Secure Coding Review: Python Web Application (Login Functionality)

Language: Python **Application:** A simple web application with a login functionality.

Code Snippet (login.py):

Python
```
from flask import Flask, request, render_template

app = Flask(__name__)

username = "admin"
password = "password123"  # Hardcoded credentials - BAD PRACTICE

@app.route('/login', methods=['GET', 'POST'])
def login():
  if request.method == 'POST':
    user_name = request.form['username']
    user_pass = request.form['password']
    if user_name == username and user_pass == password:
      return render_template("welcome.html")
    else:
      return render_template("login.html", error="Invalid username or
password")
  else:
    return render_template("login.html")

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

Security Vulnerabilities:

- **Hardcoded Credentials:** Storing username and password directly in the code (`username` and `password` variables) is a major security risk. Attackers can potentially access the source code and steal the credentials.
- **Insecure Login Handling:**
    - The code doesn't implement any mechanisms to prevent brute-force attacks, where attackers can guess login credentials repeatedly.
    - It transmits passwords in plain text, making them vulnerable to interception during network traffic analysis.

Recommendations for Secure Coding Practices:

- **Store Credentials Securely:**
    - Use environment variables or a secure configuration file to store credentials outside the application code.
    - Consider using a password hashing mechanism to store passwords securely in a database.

- **Implement Login Security Measures:**
    - Implement techniques like rate limiting to restrict login attempts and prevent brute-force attacks.
    - Use libraries or frameworks that provide secure login functionalities, such as Flask-Login, which handle password hashing and session management.
- **Secure Data Transmission:**
    - Enable HTTPS on the web server to encrypt communication between the client and server, protecting sensitive data like passwords from eavesdropping.

Code Review Methods:

- **Static Code Analysis Tools:** Tools like Bandit or Pylint can be used to identify potential security vulnerabilities and coding practices in Python code.
- **Manual Code Review:** A thorough manual review of the code with security best practices in mind can reveal security issues.