Microsoft Security Development Lifecycle

EC-Council

Shifting Left

- **What It Is:** "Shifting left" means moving security concerns earlier in the software development lifecycle (SDLC). Traditionally, security was often tacked on at the end. Shifting left integrates security into planning, design, coding, and testing phases.
- **Benefits:**
    - **Early Detection:** Uncovers vulnerabilities sooner when they are cheaper and easier to fix.
    - **Reduced Risk:** Mitigates the chance of security flaws making it into production.
    - **Cost Savings:** Prevents expensive remediation efforts later in the project.
    - **Improved Security Culture:** Makes security everyone's responsibility.
- **How It's Done:**
    - **Threat Modeling:** Identifying potential threats early in design.
    - **Static Code Analysis:** Scanning code for vulnerabilities during development.
    - **Security Unit Testing:** Writing tests specifically to uncover security weaknesses.
    - **Security Training:** Educating developers on secure coding practices.

Secure by Design

- **What It Is:** This is a proactive approach to security where security is a fundamental aspect of software design. It involves making deliberate choices to minimize attack surfaces and build in resilience.
- **Principles:**
    - **Least Privilege:** Give users and components the minimum permissions needed.
    - **Defense in Depth:** Multiple layers of security to protect against various attack vectors.

    - **Fail Securely:** Errors should default to a safe state, not an insecure one.
    - **Secure Defaults:** Start with secure configurations out of the box.
    - **Simplicity:** Complex systems are harder to secure, so aim for simplicity.

Least Privilege

- **What It Is:** Granting users, processes, or systems only the permissions they absolutely require to perform their functions.
- **Why It Matters:**
    - **Limit Damage:** If a user or system is compromised, the attacker's access is restricted.
    - **Reduced Attack Surface:** Fewer privileges mean fewer ways to exploit vulnerabilities.
- **How to Implement:**
    - **Role-Based Access Control (RBAC):** Assign permissions based on job roles.
    - **Principle of Least Authority (POLA):** Give only the minimum authority necessary for a specific task.
    - **Time-Based Privileges:** Grant elevated privileges only when needed for a short duration.

Secure Coding Practices

- **What It Is:** A set of guidelines and techniques for writing code that is resistant to vulnerabilities.
- **Key Practices:**
    - **Input Validation:** Always validate and sanitize user input.
    - **Output Encoding:** Encode output to prevent cross-site scripting (XSS) attacks.
    - **Error Handling:** Handle errors gracefully to avoid revealing sensitive information.
    - **Authentication and Authorization:** Strong authentication and authorization mechanisms.
    - **Cryptography:** Use strong cryptographic algorithms and practices.

Input Validation

- What It Is: The process of checking user input to ensure it meets expected criteria and doesn't contain malicious content.
- Why It's Critical: The primary defense against injection attacks (SQL injection, command injection, etc.).
- How to Do It:
    - Whitelisting: Define allowed input formats and reject anything that doesn't match.
    - Blacklisting: (Less preferred) Look for known bad patterns, but be aware this can be bypassed.
    - Escaping: Transform special characters so they are treated as literal text.

Regular Testing and Updates

- Importance:
    - Uncover Vulnerabilities: Testing finds security holes before attackers do.
    - Patching: Regularly update software to fix known vulnerabilities.
    - Regression Testing: Ensure updates don't introduce new problems.

Secure Coding Tools and Libraries

- Examples:
    - Static Analysis Tools: Find vulnerabilities in code without executing it (e.g., SonarQube, Coverity).
    - Dynamic Analysis Tools: Analyze running code to identify issues (e.g., Burp Suite, OWASP ZAP).
    - Software Composition Analysis (SCA): Track and manage open-source components for vulnerabilities.
    - Secure Libraries: Use libraries designed with security in mind (e.g., cryptography libraries).