

What is it?

Broken Access Control occurs when applications fail to enforce proper user permissions, allowing unauthorised access to sensitive data, system functionality, or administrative actions.

Common causes include:


- 🔒 **Failure to implement the Principle of Least Privilege (PoLP):** Granting users more permissions than needed.
- 🔒 **Insecure direct object references (IDOR):** Exposing database keys or file paths in URLs.
- 🔒 **Client-side access controls:** Allowing attackers to bypass restrictions by modifying requests.
- 🔒 **Unprotected APIs and endpoints:** Failing to validate permissions properly.

How is it exploited?

Simply put, attackers manipulate application requests to gain unauthorised access, often using:

- 👤 **URL manipulation:** Modifying parameters, for example changing 'user_id=123' to 'user_id=456', allowing them to access another user's data.
- 👤 **Privilege escalation:** Acting as an admin while logged in as a regular user, perhaps by altering cookies, JSON Web Tokens (JWT), or request metadata.
- 👤 **CORS misconfiguration:** Exploiting improperly configured Cross-Origin Resource Sharing (CORS) to access APIs from untrusted origins.
- 👤 **Force browsing:** Manually accessing restricted pages or admin panels by guessing URLs.

Real-world example

In 2014,  Snapchat's API flaw allowed attackers to scrape 4.6 million usernames, phone numbers, and locations, simply by bypassing authentication checks - an example of how simple and powerful access control exploits can be.

How do you prevent it?

The Secure by Design approach ensures access control failures are mitigated before attackers can exploit them by, amongst other things:

- ✅ **Denying access by default:** Only allow permissions that are explicitly granted.
- ✅ **Using server-side enforcement:** Never rely on client-side access controls.
- ✅ **Implementing Role-Based Access Control (RBAC):** Assign permissions based on user roles.
- ✅ **Applying the Principle of Least Privilege (PoLP):** Grant only the minimum access required.
- ✅ **Using secure session management:** Invalidate tokens upon logout, use MFA, implement session binding, and limit session lifetimes.
- ✅ **Using rate-limiting and API security:** Prevent brute-force attempts on access control mechanisms.
- ✅ **Adopt a Zero Trust model:** Authenticate users and services continuously rather than relying on session persistence alone.
- ✅ **Conducting security testing:** Automate functional access control tests to detect vulnerabilities early.
- ✅ **Monitoring and logging failures:** Track unauthorised access attempts to alert admins.

