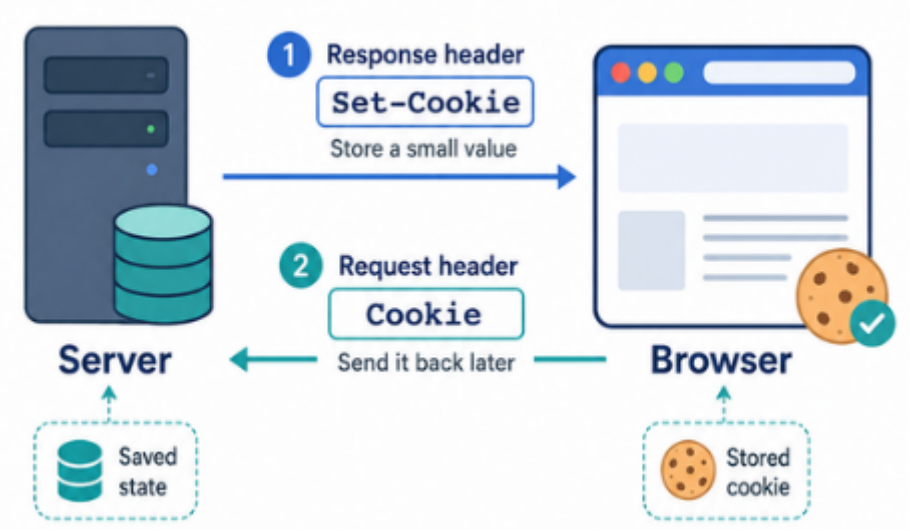
Cookies

Written by [Febogamedev]

HTTP cookies, or simply **cookies**, are small name–value pairs that a server can ask the browser to store (Barth, 2011).
In subsequent requests, the browser can automatically return those values to the server when the conditions associated with the cookie are met (Barth, 2011).
Cookies therefore allow an application to recognize a relationship between different *HTTP* requests. They are used, for example, to (Barth, 2011; OWASP Foundation, n.d.):

- Maintain an authenticated session.
- Remember a language preference.
- Store the contents of a shopping cart.
- Remember certain settings.
- Associate successive requests with the same session.
- Implement statistical or tracking systems.

The server sets a cookie through the response header **Set-Cookie;** the browser can subsequently return it through the **Cookie request** header (Barth, 2011).
This mechanism allows application state to be maintained over *HTTP*, which does not automatically associate a request with previous ones (Barth, 2011).
Cookies do not turn *HTTP* into an inherently stateful protocol (Barth, 2011).
Each request continues to be a self-contained message. However, the cookie provides the server with information that allows it to link the current request to previously stored state.
Cookies allow applications to maintain state across different *HTTP* requests by sending the browser small values that can be returned in subsequent requests (Barth, 2011).



A cookie can directly contain a simple preference, for example “lingua=it” to indicate that the user prefers to use Italian (Barth, 2011).

For sensitive or complex information, however, it is common to store only a random identifier in the cookie, such as session_id=7f3a9c81.. (OWASP Foundation, n.d.).
«The actual data can remain on the server and be associated with that random identifier:

```
Cookie in the browser
session_id=7f3a9c81
          ↓
Database or session store
7f3a9c81 → user 52, authenticated, Italian language
```

The browser does not need to understand the meaning of the identifier: it simply stores it and returns it according to the specified rules. The server application can use the value to retrieve the corresponding state. The server is nevertheless free to ignore the received cookie (Barth, 2011).

The browser keeps cookies in its own internal storage area. This may be implemented using files, internal databases, encrypted stores or other browser-dependent structures.
Cookies are suitable for small amounts of data. They should not be used to store files, images or large amounts of information, partly because they are attached to requests to which they apply (Barth, 2011).

The server can ask the browser to store a cookie through the header (Barth, 2011):
Set-Cookie: session_id=7f3a9c81.

