A06:2025 — Insecure Design

January 6, 2026

Document Summary

This document consolidates the provided content for *A06:2025 — Insecure Design* into a structured, print-ready reference, including background context, scoring metrics, conceptual definition (design vs. implementation), the three key parts of secure design (requirements/resource management, secure design methodology, and secure development lifecycle), prevention guidance, attack scenarios, references, and the mapped CWE list.

Contents

1 Background

Insecure Design slides two spots from #4 to #6 in the ranking as A02:2025 — Security Misconfiguration and A03:2025 — Software Supply Chain Failures leapfrog it. This category was introduced in 2021, and there have been noticeable improvements in the industry related to threat modeling and a greater emphasis on secure design.

This category focuses on risks related to design and architectural flaws, with a call for greater use of threat modeling, secure design patterns, and reference architectures. It includes flaws in the business logic of an application, such as failing to define unwanted or unexpected state changes inside an application.

As a community, we need to move beyond "shift-left" in the coding space to **pre-code** activities such as requirements writing and application design that are critical for Secure by Design (e.g., see *Establish a Modern AppSec Program: Planning and Design Phase*).

Notable Common Weakness Enumerations (CWEs) include:

- CWE-256: Unprotected Storage of Credentials
- CWE-269: Improper Privilege Management
- CWE-434: Unrestricted Upload of File with Dangerous Type
- CWE-501: Trust Boundary Violation
- CWE-522: Insufficiently Protected Credentials

2 Score Table

Metric	Value
CWEs Mapped	39
Max Incidence Rate	22.18%
Avg Incidence Rate	1.86%
Max Coverage	88.76%
Avg Coverage	35.18%
Avg Weighted Exploit	6.96
Avg Weighted Impact	4.05
Total Occurrences	729,882
Total CVEs	7,647

Table 1: Provided scoring summary for Insecure Design.

3 Description

3.1 Definition and Scope

Insecure design is a broad category representing different weaknesses, expressed as *"missing or ineffective control design."* Insecure design is not the source for all other Top Ten risk categories.

Design vs. Implementation

There is a difference between **insecure design** and **insecure implementation**. They have different root causes, occur at different times in the development process, and require different remediations.

- A **secure design** can still have **implementation defects** leading to exploitable vulnerabilities.
- An **insecure design** cannot be fixed by a perfect implementation because the necessary security controls were never designed to defend against specific attacks.

One factor contributing to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

3.2 Three Key Parts of Having a Secure Design

Core Components of Secure Design

1. Gathering Requirements and Resource Management
2. Creating a Secure Design
3. Having a Secure Development Lifecycle

3.2.1 Requirements and Resource Management

Collect and negotiate the business requirements for an application with the business, including the protection requirements concerning confidentiality, integrity, availability, and authenticity of all data assets and the expected business logic. Take into account how exposed your application will be and whether you need segregation of tenants (beyond those needed for access control).

Compile technical requirements, including functional and non-functional security requirements. Plan and negotiate the budget covering all design, build, testing, and operation, including security activities.

3.2.2 Secure Design

Secure design is a culture and methodology that constantly evaluates threats and ensures that code is robustly designed and tested to prevent known attack methods. Threat modeling should be integrated into refinement sessions (or similar activities), looking for changes in data flows, access control, or other security controls.

During user story development, determine the correct flow and failure states, ensure they are well understood and agreed upon by responsible and impacted parties. Analyze assumptions and conditions for expected and failure flows to ensure they remain accurate and desirable. Determine how to validate assumptions and enforce conditions needed for proper behaviors. Ensure results are documented in the user story.

Learn from mistakes and offer positive incentives to promote improvements. Secure design is neither an add-on nor a tool that can simply be applied to software.

3.2.3 Secure Development Lifecycle

Secure software requires a secure development lifecycle, a secure design pattern, a paved road methodology, a secure component library, appropriate tooling, threat modeling, and incident post-mortems that are used to improve the process.

Engage security specialists at the beginning of a software project, throughout the project, and for ongoing software maintenance. Consider leveraging the OWASP Software Assurance Maturity Model (SAMM) to help structure secure software development efforts.

