

A02:2025 — Security Misconfiguration

January 6, 2026

Document Summary

This document consolidates the provided content for *A02:2025 — Security Misconfiguration* into a structured, print-ready reference, including background context, scoring metrics, vulnerability indicators, prevention guidance, practical attack scenarios, references, and the mapped CWE list.

Contents

1	Background	2
2	Score Table	2
3	Description	2
3.1	Indicators of Misconfiguration	2
4	How to Prevent	3
5	Example Attack Scenarios	3
5.1	Scenario #1: Sample Applications and Default Credentials	3
5.2	Scenario #2: Directory Listing and Reverse Engineering	4
5.3	Scenario #3: Verbose Error Messages	4
5.4	Scenario #4: Insecure Cloud Storage Defaults	4
6	References	4
7	List of Mapped CWEs	5

1 Background

Moving up from #5 in the previous edition, 100% of the applications tested were found to have some form of misconfiguration, with an average incidence rate of 3.00%, and over 719k occurrences of a Common Weakness Enumeration (CWE) in this risk category. With more shifts into highly configurable software, it is not surprising to see this category moving up. Notable CWEs included are CWE-16 (Configuration) and CWE-611 (Improper Restriction of XML External Entity Reference, XXE).

2 Score Table

Metric	Value
CWEs Mapped	16
Max Incidence Rate	27.70%
Avg Incidence Rate	3.00%
Max Coverage	100.00%
Avg Coverage	52.35%
Avg Weighted Exploit	7.96
Avg Weighted Impact	3.97
Total Occurrences	719,084
Total CVEs	1,375

Table 1: Provided scoring summary for Security Misconfiguration.

3 Description

Security misconfiguration is when a system, application, or cloud service is set up incorrectly from a security perspective, creating vulnerabilities.

3.1 Indicators of Misconfiguration

The application might be vulnerable if:

- It is missing appropriate security hardening across any part of the application stack or has improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, testing frameworks, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- There is a lack of central configuration for intercepting excessive error messages; error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- Excessive prioritization of backward compatibility leads to insecure configuration.

- Security settings in application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- Without a concerted, repeatable application security configuration hardening process, systems are at a higher risk.

4 How to Prevent

Secure installation processes should be implemented, including:

1. **Repeatable hardening:** Enable fast and easy deployment of additional environments that are appropriately locked down. Development, QA, and production environments should be configured identically, with different credentials used in each environment. Automate this process to minimize the effort required to set up a new secure environment.
2. **Minimal platform footprint:** Use a minimal platform without unnecessary features, components, documentation, or samples. Remove or do not install unused features and frameworks.
3. **Patch and configuration governance:** Review and update configurations appropriate to all security notes, updates, and patches as part of patch management. Review cloud storage permissions (e.g., S3 bucket permissions).
4. **Segmentation and isolation:** Use a segmented application architecture to provide effective separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
5. **Client security directives:** Send security directives to clients (e.g., Security Headers).
6. **Automated verification:** Implement an automated process to verify the effectiveness of configurations and settings in all environments.
7. **Centralized error interception:** Proactively add a central configuration to intercept excessive error messages as a backup.
8. **Manual verification cadence:** If verifications are not automated, manually verify annually at a minimum.
9. **Prefer federated or ephemeral credentials:** Use identity federation, short-lived credentials, or role-based access mechanisms provided by the underlying platform instead of embedding static keys or secrets in code, configuration files, or pipelines.

5 Example Attack Scenarios

5.1 Scenario #1: Sample Applications and Default Credentials

The application server comes with sample applications not removed from the production server. These sample applications have known security flaws that attackers use to compromise the server. Suppose one of these applications is the admin console, and default accounts were not changed; in that case, the attacker logs in with the default password and takes over.

5.2 Scenario #2: Directory Listing and Reverse Engineering

Directory listing is not disabled on the server. An attacker discovers they can list directories, finds and downloads the compiled Java classes, and then decompiles and reverse engineers them to view the code. The attacker then identifies a severe access control flaw in the application.

5.3 Scenario #3: Verbose Error Messages

The application server's configuration allows detailed error messages, such as stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws, such as component versions that are known to be vulnerable.

5.4 Scenario #4: Insecure Cloud Storage Defaults

A cloud service provider (CSP) defaults to having sharing permissions open to the Internet. This allows sensitive data stored within cloud storage to be accessed.

6 References

- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- Application Security Verification Standard V13: Configuration
- NIST Guide to General Server Hardening
- CIS Security Configuration Guides/Benchmarks
- Amazon S3 Bucket Discovery and Enumeration
- ScienceDirect: Security Misconfiguration

7 List of Mapped CWEs

CWE	Title
CWE-5	J2EE Misconfiguration: Data Transmission Without Encryption
CWE-11	ASP.NET Misconfiguration: Creating Debug Binary
CWE-13	ASP.NET Misconfiguration: Password in Configuration File
CWE-15	External Control of System or Configuration Setting
CWE-16	Configuration
CWE-260	Password in Configuration File
CWE-315	Cleartext Storage of Sensitive Information in a Cookie
CWE-489	Active Debug Code
CWE-526	Exposure of Sensitive Information Through Environmental Variables
CWE-547	Use of Hard-coded, Security-relevant Constants
CWE-611	Improper Restriction of XML External Entity Reference
CWE-614	Sensitive Cookie in HTTPS Session Without <code>Secure</code> Attribute
CWE-776	Improper Restriction of Recursive Entity References in DTDs (<i>XML Entity Expansion</i>)
CWE-942	Permissive Cross-domain Policy with Untrusted Domains
CWE-1004	Sensitive Cookie Without <code>HttpOnly</code> Flag
CWE-1174	ASP.NET Misconfiguration: Improper Model Validation