Example shown in the textual syntax of *HTTP/1.1*:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: session_id=7f3a9c81; Path=/; Secure; HttpOnly; SameSite=Lax
```

Note that a response can contain multiple Set-Cookie headers, one for each cookie (Barth, 2011).

When the browser makes a request that matches the scope of the cookies, it can send (Barth, 2011):

```
GET /profilo HTTP/1.1
Host: example.com
Cookie: session_id=7f3a9c81; lingua=it
```

The Cookie header normally returns only names and values. In this case (Barth, 2011):

- session_id=7f3a9c81.
- lingua=it.

Attributes such as Secure, HttpOnly, Path and SameSite are not sent back to the server: the browser uses them to determine when and how to use the cookie (Barth, 2011; Bingler et al., 2025).

The browser therefore does not send a cookie to any website that requests it (Barth, 2011; Bingler et al., 2025).
It automatically decides whether to attach it by considering:

- Host or domain.
- Path.
- Connection type.
- Same-site or cross-site context.
- Lifetime.
- Browser policies and settings.

The basic structure of a cookie consists of a name=value pair (Barth, 2011).

Example
Set-Cookie: lingua=it

- "lingua" is the name.
- "it" is the value.

The name and value make up the actual cookie. They can be followed by attributes that control its lifetime, scope and protection (Barth, 2011; Bingler et al., 2025).
The main attributes are:

- Domain.
- Path.
- Secure.
- Expires.
- Max-Age.
- HttpOnly.
- SameSite.

Domain
The Domain attribute indicates the hosts to which the cookie can be sent (Barth, 2011).
Set-Cookie: preferenza=it; Domain=example.com
In this case, the cookie can be sent to:

- example.com.
- www.example.com.
- account.example.com.

When Domain is omitted, the cookie is normally limited to the host that set it and is not sent to its subdomains. Omitting Domain therefore produces a more restricted cookie (Barth, 2011).

Path
The Path attribute limits the paths for which the browser sends the cookie (Barth, 2011).
Set-Cookie: preferenza=it; Path=/blog

The cookie can accompany requests to:

- /blog.
- /blog/.
- /blog/articolo-1.

When Path=/, the cookie can be used for all matching paths on the host (Barth, 2011).
When Path is omitted, the default value is derived from the path of the resource that set the cookie (Barth, 2011).

Secure
Secure tells the browser to send the cookie only over a secure connection, normally HTTPS (Barth, 2011).

Expires
Specifies a date after which the browser must no longer store the cookie (Barth, 2011).

```
Set-Cookie: lingua=it; Expires=Wed, 18 Aug 2027 10:00:00 GMT
```

If Expires is not defined, the cookie lasts until the end of the session defined by the browser (Barth, 2011).

Max-Age
A cookie can specify a maximum lifetime through Max-Age (Barth, 2011):
The value is expressed in seconds. In this example, the cookie can remain for about thirty days.

```
Set-Cookie: lingua=it; Max-Age=2592000
```

When both are present, Max-Age takes precedence over Expires (Barth, 2011).

HttpOnly
HttpOnly makes the cookie inaccessible through ordinary JavaScript APIs, but does not prevent it from being sent in HTTP requests (Barth, 2011).
This attribute is particularly important for cookies containing sensitive data because it reduces the chance of a malicious script directly reading their value (OWASP Foundation, n.d.).

SameSite
SameSite controls whether a cookie can accompany requests originating from the same site or from a different site; the latter is called a cross-site request (Bingler et al., 2025).

A cross-site request is a request made while the browser is in the context of one site but directed to a different site (Bingler et al., 2025).
For example, the user is visiting “https://negozio-a.com” and the page contains “<img src="https://immagini-b.com/foto.jpg">”.
To display the image, the browser makes a request to immagini-b.com:

