Engineer

Cookie

a small value a site asks the browser to keep

What it is

A cookie, or HTTP cookie, is a short name and value that a web server sends to the browser in a response header, and that the browser returns with its later requests to the same server. Cookies are how HTTP, a protocol that treats every request as a stranger, gets a memory: a login that survives the next click, a basket still full at the checkout, a language chosen once and kept.

The server sets each cookie with its own `Set-Cookie` header, and the browser sends back every cookie that applies in one `Cookie` header. Here the first three lines are a response and the rest is a later request from the same browser:

```
HTTP/1.1 200 OK
Set-Cookie: __Host-sid=4f2c9a71e0b8; Path=/; Secure; HttpOnly; SameSite=Lax
Set-Cookie: lang=da; Max-Age=31536000; Path=/

GET /account/ HTTP/1.1
Host: shop.example
Cookie: __Host-sid=4f2c9a71e0b8; lang=da
```

Everything after the first semicolon is an attribute, an instruction about how long to keep the cookie and where to send it. `Expires` or `Max-Age` sets a maximum lifetime; with neither, the cookie lasts until the browser decides the session is over. `Domain` widens the cookie to a whole domain and its subdomains, and without it the cookie goes back only to the host that set it. `Path` narrows it to part of the site. `Secure` keeps it off unencrypted connections, `HttpOnly` hides it from scripts that read `document.cookie`, and `SameSite` decides whether it rides along on requests another site starts.

The return trip carries names and values only, so the server cannot tell from it when a cookie expires, whether it was set with `Secure`, or who last wrote it.

The rules are in RFC 6265, published in 2011. Its revision, known as rfc6265bis, adds `SameSite`, the `__Secure-` and `__Host-` name prefixes and a requirement that browsers cap every lifetime at no more than 400 days. At the time of writing it has been approved and is waiting for the RFC Editor to publish it as the replacement for RFC 6265.

Use the concept in a decision

Put “What is a cookie?” to work in a real decision. We can help relate the concept to the system you are building or buying. A site that is quick, findable and editable by your own staff, without a bill for every change of wording.

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. Developed 100 web applications using HTML/HTML5 — 100 web applications built

Developed 100 web applications using HTML/HTML5 Get in touch →

Why it matters

A cookie is usually the only thing that makes a signed-in request different from an anonymous one. After a login, the session identifier in the cookie is the user as far as the server is concerned: whoever presents it gets the account, and no password is asked for. Most of what goes wrong with cookies is somebody else reading, copying or using one.

The browser makes the last of those easy. It attaches cookies to every request that matches them, whoever started that request. A form on another site that posts to a shop arrives carrying the visitor’s cookies, and the server sees a request from a signed-in user. RFC 6265 calls this ambient authority and names the attacks it invites: cross-site request forgery and the confused deputy.

Cookies are also the oldest tracking tool on the web. A script embedded on thousands of sites can set a cookie from its own domain, which the browser sends back from every one of those sites, so one company can line up a person’s visits across all of them. The revision describes this use of third-party cookies, notes that most browsers now block or partition them, and recommends a policy as restrictive as compatibility allows.

The law follows the storage rather than the technology. In the EU, Article 5(3) of the ePrivacy Directive allows a site to store information on a visitor's device, or to read what is stored there, only with consent given after clear and comprehensive information. The exceptions are storage used solely to transmit a communication, and storage strictly necessary for a service the visitor explicitly asked for. That article is where consent banners come from, and it never mentions cookies by name. The European Data Protection Board's guidelines on its technical scope apply it to tracking pixels and tracking links too.

What it looks like in practice

This site sets none. Its cookies page explains why: there are no accounts, no basket and nothing to follow a visitor with, so there is no consent banner, because there is nothing to consent to. It also lists the other places a site can leave something behind, from `localStorage` to a service worker, and says none of them is used either.

Going without cookies changes how a site learns about its readers. Counting readers from the access log alone measures readership with no cookie, script or pixel, from the web server's own log, and drops the `Cookie` request header from every line as it is written, along with the authorization header. The result is coarse by design: unique visitors and a visitor's path through the site are out of reach, and the page prints those limits beside the numbers.

Where an application does need a cookie, the useful question is what it is trusted with. On a platform where one person acts for several organizations, the active organization is kept in the browser, and a server-side cookie mirrors it so that pages rendered on the server show the same context. Neither copy decides access. The server checks authorization on every request, so someone who edits the value changes what their own screen shows and nothing more. That is the safe shape for any cookie: the value may choose what to render, and the decision about what someone may do stays on the server.

Where it goes wrong

Script on the page reads the session. Without `HttpOnly`, a script an attacker injects can read `document.cookie` and send the session identifier anywhere. `HttpOnly` takes the cookie out of that API. It stops theft, not misuse: the injected script can still send requests, and the browser still attaches the cookie, so `HttpOnly` sits beside a Content Security Policy rather than replacing one.

The cookie crosses plain HTTP. A cookie without `Secure` goes out on any `http://` request to its host, readable by anyone on the network path. Under RFC 6265, `Secure` guards confidentiality alone, and an attacker on the network can still overwrite a secure cookie from an insecure connection. The revision narrows that gap by ignoring `Secure` cookies set over plain HTTP and refusing a plain-HTTP cookie that would overlay a secure one.

A sibling subdomain sets the cookie. `Domain=example.com` sends a cookie to every host under that domain, and any of those hosts can set a cookie for the whole domain, which the main site then reads as its own. `Path` is no security boundary either. The `__Host-` prefix answers this: a browser accepts such a cookie only when it is `Secure`, has `Path=/` and has no `Domain`, which locks it to the one host that set it.

Another site starts the request. `SameSite=Strict` keeps the cookie to requests made from the same site, `Lax` adds top-level navigations with safe methods, such as following a link, and `None` sends it everywhere and is ignored unless the cookie is also `Secure`. The revision treats a cookie with no `SameSite` much like `Lax`; a server that depends on that should say so. A site, for this purpose, is the scheme plus the registrable domain, so `SameSite` does nothing against a hostile sibling subdomain.

The server believes the value. Anything in a cookie can be edited by the person whose browser holds it. RFC 6265 recommends that servers encrypt and sign cookie contents, and warns that this still leaves a cookie open to replay. The sturdier pattern is a random session identifier pointing at state kept on the server, replaced at login, so that an identifier an attacker planted beforehand, the session fixation the RFC warns about, means nothing afterwards.

The browser lets it go. An expiry date is a maximum. Browsers evict cookies to stay within their limits, and an attacker can push cookies out by setting many of its own. Servers should not rely on browsers keeping cookies. Deleting one has a trap too: a `Set-Cookie` with an expiry in the past removes the cookie only when its `Path` and `Domain` match the ones it was created with.

Further reading

- RFC 6265: HTTP State Management Mechanism — the standard for `Set-Cookie` and `Cookie`, with the attribute rules and a security section on ambient authority, weak integrity and session identifiers.

- Cookies: HTTP State Management Mechanism (rfc6265bis) — the approved revision, with `SameSite`, the `__Secure-` and `__Host-` prefixes, the 400-day cap and a section on third-party cookies.
- Directive 2002/58/EC, consolidated text — the ePrivacy Directive as amended in 2009, with Article 5(3) on storing and reading information on a user's device.
- EDPB Guidelines 2/2023 on the technical scope of Article 5(3) — how the rule reaches tracking pixels, tracking links, local processing and other techniques beyond cookies.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is a cookie?" 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/cookie/