Vulnerability Explanation - Vulnerable Flask App

This document explains the vulnerabilities intentionally built into the vulnerable Flask application as part of the cybersecurity assessment task. Each vulnerability includes its description, location in the code, the potential risk, and recommended mitigation strategies.

# 1. Broken Access Control (OWASP A01:2021)

* Location: /login endpoint (login() function)
* Description: After successful login, the application exposes the full list of users including roles and usernames without enforcing any access control based on user privileges.
* Risk: Any authenticated user can see sensitive user data, potentially leading to information leakage and privilege abuse.
* Recommended Fix: Enforce strict role-based access control. Only users with 'admin' role should be allowed to access user data.

# 2. Cryptographic Failures (OWASP A02:2021)

* Location: /login endpoint (login() function)
* Description: Passwords are compared directly in plaintext without any hashing or cryptographic protection. No secure password storage practices are implemented.
* Risk: Plaintext password handling exposes the application to credential theft and offline brute-force attacks.
* Recommended Fix: Use secure password hashing algorithms like bcrypt or Argon2 to hash passwords before storing and during comparison.

# 3. Injection - SQL Injection (OWASP A03:2021)

* Location: /query endpoint (query() function)
* Description: User input from the 'username' parameter is directly embedded into SQL queries without sanitization, allowing potential SQL Injection attacks.
* Risk: An attacker can manipulate SQL queries to access unauthorized data or modify the database.
* Recommended Fix: Use parameterized queries or prepared statements to safely handle user input in database operations.

# 4. Insecure Design - Command Injection (OWASP A04:2021)

* Location: /ping endpoint (ping() function)
* Description: User input from the 'target' parameter is directly passed to the operating system's 'ping' command without validation or sanitization.
* Risk: Attackers can inject arbitrary system commands leading to remote code execution and server compromise.
* Recommended Fix: Validate and sanitize user input to allow only safe values (e.g., IP addresses or domain names). Prefer using safe libraries instead of direct OS command execution.

# 5. Security Misconfiguration - Cloud Storage (OWASP A05:2021)

* Location: /upload endpoint (upload() function)
* Description: AWS S3 access credentials (access key and secret key) are hardcoded inside the application code, and files are uploaded to a simulated public bucket without access controls.
* Risk: Exposure of AWS credentials can lead to unauthorized access, data breaches, and full account compromise.
* Recommended Fix: Store API keys securely using environment variables or secret management systems (e.g., AWS Secrets Manager). Apply strict access policies to cloud resources.