A01:2025 — Broken Access Control

January 6, 2026

Document Summary

This document consolidates the provided content for *A01:2025 — Broken Access Control* into a structured, print-ready reference, including background context, scoring metrics, common vulnerability patterns, prevention guidance, practical attack scenarios, references, and the mapped CWE list.

Contents

1 Background

Maintaining its position at #1 in the Top Ten, 100% of the applications tested were found to have some form of broken access control. Notable CWEs included are CWE-200: Exposure of Sensitive Information to an Unauthorized Actor, CWE-201: Exposure of Sensitive Information Through Sent Data, CWE-918: Server-Side Request Forgery (SSRF), and CWE-352: Cross-Site Request Forgery (CSRF). This category has the highest number of occurrences in the contributed data, and the second highest number of related CVEs.

2 Score Table

Metric	Value
CWEs Mapped	40
Max Incidence Rate	20.15%
Avg Incidence Rate	3.74%
Max Coverage	100.00%
Avg Coverage	42.93%
Avg Weighted Exploit	7.04
Avg Weighted Impact	3.84
Total Occurrences	1,839,701
Total CVEs	32,654

Table 1: Provided scoring summary for Broken Access Control.

3 Description

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification or destruction of all data, or performing a business function outside the user's limits.

3.1 Common Access Control Vulnerability Patterns

- Violation of the principle of least privilege (deny by default), where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool that modifies API requests.
- Permitting viewing or editing someone else's account by providing its unique identifier (insecure direct object references).
- An accessible API with missing access controls for `POST`, `PUT`, and `DELETE`.
- Elevation of privilege: acting as a user without being logged in or gaining privileges beyond those expected of the logged-in user (e.g., admin access).

- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, a cookie, or hidden field manipulated to elevate privileges, or abusing JWT invalidation.
- CORS misconfiguration that allows API access from unauthorized or untrusted origins.
- Force browsing (guessing URLs) to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

4 How to Prevent

Access control is only effective when implemented in trusted server-side code or serverless APIs, where the attacker cannot modify the access control check or metadata.

1. Except for public resources, deny by default.
2. Implement access control mechanisms once and reuse them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
3. Model access controls should enforce record ownership rather than allowing users to create, read, update, or delete any record.
4. Unique application business limit requirements should be enforced by domain models.
5. Disable web server directory listing and ensure file metadata (e.g., `.git`) and backup files are not present within web roots.
6. Log access control failures; alert administrators when appropriate (e.g., repeated failures).
7. Implement rate limits on API and controller access to minimize the harm from automated attack tooling.
8. Stateful session identifiers should be invalidated on the server after logout. Stateless JWT tokens should be short-lived to minimize the window of opportunity for an attacker. For longer-lived JWTs, consider using refresh tokens and following OAuth standards to revoke access.
9. Use well-established toolkits or patterns that provide simple, declarative access controls.
10. Developers and QA staff should include functional access control in their unit and integration tests.

5 Example Attack Scenarios

5.1 Scenario #1: Parameter Tampering Against Account Lookups

The application uses unverified data in an SQL call that is accessing account information:

Example Code (Java)

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker can simply modify the browser's `acct` parameter to send any desired account number. If not correctly verified, the attacker can access any user's account.

Illustrative Request

```
https://example.com/app/accountInfo?acct=notmyacct
```

5.2 Scenario #2: Forced Browsing to Privileged Endpoints

Admin rights are required for access to the admin page:

Illustrative Requests

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

5.3 Scenario #3: Front-End Only Access Control

An application places all access control in its front-end. While the attacker cannot get to `https://example.com/app/admin_getappInfo` due to JavaScript code running in the browser, they can simply execute:

Illustrative Command

```
$ curl https://example.com/app/admin_getappInfo
```

6 References

- OWASP Proactive Controls: C1: Implement Access Control
- OWASP Application Security Verification Standard: V8 Authorization
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Authorization
- PortSwigger: Exploiting CORS misconfiguration
- OAuth: Revoking Access

7 List of Mapped CWEs

CWE	**Title**
CWE-22	Improper Limitation of a Pathname to a Restricted Directory (*Path Traversal*)
CWE-23	Relative Path Traversal
CWE-36	Absolute Path Traversal
CWE-59	Improper Link Resolution Before File Access (*Link Following*)
CWE-61	UNIX Symbolic Link (Symlink) Following
CWE-65	Windows Hard Link
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor
CWE-201	Exposure of Sensitive Information Through Sent Data
CWE-219	Storage of File with Sensitive Data Under Web Root
CWE-276	Incorrect Default Permissions
CWE-281	Improper Preservation of Permissions
CWE-282	Improper Ownership Management
CWE-283	Unverified Ownership
CWE-284	Improper Access Control
CWE-285	Improper Authorization
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-359	Exposure of Private Personal Information to an Unauthorized Actor
CWE-377	Insecure Temporary File
CWE-379	Creation of Temporary File in Directory with Insecure Permissions
CWE-402	Transmission of Private Resources into a New Sphere (*Resource Leak*)
CWE-424	Improper Protection of Alternate Path
CWE-425	Direct Request (*Forced Browsing*)
CWE-441	Unintended Proxy or Intermediary (*Confused Deputy*)
CWE-497	Exposure of Sensitive System Information to an Unauthorized Control Sphere
CWE-538	Insertion of Sensitive Information into Externally-Accessible File or Directory
CWE-540	Inclusion of Sensitive Information in Source Code
CWE-548	Exposure of Information Through Directory Listing
CWE-552	Files or Directories Accessible to External Parties
CWE-566	Authorization Bypass Through User-Controlled SQL Primary Key
CWE-601	URL Redirection to Untrusted Site (*Open Redirect*)
CWE-615	Inclusion of Sensitive Information in Source Code Comments
CWE-639	Authorization Bypass Through User-Controlled Key
CWE-668	Exposure of Resource to Wrong Sphere
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-749	Exposed Dangerous Method or Function
CWE-862	Missing Authorization

CWE	Title
CWE-863	Incorrect Authorization
CWE-918	Server-Side Request Forgery (SSRF)
CWE-922	Insecure Storage of Sensitive Information
CWE-1275	Sensitive Cookie with Improper SameSite Attribute