Security Culture Emphasis

Self-responsibility of developers is often underappreciated. Foster a culture of awareness, responsibility, and proactive risk mitigation. Regular security exchanges (e.g., during threat modeling sessions) can generate a mindset for including security in important design decisions.

4 How to Prevent

- Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls.
- Establish and use a library of secure design patterns or paved-road components.
- Use threat modeling for critical parts of the application such as authentication, access control, business logic, and key flows.
- Use threat modeling as an educational tool to generate a security mindset.
- Integrate security language and controls into user stories.
- Integrate plausibility checks at each tier of the application (from frontend to backend).
- Write unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases and misuse-cases for each tier of the application.
- Segregate tier layers on system and network layers depending on exposure and protection needs.
- Segregate tenants robustly by design throughout all tiers.

5 Example Attack Scenarios

5.1 Scenario #1: Weak Credential Recovery Workflow

A credential recovery workflow might include “questions and answers,” which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity, as more than one person can know the answers. Such functionality should be removed and replaced with a more secure design.

5.2 Scenario #2: Business Logic Abuse of Group Booking Discounts

A cinema chain allows group booking discounts and has a maximum of fifteen attendees before requiring a deposit. Attackers could threat model this flow and test whether an attack vector exists in the business logic, such as booking six hundred seats across all cinemas in a few requests, causing a massive loss of income.

5.3 Scenario #3: Missing Anti-bot Design for Scarce Goods

A retail chain's e-commerce website does not protect against bots run by scalpers buying high-end video cards to resell on auction websites. This creates negative publicity for the video card makers and retail chain owners, and enduring bad blood with enthusiasts who cannot obtain these cards at any price. Careful anti-bot design and domain logic rules (e.g., suspicious purchases made within seconds of availability) can help identify inauthentic purchases and reject transactions.

6 References

- OWASP Cheat Sheet: Secure Design Principles
- OWASP SAMM: Design | Secure Architecture
- OWASP SAMM: Design | Threat Assessment
- NIST — Guidelines on Minimum Standards for Developer Verification of Software
- The Threat Modeling Manifesto
- Awesome Threat Modeling

7 List of Mapped CWEs

CWE	Title
CWE-73	External Control of File Name or Path
CWE-183	Permissive List of Allowed Inputs
CWE-256	Unprotected Storage of Credentials
CWE-266	Incorrect Privilege Assignment
CWE-269	Improper Privilege Management
CWE-286	Incorrect User Management
CWE-311	Missing Encryption of Sensitive Data
CWE-312	Cleartext Storage of Sensitive Information
CWE-313	Cleartext Storage in a File or on Disk
CWE-316	Cleartext Storage of Sensitive Information in Memory
CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization (*Race Condition*)
CWE-382	J2EE Bad Practices: Use of `System.exit()`
CWE-419	Unprotected Primary Channel
CWE-434	Unrestricted Upload of File with Dangerous Type
CWE-436	Interpretation Conflict
CWE-444	Inconsistent Interpretation of HTTP Requests (*HTTP Request Smuggling*)
CWE-451	User Interface (UI) Misrepresentation of Critical Information
CWE-454	External Initialization of Trusted Variables or Data Stores
CWE-472	External Control of Assumed-Immutable Web Parameter
CWE-501	Trust Boundary Violation
CWE-522	Insufficiently Protected Credentials
CWE-525	Use of Web Browser Cache Containing Sensitive Information
CWE-539	Use of Persistent Cookies Containing Sensitive Information
CWE-598	Use of GET Request Method With Sensitive Query Strings
CWE-602	Client-Side Enforcement of Server-Side Security
CWE-628	Function Call with Incorrectly Specified Arguments
CWE-642	External Control of Critical State Data
CWE-646	Reliance on File Name or Extension of Externally-Supplied File
CWE-653	Insufficient Compartmentalization
CWE-656	Reliance on Security Through Obscurity
CWE-657	Violation of Secure Design Principles
CWE-676	Use of Potentially Dangerous Function
CWE-693	Protection Mechanism Failure
CWE-799	Improper Control of Interaction Frequency
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-841	Improper Enforcement of Behavioral Workflow
CWE-1021	Improper Restriction of Rendered UI Layers or Frames

CWE	**Title**
CWE-1022	Use of Web Link to Untrusted Target with `window.opener` Access
CWE-1125	Excessive Attack Surface