**Assignment 4: Software Testing Document (STD) with Security by Design (SBD)**

|  |  |  |
| --- | --- | --- |
| **Project Name:** | **Team ID:** | **Mentor(s):** |

**Objective:**

Test your project for **at least one security vulnerability** by completing **Table 1** (Vulnerabilities and Mitigation Summary) and document detailed test cases and results in **Table 2 (Testing Log)**. Perform **at least one automated unit test and two manual tests (Exploratory/User level)** to confirm the vulnerability and its mitigation.. Ensure your team reviews the work, and **report any use of AI tools in Table 3**.

**Instructions:**

1. **Test Authentication (if applicable):** Ensure that your login system correctly rejects invalid usernames/passwords. Use **unit tests** to verify that only correct credentials grant access.
2. **Identify Vulnerabilities:** Select at least one security vulnerability relevant to your project from the categories in Table 1.
3. **Mitigation Plan:** For each vulnerability, **provide a brief solution** describing how you will fix the issue.
   1. **Example Mitigation Plans:**
      * **Weak password**: Use strong passwords and password hashing (like bcrypt).
      * **Sensitive data storage**: Encrypt sensitive information (like passwords) before saving.
      * **No input validation:** Make sure inputs are checked before being used in your system.
      * **Buffer Overflow:** Validate all input lengths and use safe functions that limit buffer sizes to avoid overflows.
      * **Insecure API Endpoints:** Implement authentication and authorization checks on all API endpoints; use tokens or OAuth for access control.
4. **Manual Testing:** Perform **at least two manual tests** to confirm that the vulnerability are addressed. **For instance:**
   * + Test that invalid login attempts are rejected.
     + Check that inputs are validated to prevent issues like SQL injection.
     + Test any security measures (e.g., encryption) to ensure they function correctly.
5. **Testing Your Project:** Focus on simple **unit tests** to check that authentication works and that functions run as expected.
   1. **Optional:** If your project includes **a front-end, you may choose** to perform front-end testing (e.g., checking if user interactions are properly handled), **but this is not required**.
6. **Team Review and AI Tools Usage:** Before submitting, make sure your team reviews the document to ensure everything is clear and correct. State whether AI tools were used; if yes, specify how.
7. **Grading and Resit:**
   1. This is a **pass/fail assignment**. If you fail the assignment, you may only **resubmit during the resit period**.
   2. You need a score of **7/10 or higher** to pass.
   3. Follow the **rubric carefully** to ensure your submission is complete and meets the required standards.
   4. If you **fail both attempts**, it will result **in failing the entire project**.

**Table 1: Vulnerabilities and Mitigation Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Points (Manual and Automated Testing)** | **Vulnerabilities Identified (name them)** | **Mitigation Plan** | **Tick ✔ (if completed all points)** | **Remarks (if any)** |
| 1. **Example- Unit Test 1:** Access Control & Authentication Issues | **Example- Authentication Flaw:** System accepts invalid username/password or lacks authentication | Use a simple unit test to check login functionality (correct/incorrect username & password) | ✔ | Example: Tested login or restricted access with invalid credentials; access denied as expected. If no login, mark N/A. |
| 1. **Example - Unit Test 2:** Weak Security in Functions (e.g., weak password hashing) | **Example- Weak Password Hashing:** Using MD5 for passwords or no encryption for sensitive data | Replace weak encryption/hashing with strong algorithms like bcrypt or AES | **✔** | Example: Updated password hashing from MD5 to bcrypt or encrypted sensor data before storage. |
| 1. **Example - Manual Security Test 1:** Access Control Check | **Example-** Unauthorized access to restricted features | Manually test restricted areas or commands to confirm only authorized access is allowed. | **✔** | Example: Tried accessing admin menu without proper auth; access blocked. |
| 1. **Example – Manual Security Test 2:** Input Validation Test | **Example-** Improper input handling **(e.g., no validation or sanitization)** | Validate and sanitize all inputs (user, sensor, config) | **✔** | Example: Entered empty/special chars; system handled safely without error. |

