
✅ Security Vulnerabilities Identified & Resolved in SafeVault

1. 🛑 SQL Injection

- **Vulnerability:** Raw SQL queries using string concatenation were susceptible to injection attacks.
- **Fix Applied:** All SQL queries were rewritten using **parameterized queries** with Dapper, ensuring safe handling of user input.
- **Copilot's Role:** Suggested safe query syntax and generated unit tests to simulate SQL injection attempts (e.g., ' OR '1'='1), all of which correctly failed.

---

2. 🕸️ Cross-Site Scripting (XSS)

- **Vulnerability:** HTML content in form handling risked rendering unsanitized user input (innerHTML used in JS).
- **Fix Applied:** User inputs are now sanitized by using textContent instead of innerHTML when rendering into the DOM.
- **Copilot's Role:** Identified dangerous use of innerHTML, proposed secure alternatives, and generated automated tests to simulate <script> injections.

---

3. 🔒 Missing Role-Based Access Control

- **Vulnerability:** No restriction on which users could access admin or sensitive routes.
- **Fix Applied:** Implemented **RBAC** using [Authorize(Roles = "...")], with roles like Admin, User, and Guest, and protected API routes accordingly.
- **Copilot's Role:** Generated role-based authorization policies, controller decorators, and integration tests verifying access control behavior.

---

4. 🔐 Weak Authentication

- **Vulnerability:** No password hashing or secure credential handling.
- **Fix Applied:** Introduced secure password hashing using **BCrypt**, with verification during login.
- **Copilot's Role:** Provided implementation for hashing and verifying passwords, and test cases for login edge cases.

✅ **How Copilot Helped**

- 🕵️ Flagged risky patterns (SQL injection, XSS points)
- 🔧 Generated robust, secure code for database access, authentication, and role checks
- 🧪 Created automated test cases to simulate real-world attacks and confirm defenses work

---