)**: Attackers may obtain unauthorized access if files or database entries are directly referenced from user input.
* **Unrestricted File Uploads**: Attackers may upload malicious files (such as a PHP script masquerading as an image) if the application permits file uploads without verifying the file type.

# Security Vulnerability Analysis and Recommendations

1. **SQL Injection**:
   * **Vulnerability**: The code immediately inserts user inputs into the SQL query using string formatting.
   * **Recommendation**: To avoid SQL injection, use parameterized queries or an ORM such as SQLAlchemy.

**Fixed Code**:

python

CopyEdit

query = "SELECT \* FROM users WHERE username = ? AND password = ?"

cursor.execute(query, (username, password))

1. **Cross-Site Scripting (XSS)**:
   * **Vulnerability**: In HTML answers, user inputs (such as usernames and passwords) are utilized directly without being sanitized.
   * **Recommendation**: Employ frameworks that auto-escape output, such as Jinja2, to avoid XSS. If required, sanitize the input data as well.

**Fixed Code** (Using Jinja2 auto-escaping):

html

CopyEdit

<p>Welcome, {{ username }}!</p>

1. **Insecure Password Storage**:
   * **Vulnerability**: The code does not hash passwords; instead, it compares them directly.
   * **Recommendation**: Never save passwords in plain text; instead, hash them using a powerful hashing technique like bcrypt.

**Fixed Code** (Using bcrypt):

python

CopyEdit

import bcrypt

hashed\_pw = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

# Save the hashed\_pw, not the plain password

1. **Cross-Site Request Forgery (CSRF)**:
   * **Vulnerability**: CSRF tokens are not used by the application to secure forms.
   * **Recommendation**: Add anti-CSRF tokens to form submissions by using a CSRF protection library (such as Flask-WTF).

**Fixed Code**:

python

CopyEdit

from flask\_wtf.csrf import CSRFProtect

csrf = CSRFProtect(app)

1. **Unrestricted File Uploads**:
   * **Vulnerability**: The application might not verify the file type if a file upload function is included, which would enable hackers to upload harmful scripts.
   * **Recommendation**: Limit the size of files and verify their types. To stop execution, store files outside the web root.

**Fixed Code**:

python

CopyEdit

allowed\_extensions = {'png', 'jpg', 'jpeg', 'gif'}

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in allowed\_extensions

# Static Code Analyzers

The following static code analysers can be used to improve the security review process:

* **Bandit**: A security linter for Python that finds typical security flaws in Python code.
* **PyLint**: A multipurpose linter that may also detect security flaws, such as the usage of risky functions.
* **SonarQube**: An open-source platform that continuously checks the quality of code, including security flaws.

# Manual Code Review Process

Manual code review involves:

* **Checking Input Validation**: Before being utilized in any section of the application, make sure that every user input has been verified, cleaned, and appropriately escaped.
* **Access Control**: Make sure that sensitive actions have the appropriate role-based access control checks in place.
* **Dependency Management**: Verify that dependencies are current and check third-party libraries for known vulnerabilities.
* **Error Handling**: Make sure that no private information is revealed in error messages.