Insecure Design

Part 1: Beginner Level

What is Insecure Design?

Insecure design refers to flaws in application security architecture that exist before a line of code is written. Unlike implementation bugs, insecure design represents fundamental errors in how security controls are conceived and planned. These vulnerabilities arise when security requirements aren't properly identified during the planning phase, resulting in systems that are inherently vulnerable regardless of how well they're implemented.

Key Concepts

1. **Design vs. Implementation:**
    - **Design flaws: Missing or ineffective security controls in the system's architecture**
    - **Implementation flaws: Coding errors in existing security controls**
2. **Security by Design:**
    - **Proactive approach to building security into products from conception**
    - **Focuses on preventing security issues rather than reacting to them**
3. **Threat Modeling:**
    - **Structured approach to identifying potential threats**
    - **Helps determine what could go wrong in a system before it's built**

Common Insecure Design Issues

1. Missing Authentication

Example: A banking application allowing account balance checking without requiring login

Problem: Sensitive information is accessible without identity verification

2. Insufficient Authorization

Example: All logged-in users can access administrative functions

Problem: Access is not restricted based on user roles/permissions

3. Excessive Trust in Client-Side Controls

Example: Relying on JavaScript validation for business logic enforcement

Problem: Client-side controls can be bypassed with browser tools or API requests

4. Business Logic Flaws

Example: Allowing negative quantities in shopping carts to reduce total price

Problem: System doesn't account for logical constraints in business processes

5. Lack of Rate Limiting

Example: No limits on login attempts or API calls

Problem: Enables brute force attacks and resource exhaustion

Real-World Examples

1. **Password Reset Flaws:**
    - **Sending reset links without verifying identity**
    - **Using predictable tokens or IDs**
    - **Not invalidating tokens after use**
2. **Multi-step Process Vulnerabilities:**
    - **Skipping steps in checkout processes**
    - **Manipulating order information between steps**
    - **Tampering with prices or quantities**
3. **Insufficient Access Controls:**
    - **Horizontal privilege escalation (accessing other users' data)**
    - **Vertical privilege escalation (accessing higher-level functions)**
    - **Missing function-level authorization checks**

Basic Prevention Techniques

1. **Secure Development Lifecycle (SDL):**
    - **Incorporate security requirements from the start**
    - **Regular security reviews throughout development**
    - **Security testing before release**
2. **Simple Threat Modeling:**
    - **Identify assets to protect**
    - **List potential threats**
    - **Develop mitigations for each threat**
3. **Security Requirements:**
    - **Document specific security needs before development**
    - **Include security acceptance criteria in user stories**
    - **Consider security implications of all features**
4. **Defense in Depth:**
    - **Implement multiple layers of protection**
    - **Don't rely on single security controls**

- **Assume outer defenses may be breached**

5. **Least Privilege:**
    - **Grant minimal access rights necessary for functions**
    - **Default to secure/restricted state**
    - **Always require explicit authorization for sensitive actions**

Key Tools and Resources

1. **Threat Modeling Tools:**
    - **Microsoft Threat Modeling Tool**
    - **OWASP Threat Dragon**
    - **Simple diagrams and checklists**
2. **Security Requirements Framework:**
    - **OWASP Application Security Verification Standard (ASVS)**
    - **NIST Cybersecurity Framework**
    - **Security User Stories**