**Table 2: Testing Log (Unit and Manual Tests):**

* 1. **Test Strategy** - Unit Test or Manual Testing as applicable.
  2. **Date** - When did you perform the testing?
  3. **Process/Function** - What part of your system you are testing?
  4. **Test Case Step** - Specific step or scenario you tested.
  5. **Description** - What you did and why.
  6. **Status (Passed/Failed/Open)** - Whether the test passed, failed, or is still open.
  7. **Expected Results** - What should happen if the system works correctly.
  8. **Actual Result** - What actually happened during your test.
  9. **Mitigation Plan/Solutions** - How you fixed or plan to fix issues.
  10. **Review on Mitigation Plan (Passed/Failed)** - Team review pass/fail on mitigation.
  11. **Remarks on the Failed Mitigation Plan** - Any additional notes or explanations.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Strategy** | **Date** | **Process/Function** | **Test Case Step** | **Description** | **Status (Passed/Failed/Open)** | **Expected Results** | **Actual Result** | **Mitigation Plan/Solutions** | **Review on Mitigation Plan (Passed/Failed)** | **Remarks on the Failed Mitigation Plan** |
| Unit Test | [Enter Date] | Authentication/Login | Step 1: Test user login with valid credentials | Verify user can log in successfully | Passed/Failed/Open | User is logged in successfully | [Your result] | Correct login validation logic implemented | Passed / Failed | - |
| Unit Test | [Enter Date] | Authentication/ Login | Step 2: Test user login with invalid credentials | Verify invalid login attempts are rejected | Passed/Failed/Open | Login is denied with invalid input | [Your result] | Proper error handling for invalid credentials | Passed / Failed | - |
| Manual Testing | [Enter Date] | Access Control (Restricted features) | Step 1: Attempt unauthorized access | Try to access restricted features without valid input | Passed/Failed/Open | Access denied to unauthorized users | [Your result] | Implement access control checks | Passed / Failed | - |
| Manual Testing | [Enter Date] | Input Validation (User input from any interface) | Step 1: Enter invalid inputs (e.g., empty, special chars) | Ensure system safely handles unexpected or malicious inputs | Passed/Failed/Open | System safely rejects or sanitizes invalid inputs | [Your result] | Input validation and sanitization implemented | Passed / Failed | - |

**Table 3: Usage of AI Tools and Team Review:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Team Member Name** | **Reviewed Document (✔)** | **Date Reviewed** | **AI Tools Used (Yes/No)** | **If Yes, Specify How Used** |
| Member 1 | ✔ | [Date] | Yes / No | **Example:** Used ChatGPT to refine draft wording, clarify concepts. No direct copying of code or text blocks. |
| Member 2 | ✔ | [Date] | Yes / No | **UNETHICAL Example:** Using AI to write the entire project or assignment with minimal personal input or learning. |
| Member 3 | ✔ | [Date] | Yes / No | **Example:** GitHub Copilot was used to suggest code snippets during development. We carefully reviewed and tested all code before including it in the project. |
| Member 4 | ✔ | [Date] | Yes / No |  |
| Member 5 | ✔ | [Date] | Yes / No | **Example:** AI tools were used to brainstorm ideas and outline the testing strategy. We ensured full understanding and independently implemented the code and tests. |
| Member 6 | ✔ | [Date] | Yes / No |  |

**Tool Usage Guidelines:**

The following tools are **suggested** to help students perform **automated unit testing and manual testing**, but students are free to use any other appropriate tools that suit their project’s language, platform, and complexity. Additionally, **a list of optional tools** is provided to support teams who wish to deepen their understanding of software testing and security analysis. These optional tools are **not required for this assignment**, but may enhance the project by introducing practices like static code analysis, CI/CD integration, and dynamic application testing.

**Unit Testing (Functional Testing):**

|  |  |
| --- | --- |
| **Tool** | **Link** |
| **Python – unittest or pytest** | [unittest](https://docs.python.org/3/library/unittest.html), [pytest](https://docs.pytest.org/) |
| **Java – JUnit** | [JUnit Docs](https://junit.org/junit5/docs/current/user-guide/) |
| **JavaScript – Jest, Mocha** | [Jest](https://jestjs.io/), [Mocha](https://mochajs.org/) |

**Manual Security Testing (Focused on Security-Related Testing):**

|  |  |
| --- | --- |
| **Tool** | **Link** |
| **OWASP ZAP** | [OWASP ZAP](https://www.zaproxy.org/) |
| **Burp Suite (Community Edition)** | [Burp Suite](https://portswigger.net/burp/communitydownload) |
| **Postman** | [Postman](https://www.postman.com/) |

**Optional Tools (For Advanced Learning or Enhancements):**

|  |  |
| --- | --- |
| **Tool** | **Link** |
| **SonarQube** | [SonarQube Docs](https://docs.sonarsource.com/sonarqube-server/latest/) |
| **GitLab (SAST)** | [GitLab SAST](https://docs.gitlab.com/ee/user/application_security/sast/) |
| **GitLab (DAST)** | [GitLab DAST](https://docs.gitlab.com/ee/user/application_security/dast/) |
| **SonarQube + GitLab Integration** | [SonarQube-GitLab Integration](https://docs.sonarsource.com/sonarqube-server/10.8/devops-platform-integration/gitlab-integration/introduction/) |