Final Project: Phase III

Build it & Break it

Security Requirements Assignment

Fall 2024

Due date 11/20

Objective:

The objective of this assignment is to implement secure coding practices as you develop the core functionality of your project. In this phase, you will focus on writing clean, efficient, and secure code to ensure that your application is free from vulnerabilities and adheres to the security requirements defined in earlier phases.

Instructions:

1. **Implement Secure Coding Practices (Build it):**
    - You must follow secure coding principles throughout the implementation of your project. Be sure to:
        - **Input Validation:**
            - Validate and sanitize all user inputs to prevent injection attacks (e.g., SQL injection, command injection).
        - **Output Encoding:**
            - Ensure that output is properly encoded, especially when rendering data on interfaces, to prevent cross-site scripting (XSS) in web-based components.
        - **Use Parameterized Queries:**
            - Avoid constructing SQL queries dynamically using user input. Use prepared statements or parameterized queries to prevent SQL injection.
        - **Error Handling and Logging:**
            - Implement proper error handling to avoid leaking sensitive system details. Log security-related events without exposing sensitive data in error messages.
        - **Memory Management (C/C++ specific):**
            - Ensure proper handling of memory allocation and deallocation to prevent buffer overflows and memory leaks.
            - Use secure functions like `strncpy()` and avoid unsafe functions such as `strcpy()`.
        - **Avoid Hardcoding Sensitive Information:**

        - Never hardcode sensitive data such as passwords, API keys, or encryption keys directly into the source code. Use environment variables or secure storage mechanisms.
    - **Use Cryptography Appropriately:**
        - When implementing encryption, ensure you are using up-to-date and secure cryptographic algorithms (e.g., AES, RSA). Avoid using outdated or vulnerable algorithms such as MD5 or DES.
    - **Secure Authentication and Session Management:**
        - For applications requiring authentication, use secure password storage techniques (e.g., bcrypt, PBKDF2) and manage sessions securely (e.g., by using tokens with expiration).
2. **Follow the Security Requirements:**
    - As you implement the functionality, ensure that your code adheres to the security requirements defined during the requirements phase. Address both:
        - **Functional Requirements** (e.g., authentication, access control, input validation).
        - **Non-Functional Requirements** (e.g., performance under attack scenarios, usability of security features, compliance with legal standards).
3. **Incorporate Defensive Programming:**
    - Implement defensive programming techniques to anticipate and handle potential errors or attacks:
        - **Boundary Checking:** Ensure all array accesses are within bounds.
        - **Null Pointer Checks:** Avoid null pointer dereferences.
        - **Exception Handling:** Use try/catch blocks to gracefully handle unexpected errors.
        - **Assertions:** Use assertions to enforce security assumptions in critical sections of the code.
4. **Code Review (Break it):**
    - Conduct a peer code review (each group review Group i+1if you are group 1 review group 2. If you are group 10 review group 1) to identify potential security flaws before submission. Use static analysis tools to detect issues such as:
        - Buffer overflows
        - Race conditions
        - Memory leaks
        - Insecure API usage
5. **Document Security Features in Code:**
    - Comment your code to explain how you implemented security features. For example:
        - Explain input validation logic.
        - Detail encryption mechanisms and why they were chosen.
        - Document critical security assumptions or external dependencies (e.g., cryptographic libraries).
6. **Unit Testing for Security:**
    - Implement unit tests for your application, focusing on testing for security-related functionality:

        - Test for boundary cases, including inputs that might cause buffer overflows.
        - Test authentication and authorization mechanisms.
        - Write tests to validate error handling under potential attack conditions (e.g., malformed inputs).
7. **Secure Configuration Management:**
    - Ensure that your application's configuration files are secure:
        - **Sensitive Data**: Do not store sensitive information (e.g., credentials, API keys) in version control.
        - **Environment-Specific Settings**: Use environment variables for different development, testing, and production environments.
        - **Default Settings**: Disable default accounts or passwords before releasing the application.
8. **Deliverables:**
    - **Source Code:** Submit your source code, following your project's repository structure, ensuring the use of proper version control (e.g., Git).
    - **Code Documentation:** Provide inline comments and an additional readme file that explains the secure coding practices used, including any external libraries or tools used for security purposes.
    - **Test Cases:** Include a file with all security-related test cases, as well as instructions on how to run them.
    - **Code Review Report:** submit a brief report detailing what was reviewed, any issues found, and how they were addressed.
9. **Submission:**
    - Submit your source code, documentation, and test cases on Canvas by **11/20**.

Evaluation Criteria:

Your coding phase assignment will be evaluated based on:

- **Adherence to Secure Coding Practices:** Proper implementation of input validation, memory management, error handling, etc.
- **Code Quality:** Well-structured, readable, and maintainable code.
- **Security Feature Implementation:** Correct implementation of the security requirements specified earlier.
- **Test Coverage:** Adequate testing of security features and vulnerability edge cases.
- **Documentation:** Clear and thorough documentation of code and security features.

Additional Resources:

- OWASP Secure Coding Practices
- MITRE CWE (Common Weakness Enumeration)
- CERT Secure Coding Guidelines