# Remediation Explanation – Vulnerable Flask App

This document explains the **remediation measures** implemented to address the intentionally built vulnerabilities in the vulnerable Flask application.  
Each section outlines the fixed vulnerability, where it was located in the code, what was changed, and why the fix mitigates the risk.

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

* **Location**: /login endpoint (login() function)
* **Remediation**:  
  Instead of exposing all user data after login, the application now only returns the role of the authenticated user.
* **Fix Summary**:  
  Removed the users\_exposed() function call and added role-based access logic.
* **Security Outcome**:  
  Prevents unauthorized access to other users’ information, aligning with Role-Based Access Control (RBAC) principles.

### 2. ****Cryptographic Failures (OWASP A02:2021)****

* **Location**: /login endpoint (login() function)
* **Remediation**:  
  Passwords are now stored and compared using the **bcrypt** hashing algorithm.
* **Fix Summary**:
  + Passwords are hashed during database initialization.
  + During login, bcrypt is used to securely check the password.
* **Security Outcome**:  
  Prevents plaintext password exposure and resists brute-force and credential replay attacks.

### 3. ****Injection – SQL Injection (OWASP A03:2021)****

* **Location**: /query endpoint (query() function)
* **Remediation**:  
  All SQL queries now use **parameterized statements** with placeholders (?) instead of string concatenation.
* **Fix Summary**:  
  Replaced vulnerable query construction with cursor.execute("SELECT ... WHERE username = ?", (username,))
* **Security Outcome**:  
  Blocks malicious input from being executed as SQL commands, eliminating SQL injection vectors.

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

* **Location**: /ping endpoint (ping() function)
* **Remediation**:  
  Removed use of os.system() and replaced it with a simulated response after validating the input.
* **Fix Summary**:  
  Input is checked to ensure it only contains alphanumeric characters and periods (e.g., IP/domain).
* **Security Outcome**:  
  Prevents execution of arbitrary shell commands, thus eliminating the risk of remote code execution via command injection.

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

* **Location**: /upload endpoint (upload() function)
* **Remediation**:  
  Removed hardcoded AWS credentials from the source code. The application now uses **environment variables**.
* **Fix Summary**:
  + Access key and secret are read via os.environ.get().
  + Input filenames are validated (e.g., must end in .txt).
* **Security Outcome**:  
  Ensures secret credentials are not exposed in source code. Also enforces safer access controls around file uploads.

### 6. ****Insecure Deserialization (OWASP A08:2021)****

* **Location**: /deserialize endpoint (deserialize() function)
* **Remediation**:  
  Replaced the use of pickle.loads() with json.loads() and allowed only form-submitted JSON strings.
* **Fix Summary**:
  + Rejected binary deserialization.
  + Accepts and parses **trusted, valid JSON only**.
* **Security Outcome**:  
  Prevents arbitrary code execution through unsafe deserialization.

### Simulated Vulnerability (Real-World CVE Demonstration)

#### Simulated Use-After-Free (CVE-2025-29824 Concept)

* **Location**: /allocate, /free, /use endpoints (allocate(), free(), use() functions)
* **Remediation**:  
  Object existence is validated before access. The application now checks if an object has already been freed.
* **Fix Summary**:
  + Used .get() safely when accessing objects.
  + Added checks to return an error if a deleted object is accessed.
* **Security Outcome**:  
  Ensures consistent and safe access to resources. Prevents logical memory handling errors seen in low-level systems.

***Summary of Controls Implemented***

| Vulnerability | Remediation Action |
| --- | --- |
| Access Control | Enforced RBAC |
| Plaintext Passwords | Used bcrypt hashing |
| SQL Injection | Parameterized queries |
| Command Injection | Input validation, no shell calls |
| Cloud Misconfig | Secrets via env vars |
| Deserialization | JSON only, no pickle |
| Use-After-Free | Safe object state checks |