UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

TOPIC:

Secure Coding Principles Specification

STUDENT:

Maciel Leyva Ariana Lizeth

GROUP:

10A

SUBJECT:

Comprehensive Mobile Development

TEACHER:

Ray Brunett Parra Galaviz

Tijuana, Baja California, January 21th of 2025

“Secure Coding Principles Specification”

What is Secure Coding?

Secure coding standards govern the coding practices, techniques, and decisions that developers make while building software. They aim to ensure that developers write code that minimizes security vulnerabilities. Development tasks can be solved in many different ways, with varying levels of complexity. Some solutions are more secure than others, and secure coding standards encourage developers and software engineers to take the safest approach, even if it is not the fastest.

For example, secure coding best practices often mandate a “default deny” approach to access permissions. Developers using secure coding techniques create code that denies access to sensitive resources unless an individual can demonstrate that they are authorized to access it.

Why is secure coding important?

Secure coding embeds security into your software’s DNA to prevent vulnerabilities like SQL injection, buffer overflows, cross-site scripting, and more. Beyond just preventing breaches, it’s a way to safeguard user trust, shift security left, and meet the tough standards of data protection laws.

The payoff? Fewer nasty surprises after launch, stronger apps, and better protection for both users and your organization.

The Best Secure Coding Principles and Practices

Validating user input, output encoding to keep data secure, authentication and authorization for verifying user access, responding to errors securely with error handling, protecting data in transit through secure communication practices, and learning from community resources are crucial aspects of secure coding principles and best practices.

Input Validation: Never Trust User Input

A core principle of secure coding is input validation, which emphasizes the importance of never trusting user input. This precautionary measure stems from the fact that users can intentionally or inadvertently introduce security vulnerabilities through their inputs. To counter such risks, it's crucial to utilize strict constraints on what type and format of data should be accepted by the system, as well as apply techniques like whitelisting valid characters and pattern matching for specific categories (e.g., alphabetical-only entries for names).

Output Encoding: Keep Data Secure

One of the critical aspects of secure coding is ensuring that sensitive data remains protected while being transmitted within a system. Output encoding, also known as output escaping, involves converting special characters into their equivalent HTML entities to keep data secure. This process prevents attackers from injecting malicious code into web pages and compromising user information. Output encoding is an essential practice for preventing cross-site scripting (XSS) attacks, which remain one of the most common security vulnerabilities today.

Authentication And Authorization: Verify User Access

Authentication and authorization are vital secure coding practices that ensure only authorized users can access certain features or information in an application. Authentication involves verifying the identity of a user, while authorization determines what actions and resources a user is allowed to access based on their role or permissions.

One common example of implementing authentication and authorization is using OAuth protocols for third-party logins, such as Facebook or Google. This process grants temporary access tokens to applications rather than passing login credentials directly from users to the application's servers. Embedding these principles into

software development life cycle (SDLC) ensures that security vulnerabilities are identified early enough before deployment.

Error Handling: Respond to Errors Securely

Error handling is an essential part of secure coding. When errors occur in software, they can provide attackers with valuable information that they can use to exploit vulnerabilities in the system. To avoid this, developers should handle errors securely by providing minimal error information and never revealing sensitive data or system details.

Another important aspect of error handling is logging and monitoring. By tracking errors and their root causes, developers can identify potential security threats and respond quickly if an attack occurs. It's also important to test error scenarios thoroughly during development to ensure that the application handles them appropriately and doesn't expose any vulnerabilities.

Secure Communication: Protect Data in Transit

When it comes to secure coding practices, protecting data in transit is an essential principle that developers need to follow. It involves ensuring that the communication between different systems, servers, or devices is secure and encrypted. One way to implement this practice is to use secure communication libraries such as OpenSSL or Bouncy Castle. These libraries provide APIs for encryption and decryption of data transmitted across networks.

Another effective approach is implementing two-factor authentication which adds an extra layer of security when communicating sensitive information over the internet.

What are the guidelines for personal data protection in mobile applications?

- **Data Minimization:** Collect only the personal data necessary for the application's functionality. Avoid requesting excessive permissions.

