

## **Intermediate Level**

### **Advanced Design Vulnerability Patterns**

#### **Data Flow Vulnerabilities**

##### **1. Trust Boundary Violations:**

- Design fails to validate data when crossing trust boundaries
- Example: Using user-provided data from a database without re-validation
- Prevention: Validate data at each trust boundary crossing

##### **2. Missing Cryptographic Controls:**

- Inadequate encryption for data at rest or in transit
- Example: Storing passwords in recoverable format
- Prevention: Identify sensitive data and apply appropriate encryption

##### **3. Insecure Data Handling:**

**Problem:** System designed to log sensitive information

**Vulnerability:** Logs containing PII/PHI/PCI data

**Mitigation:** Design comprehensive data classification and handling policies

#### **Authentication Design Flaws**

##### **1. Weak Authentication Schemes:**

**Problem:** Single-factor authentication for sensitive systems

**Vulnerability:** Compromise of one factor leads to complete account takeover

**Mitigation:** Implement MFA, especially for privileged access

##### **2. Centralized Authentication Failures:**

**Problem:** Single point of failure in authentication systems

**Vulnerability:** If the central authentication system is compromised, all connected systems are affected

**Mitigation:** Defense-in-depth with multiple validation points

##### **3. Session Management Design Flaws:**

**Problem:** Poor session lifecycle management

**Vulnerability:** Sessions that don't expire or have overly long timeouts

**Mitigation:** Design proper session creation, validation, and termination flows

#### **Authorization Matrix Failures**

##### **1. Role-Based Access Control (RBAC) Misconfigurations:**

**Problem: Overly permissive role definitions**

**Vulnerability: Users get more permissions than necessary**

**Mitigation: Design granular roles with least privilege**

## **2. Missing Contextual Authorization:**

**Problem: Authorization checks based solely on role, not context**

**Vulnerability: Users can access data or functions outside their context**

**Mitigation: Include contextual factors in authorization decisions**

**Example authorization matrix:**

**Feature | Anonymous | User | Manager | Admin**

-----

**View public |   ✓  |   ✓  |   ✓  |   ✓**

**View own |   X  |   ✓  |   ✓  |   ✓**

**View others |   X  |   X  |   ✓  |   ✓**

**Modify own |   X  |   ✓  |   ✓  |   ✓**

**Modify any |   X  |   X  |   X  |   ✓**

## **Systemic Design Issues**

### **1. Inadequate Error Handling and Logging**

**Problem: System not designed to handle unexpected inputs or states gracefully**

**Vulnerability: Error leakage, insufficient forensic information**

**Mitigation: Design comprehensive error handling strategy**

**Example of proper error handling design:**

- 1. Log detailed error information (internal)**
- 2. Generate unique error reference**
- 3. Return generic error message with reference to user**
- 4. Maintain centralized error monitoring**

### **2. Race Conditions by Design**

**Problem: Business processes with time-of-check to time-of-use gaps**

**Vulnerability: State manipulation between steps**

**Mitigation: Design atomic operations or implement proper controls**

**Example:**

**Vulnerable process:**

1. Check if user has funds
2. Reserve item
3. Process payment

Fixed design:

1. Lock user funds
2. Reserve item
3. Process payment
4. Release lock (success or compensating transaction)

### 3. Insecure Defaults

Problem: Systems designed with convenience over security

Vulnerability: Default open access, permissive settings

Mitigation: Design secure defaults with opt-in for less secure options

Secure Design Methodologies

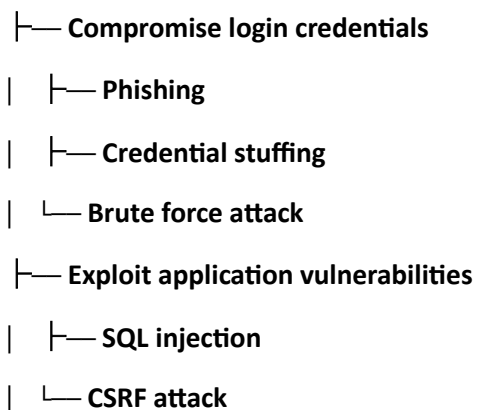
Threat Modeling Techniques

#### 1. STRIDE Analysis:

- Spoofing: Pretending to be someone else
- Tampering: Modifying data or code
- Repudiation: Denying actions
- Information Disclosure: Exposing information
- Denial of Service: Making system unavailable
- Elevation of Privilege: Gaining higher access

#### 2. Attack Trees:

Goal: Transfer money from victim account



└─ Manipulate transaction processing

└─ Race condition

└─ Transaction parameter tampering

### 3. Data Flow Diagram (DFD) Analysis:

- Map where data moves through the system
- Identify trust boundaries
- Analyze points where security controls are needed

## Secure Architecture Patterns

### 1. API Gateway Pattern:

#### Benefits:

- Centralized authentication and authorization
- Rate limiting and monitoring
- Input validation

#### Implementation:

- Place gateway in front of microservices
- Configure security policies per endpoint
- Monitor for abnormal patterns

### 2. Secure Microservices Architecture:

#### Design principles:

- Service-level authentication
- Defense in depth with layered services
- Least privilege communication
- Independent security controls

### 3. Zero Trust Architecture:

#### Core concepts:

- Never trust, always verify
- Assume breach
- Verify explicitly
- Least privilege access
- Continuous monitoring and validation

## **Testing for Design Flaws**

### **1. Architecture Reviews:**

- **Systematic examination of design documents**
- **Security control mapping**
- **Gap analysis against requirements**

### **2. Abuse Case Analysis:**

- **Create scenarios where actors attempt to misuse the system**
- **Test against security assumptions**
- **Identify missing controls**

### **3. Threat-Based Testing:**

- **Derive test cases from threat models**
- **Focus on business logic rather than just technical vulnerabilities**
- **Test security control boundaries**