Engineer

HSTS

a browser's promise to use HTTPS next time

What it is

HTTP Strict Transport Security, nearly always shortened to HSTS, is a response header that tells a browser to reach a site only over HTTPS for a stated number of seconds, and to refuse the connection outright when the certificate is wrong. The header is named `Strict-Transport-Security`, and RFC 6797 has defined it since November 2012. It encrypts nothing itself; it changes what a browser is willing to do over HTTPS.

A typical value carries two directives:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

`max-age` is how long the browser keeps the rule, here one year. With `includeSubDomains` the rule reaches every name below the host, so `api.example.com` and `old.example.com` inherit what `example.com` asked for. A third token, `preload`, appears in many configurations. It comes from the preload list run at hstspreload.org rather than from the RFC, and it is the site's consent to have its name built into browsers.

Once a browser has seen the header on a secure response, it does three things for that host until the time runs out. It rewrites every `http://` address to `https://` before a request leaves the machine. It treats any certificate error as fatal, with no button to carry on regardless. And it restarts the clock each time the header arrives again. A response carrying `max-age=0` makes it forget the host.

The usual comparison is a redirect from port 80 to port 443. A redirect works, but only after the browser has sent its first request in the clear, and anyone on the same network can answer that request themselves and keep the visitor on an unencrypted copy of the site. HSTS removes that plain request from every visit after the first. Upgrade Insecure Requests, a Content Security Policy directive, is the other neighbour: it upgrades the requests a page makes for its own scripts and images, and leaves a visitor who arrives from someone else's `http://` link to the redirect.

Use the concept in a decision

Put "What is HSTS?" 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

Turning HSTS on costs one line of configuration. The real decision is how far to take it, because each step protects more visitors and is harder to take back.

The header alone leaves a gap that RFC 6797 names itself. A browser that has never seen the site, or whose remembered policy has expired, still starts with a plain request. A long `max-age` shrinks the second half of that gap, which is why a year is the common setting. The first half closes only through the preload list: host names compiled into Chrome, with Firefox, Safari and Edge keeping lists based on it, so that a browser fresh from the installer already knows to use HTTPS.

The list asks a great deal. The base domain needs a valid certificate, a redirect from HTTP to HTTPS that stays on the same host, every subdomain served over HTTPS, and a header with a `max-age` of at least 31536000 seconds, `includeSubDomains` and `preload`. Its maintainers say plainly that this covers internal subdomains the public never sees. Leaving is the part that settles the question: a removal takes months to reach users through Chrome updates, the other browsers come with no guarantee, and the site describes removal as "slow and painful".

So the choice runs up a ladder, one rung at a time:

- **The header without** `includeSubDomains` suits almost any site that already serves HTTPS. RFC 9325, the current best practice for TLS, says web servers should send it.
- **Adding** `includeSubDomains` is safe once someone has listed every name under the domain and confirmed that each one answers on HTTPS, including the ones a vendor or another department runs.
- **Preloading** fits a domain whose owners are certain no name under it will need plain HTTP again: a login domain, a payments domain, a company that controls all of its own DNS. A marketing domain where a campaign tool may soon ask for a CNAME is a poor candidate, and so is any domain shared with office equipment.

The list's maintainers also give the pace for the climb: `max-age` at 300 seconds, then a week, then a month, each with `includeSubDomains`, waiting out the full age of one stage before starting the next. A browser keeps whichever value it saw last, so a mistake made at the five-minute rung costs five minutes.

What it looks like in practice

On one platform holding professional and organizational data, the company worked as Security Engineer and hardened every layer, and HSTS was one of them. In the Next.js proxy middleware, `proxy.ts`, the response carries HSTS alongside a Content Security Policy with a fresh nonce on every request and `strict-dynamic`, and SameSite cookies. A proxy middleware runs in front of the routes it matches, which keeps the browser-facing rules in one file rather than spread across pages.

The same piece of work shows where HSTS stops. Its working assumption was that the client is hostile: anything the browser enforces can be switched off by whoever is holding the browser. HSTS protects an honest visitor from a hostile network. It does nothing about a hostile client, which is why the work went on below it. The API added rate limiting, CORS, request-size limits, input validation before anything reached the database, and logging of security-relevant events. The API connected to the database as a role that could only execute the application's functions, and entitlements were checked again on the server on every request, with the frontend's gates treated as user experience only. In that stack HSTS keeps the traffic itself out of reach, and it is the cheapest layer to add.

Where it goes wrong

The header is in the configuration and browsers behave as though it were absent. It is being sent on the plain HTTP response, usually the redirect, or on an HTTPS response whose certificate the browser rejected. RFC 6797 tells browsers to ignore it in both cases and tells servers to leave it off plain responses altogether. A related slip sends `example.com` straight to `https://www.example.com` with the header only on `www`, which leaves the base domain and its other names uncovered. Put the header on every HTTPS response, redirects included, and check both doors:

```
curl -sI https://example.com/ | grep -i strict-transport-security
curl -sI http://example.com/ | grep -i strict-transport-security
```

The first line should print the header. The second belongs to the redirect and should print nothing at all.

An old service under the domain stops loading for everyone who has visited the main site. `includeSubDomains` on the base domain covers every name beneath it: a status page hosted by a vendor, a printer’s admin page, an intranet tool that never had a certificate. Those visitors get a connection error with no way past it, while colleagues who never opened the main site reach the service without trouble, so the reports are hard to reproduce. The directive went out before anyone made the list. Make it first, from the DNS zone and from certificate transparency logs, which record every name a public certificate was issued for. Move each name to HTTPS or to a different domain, then add the directive at a short `max-age`.

A warning that staff used to click through becomes a wall. Staging on a self-signed certificate, an internal certificate authority that visitors' laptops do not trust, a certificate that lapsed over a weekend: before HSTS each of these was a warning page with a way past it, and afterwards the browser offers none. RFC 9325 names self-signed certificates as a case where HSTS can leave security worse off. Give staging a publicly trusted certificate or a domain the policy does not reach, and treat certificate renewal as something that now takes the site down when it fails, with no choice left to the visitor.

A preloaded domain needs plain HTTP back. A device that speaks only HTTP, a vendor whose custom-domain feature serves no certificate, an acquisition that brings its own hosts under the name. The entry already sits inside browsers that have shipped, and the server can do little about it: a removal request reaches users only as their browsers update. Put the new name under a different domain, and preload only after a long run at a one-year `max-age` with `includeSubDomains` has raised no tickets.

Further reading

- RFC 6797: HTTP Strict Transport Security (HSTS) — the header, its two directives, and exactly when a browser must honour it or ignore it.
- HSTS Preload List Submission — the conditions for being listed, the staged rollout the maintainers advise, and what leaving costs.
- RFC 9325, section 3.2 — the current best practice for TLS on HSTS: clients and servers must support it, and servers should send it.
- Upgrade Insecure Requests — the W3C directive that upgrades a page's own requests, and how it divides the work with HSTS.



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 HSTS?” 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/hsts/