- **Transparency:** Clearly inform users about what data is being collected, how it is stored, and for what purpose. Implement user-friendly privacy policies.
- **Consent Management:** Obtain explicit consent from users before collecting sensitive data. Allow users to revoke permissions at any time.
- **Secure Storage:** Use encrypted storage mechanisms for sensitive data, such as Keychain in iOS or Encrypted SharedPreferences in Android.
- **Data Retention:** Retain personal data only as long as necessary. Implement automated deletion policies for unused or outdated data.
- **Compliance:** Adhere to legal frameworks like GDPR, CCPA, or local privacy laws to protect personal information.

What are the principles of secure coding in mobile application development?

- **Input Validation:** Ensure all inputs are sanitized to prevent injection attacks (e.g., SQL injection, cross-site scripting).
- **Authentication and Authorization:** Use secure protocols like OAuth2 for user authentication and role-based access controls for authorization. Avoid hardcoding sensitive credentials in the app.
- **Secure Communication:** Enforce HTTPS using TLS (Transport Layer Security) for all data exchanges. Reject insecure SSL/TLS versions.
- **Code Obfuscation:** Minimize the risk of reverse engineering by obfuscating the application code. Tools like ProGuard (Android) or LLVM Obfuscator (iOS) can help.
- **Session Management:** Implement secure session mechanisms with expiration policies, and avoid storing session tokens in insecure locations.
- **Error Handling:** Avoid exposing sensitive information in error messages or logs that attackers could exploit.
- **Regular Updates:** Address security patches promptly to keep up with evolving threats.

Identifying points of vulnerability in mobile applications

- **Insecure Data Storage:** Storing sensitive data (e.g., user credentials, tokens) in plaintext or unprotected locations.
- **Weak Authentication:** Using insecure or outdated authentication methods, such as passwords stored without hashing.
- **Insufficient Transport Layer Protection:** Failing to enforce HTTPS for network communications, leaving data susceptible to interception.
- **Unvalidated Inputs:** Allowing user input without proper validation can lead to injection attacks.
- **Unsecured APIs:** Exposing APIs without adequate security measures like rate limiting, authentication, or validation.
- **Third-Party Libraries:** Using outdated or vulnerable libraries can create attack vectors for malicious actors.
- **Reverse Engineering Risks:** Lack of code obfuscation allows attackers to analyze and exploit the app's logic.

Identifying data encryption mechanisms in information exchange

- **Symmetric Encryption:** Uses the same key for encryption and decryption (e.g., AES - Advanced Encryption Standard). Suitable for fast data encryption in secured environments.
- **Asymmetric Encryption:** Utilizes a pair of public and private keys for encryption and decryption (e.g., RSA). Commonly used in secure key exchanges.
- **TLS/SSL Protocols:** Ensures secure data exchange over networks. Enforces encryption, authentication, and data integrity during transport.
- **End-to-End Encryption (E2EE):** Encrypts data on the sender's side and decrypts it only on the receiver's side, ensuring that intermediaries cannot access the content (e.g., WhatsApp encryption).
- **Hashing:** Converts sensitive information like passwords into fixed-length values using algorithms such as SHA-256. It is irreversible and ensures data integrity.
- **Secure Key Management:** Use Hardware Security Modules (HSMs), Keychain (iOS), or Keystore (Android) to securely manage and store encryption keys.

Sources

Securecoding.org. (2023, June 1st). *Secure Coding Principles and Best Practices to Learn*. Retrieved January 21, 2025, of https://www.securecoding.org/secure-coding-principles-best-practices/#:~:text=By%20following%20principles%20like%20input,or%20cross%2Dsite%20scripting%20attacks.

Thurmond, T. (2023, December 27th). *8 Secure Coding Practices Learned from OWASP | KirkpatrickPrice*. KirkpatrickPrice. Retrieved January 21, 2025, of https://kirkpatrickprice.com/blog/secure-coding-best-practices/

Belov, M. (2024, November 2nd). *Secure coding principles*. DEV Community. Retrieved January 21, 2025, of https://dev.to/owasp/secure-coding-principles-4dg8