Engineer

Validation

checking input against the rules before trusting it

What it is

Validation is the step in which a program compares incoming data with the rules that data is meant to follow and refuses it when it falls short, before anything is stored, executed or sent on. It goes by several names depending on the layer: input validation for form fields and API requests, schema validation when the rules are written down as a JSON Schema or an OpenAPI description, and markup validation when a web page is checked against the grammar of HTML. A piece of input that passes is called valid.

OWASP's guidance splits the work into two questions. Syntax asks whether a value has the expected type and format: is this a date, is this an integer, does this body parse as JSON. Semantics asks "whether the value makes sense for the operation", so a hotel booking needs two valid dates and also an end date after its start date. Converting the string "500" into an integer settles the first question and says nothing about whether 500 nights is an acceptable stay.

Markup validation is the oldest public form of the idea on the web. The W3C Markup Validation Service describes validation as checking a document against the grammar it claims to use, and compares its validator to lint for C. Its documentation is careful about what that proves: a valid page is not necessarily a good one, only one that a tool found compliant with a specific set of rules. The same caution applies to every other kind. Validation tells you the data fits the rules you wrote; it does not tell you the rules were the right ones.

Two neighbouring ideas are often confused with it. Sanitisation changes input to make it acceptable, by stripping or rewriting parts of it, where validation accepts or rejects it as it stands. Encoding and parameterisation make data safe for the place it is going, an HTML page or an SQL query, and they are still needed after validation has passed.

Use the concept in a decision

Put “What is validation?” to work in a real decision. We can help relate the concept to the system you are building or buying. Rules and checks that catch bad data on the way in, and a clear answer to who owns which figure and where it came from.

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. One infrastructure plan for 14 departments — 14 research departments on one infrastructure plan

One infrastructure plan for 14 departments Get in touch →

Why it matters

The decision behind validation is where a system draws its trust boundary and what it checks there. Every request from a browser, every message from a queue, every file from a partner and every configuration change from a colleague arrives with assumptions attached. Checking those assumptions at the edge means the rest of the code can rely on them; skipping the check means every function downstream has to defend itself, and in practice most of them will not.

The second decision is how the rules are written. Checks written out by hand, endpoint by endpoint, start out cheap and drift apart as the system grows, because each one is its own small piece of code. Rules declared once, as types or as a schema, can be enforced by a library and published as documentation from the same source, reducing the risk that the description and the behaviour drift apart. The price is a framework to learn and defaults to understand, because a schema enforces exactly what it says and nothing it leaves unsaid.

Strictness is the last choice. An allowlist describes what is acceptable and rejects everything else, which OWASP recommends over trying to recognise every malicious string. Rejecting fields a schema does not know about catches typos and stale clients early. Rejecting the whole request on the first problem, with a clear error, is safer than carrying on with data that was only partly checked.

What it looks like in practice

One API began with request validation written by hand for each endpoint. At the documented implementation stage, moving it to Huma v2 turned the checks into types across 649 paths and 760 operations. Each endpoint had input and output structs, Huma generated the request validation from them, and an OpenAPI description came from the same types. Two WebSocket endpoints stayed on raw Fiber, where the request and response model did not fit. A CI check compared

the routes the server registered with the ones the description advertised. Recorded route parity was 100%; checks declared in the input types replaced the corresponding hand-written validation.

Configuration needs the same treatment, and SSH hardening on the company's host now runs three stages of validation on every converge. The candidate file is syntax-checked before it is installed, the assembled configuration is checked after installation, and then the daemon's own resolved settings are read back and 24 directives asserted against them. The third stage exists because an access report in September 2026 found two settings that the automation had been writing on every converge and the daemon had been overriding, because the operating system's own defaults were read first and, for those two, the first occurrence wins. As the page puts it, grepping for a directive only proves it was written.

Markup validation was part of the documented August 2026 implementation stage: HTML validation was one of the checks in a 22-linter commit gate with no warning tier, where every diagnostic was an error and inline suppressions were prohibited, so an invalid page could not be committed by marking the error as acceptable.

Where it goes wrong

The server trusts what the browser already checked. A form that refuses a bad email address in the browser looks validated, but the request can be sent without the form at all. OWASP lists client-only checks as a common pitfall: validate on the server, and treat the browser check as a quick hint for the person typing.

A schema accepts everything it does not mention. In JSON Schema, listing a property neither makes it required nor forbids other fields, and under the default 2020-12 vocabulary the `format` keyword is collected as an annotation, with format checking off by default. This schema looks strict:

```
{
    "type": "object",
    "properties": {
        "email": { "type": "string", "format": "email" }
    }
}
```

It accepts `{}`, `{"email": "nope"}` and `{"emial": "a@example.com"}`. Add `required` and `"additionalProperties": false`, and turn on format assertion in the validator, or check the format in code. Huma, for its part, rejects unknown body fields by default, documents `readOnly` and `writeOnly` without enforcing them, and ignores unknown query parameters unless told otherwise.

The check passes on one version of the data and the program uses another. A validator that reads the raw request while the handler reads a decoded copy, or a test that inspects the file on disk while the daemon applies a different setting, has checked something the system never uses. OWASP's advice reads: "Validate the representation that will actually be used". The SSH work above is the same lesson in configuration: only the daemon's read-back showed what was in force.

A denylist meets the next variant. Blocking known bad characters rejects legitimate input, an apostrophe in a surname for instance, and still leaves the query unsafe. Define what each field accepts, and reject the rest.

Valid input is treated as safe input. A string that passed every rule can still break an SQL statement or a page when it is pasted into one. Validation narrows what arrives; parameterised queries and output encoding protect the place where it ends up, and both are needed.

Further reading

- OWASP Input Validation Cheat Sheet — syntax and semantics, allowlists, parsing limits and common pitfalls.
- W3C Markup Validator documentation — the validator's own account of markup validation, with its help pages on what validity does and does not prove.
- JSON Schema Validation, draft 2020-12 — the validation keywords, including how `format` behaves by default.
- Huma: Request Validation — validation generated from Go struct tags, with the strict and loose field settings.
- OpenAPI Specification — the description format whose schemas many API validators enforce.

Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is validation?” 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/validation/