**Engineer**

# Access control

granting or refusing each request to use a resource

## What it is

Access control is the part of a system that answers one question many times a second: may this caller do this thing to this resource, right now? The caller can be a person, a service or a scheduled job. The resource can be a file, a database row, an API endpoint, a git repository or a door. The individual mechanisms that do the answering are called access controls, and NIST's glossary lists authorization as a synonym for the whole idea. Its definitions cover both halves of the job: "the decision to permit or deny a subject access to system objects", and "the process of granting or denying specific requests" to obtain and use information and related processing services, or to enter specific physical facilities.

It is easy to blur with authentication, and the two are separate steps. NIST SP 800-162 says so in plain words: "Authentication is not the same as access control". Authentication establishes who is asking. Access control takes that identity, along with whatever else is known about the request, and decides what the caller may do with it. A system can verify a person perfectly and still let them read records that belong to someone else, because the second step was never asked.

The same guide splits the work in two. A policy decision point computes the answer, and a policy enforcement point sits in the path of the request and carries it out. Both can live in one process, but the split is worth keeping in mind, because a decision that nothing enforces protects nothing. NIST SP 800-53 writes it into two controls: AC-24 asks that decisions be applied to each access request before enforcement, and AC-3 asks that approved authorizations be enforced.

The rules can take several shapes: a list of permitted identities on each object, roles as in RBAC, or attributes of the caller, the object, the action and the environment, which is the subject of SP 800-162. Whatever the shape, something

still has to ask the question on every request.

### Use the concept in a decision

Put “What is access control?” to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

**Explore our services** Get in touch →

### Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

**Wrote 1,000 RBAC rules to secure Linux servers** Get in touch →

## Why it matters

Once someone is signed in, nearly everything they can reach is decided by access control. Strong authentication in front of weak access control only records which account took the data. For a business that holds other people’s information, this is where a promise about who sees what holds or breaks.

NIST describes the ideal enforcement mechanism as a reference monitor. Control AC-25 in SP 800-53 asks for one that is tamperproof, always invoked, and small enough to be analysed and tested. Those three properties work as a practical test for any check in a real product. If the caller can alter the inputs the check relies on, it fails the first. If some route reaches the data without passing through the check, it fails the second. If the logic is scattered across hundreds of handlers so that nobody can read it end to end, it fails the third.

The OWASP Application Security Verification Standard turns this into requirements a team can test against. Its authorization chapter asks that access to functions be limited to callers with explicit permission, that access to specific data items be limited in the same way, and that the rules be enforced at a trusted service layer rather than relying on anything an untrusted client could manipulate, such as JavaScript running in the browser. It also asks that a change to the values a decision depends on take effect immediately, and where it cannot, that a mitigating control raise an alert when someone acts after losing the right to.

The trade-off is cost on the request path. Every decision needs facts, and every fact needs a lookup. Skipping the lookup tends to move the check to wherever it is cheapest, which is often the place the caller controls.

# What it looks like in practice

On a maritime platform where one person can act on behalf of several organizations, the question of who decides was settled by where the decision is made: the active organization is kept on the client and none of it is trusted for access decisions. An OrganizationContext stores the current choice in localStorage, so switching is instant, and a server-side cookie mirrors it so server-rendered pages resolve the same context. Authorization is re-checked on the server on every request. The page draws the line in one sentence: the server is the only authority on what you are allowed to do, and someone tampering with localStorage changes what their own interface shows them and nothing more, because the server still says no.

On the same kind of platform, taking money and granting permission were built as two systems. Cards through Stripe and in-app purchases from Apple and Google converge on a payments schema of 9 tables and 34 functions. A separate access schema of 11 tables and 34 functions answers “may this account do this?” without knowing which storefront paid. That answer gates the company directory, search and data export, and it is re-checked on the server on every request, because a hidden button is a courtesy and not a control. Adding a storefront now touches only the payments side, and a support question about someone’s access has one table to look at instead of three. The page names the cost: two schemas, and an entitlement lookup on requests that would otherwise be free.

When the company moved its source code onto its own hardware, the git forge was set up to refuse by default. The page lists access control among the things running a forge means running. The forge was chosen for having no administrative web interface, on the reasoning that the less interface there is, the less there is to defend. The policy starts from refusal: a caller with no identity gets nothing, a caller with no key gets nothing, and every declared repository carries the private flag, so secrecy rests on the policy rather than on obscurity. Reachability is part of the same policy. Port 23231 answers on loopback only, which makes a jump host the single way in from outside, and the multiplexer that could expose git on the public port sits in the code with its switch off: a listener nobody uses is surface.

## Where it goes wrong

**The check lives in the interface.** A button is hidden or greyed out, and the endpoint behind it answers anyone who calls it directly. The server has to make the decision again on every request, from facts the caller cannot edit.

**The function is checked and the record is not.** A request names an invoice, a document or an account by its identifier, the server confirms that the caller may view invoices, and returns the one asked for without checking whose it is. ASVS names this gap insecure direct object reference (IDOR), or broken object level authorization (BOLA). A defender recognises it by an identifier in a request that the server uses as given. The fix is to resolve every object through the caller's own scope, and in a product with several tenants, to take the tenant from the server-side session and never from the request.

**Anything new is open until someone closes it.** A new route, a new table or a new repository starts life with whatever the default is, and if the default is allow, the gap exists until someone notices. SP 800-53 states the network form of the opposite rule as deny by default, allow by exception, and the same rule fits every layer: what was never granted is refused.

**Grants only ever grow.** SP 800-162 notes that when access is not removed or revoked as time goes on, users accumulate privileges. People change teams, projects end and contractors leave, and each of those is a moment when something should be taken away. A revocation that only takes effect when a session expires hours later is a softer version of the same gap.

**Nobody can say why a request was allowed.** When decisions are made in many places and logged in none, an investigation has nothing to read. Record grants and refusals with the rule that produced them, and protect that log: SP 800-162 warns that the pattern of decisions can reveal attributes of the people behind the requests.

## Further reading

- Access control, in the NIST CSRC glossary — definitions from FIPS 201-3 and CNSSI 4009, with authorization listed as a synonym.
- NIST SP 800-53 Rev. 5 — the access control family, including AC-3 enforcement, AC-24 decisions and AC-25.

- NIST SP 800-162 — attribute-based access control, with policy decision and enforcement points.
- OWASP Application Security Verification Standard — testable authorization requirements at function, data and field level.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

**See what it costs** Get in touch →

## Make the terminology practical

Ask how “What is access control?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

**Explore our services**

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/access-control/