```
negozio-a.com
      │
      │ HTTP request
      ▼
immagini-b.com/foto.jpg
```

The request is cross-site because it is made in the context of negozio-a.com but is directed to immagini-b.com.

It is useful to distinguish two main situations:
- Cross-site navigation.
- A cross-site resource request.

Cross-site navigation occurs when the browser moves from one site to another as the main page. For example, by clicking a link:



This is called top-level navigation because the main document displayed by the browser changes (Bingler et al., 2025).

A cross-site resource request, on the other hand, occurs when the browser stays on the current page, but that page requests a resource belonging to another site (Bingler et al., 2025).
For example: <img src="https://sito-b.com/foto.jpg">
The browser stays on sito-a.com but requests the image from sito-b.com.
Other examples are:
- Requests for subresources, such as images, scripts or style sheets.
- fetch() calls.
- Embedded content, such as iframes.

Understanding what a cross-site request is is essential to understanding the values of SameSite, which can be (Bingler et al., 2025):
- Strict.
- Lax.
- None.

SameSite=Strict
The cookie is sent only in requests considered same-site (Bingler et al., 2025).

For example, if the user reaches the site by clicking a link on a different site, the Strict cookie may not be sent in the initial request. This setting offers more restrictive protection, but can interfere with some navigation flows (Bingler et al., 2025).

SameSite=Lax
The browser sends the cookie in same-site requests and may also send it when the user navigates to the site that set the cookie from another site, that is, through top-level cross-site navigations using safe methods, such as a normal click on a link (Bingler et al., 2025).
However, it is not sent when the cross-site request is for a subresource (Bingler et al., 2025).

Lax often provides a compromise between security and normal browsing.

SameSite=None
The cookie can also be sent in cross-site contexts (Bingler et al., 2025).
This mode may be necessary for some embedded content or authentication systems spanning different sites (Bingler et al., 2025).

Example
Imagine that you are already authenticated on: https://banca.example
and that the browser has this cookie: session_id=ABC123

Now let us see what happens with the three values of SameSite.
Strict

```
SameSite=Strict
Set-Cookie: session_id=ABC123; SameSite=Strict; Secure
```

The cookie is sent only when the request takes place in a same-site context.
If you are already on banca.example and click: Profile → /profilo
the browser sends: Cookie: session_id=ABC123
But imagine that you are on: https://notizie.example
and click a link: Go to your bank → https://banca.example/profilo
With Strict, the cookie is not sent in the initial request from the external site.

Lax

```
SameSite=Lax
Set-Cookie: session_id=ABC123; SameSite=Lax; Secure
```

Here the behavior is more permissive.
If you are on notizie.example and click normally on:
https://banca.example/profilo
the cookie can be sent because this is a top-level navigation using a safe method, typically GET.

```
notizie.example
      |
      | click on a link
      ▼
banca.example/profilo
```

But if notizie.example automatically attempts to make a cross-site request, for example:
<img src="https://banca.example/operazione">
or an ordinary cross-site fetch() request:
fetch("https://banca.example/operazione");

the SameSite=Lax cookie is normally not attached in that cross-site context.

None

```
SameSite=None
Set-Cookie: session_id=ABC123; SameSite=None; Secure
```

The cookie can also be sent in cross-site contexts.
SameSite=None also requires Secure (Bingler et al., 2025).

A classic example of maintaining a login is shown below,

The user opens: https://example.com/login and enters their email and password.
The browser sends the credentials to the server through an HTTPS request, normally using the POST method (OWASP Foundation, n.d.).

At this point, the application:

- Finds the account.
- Checks the password.
- Checks the user's status.
- Creates a new session.

The server generates a random identifier: 7f3a9c81… and stores the association: 7f3a9c81 → Giulia, authenticated (OWASP Foundation, n.d.).
The server responds:

```
Set-Cookie: session_id=7f3a9c81; Path=/; Secure; HttpOnly; SameSite=Lax
```

The browser stores the cookie.
When the user requests: "https://example.com/profilo".
the browser can send: "Cookie: session_id=7f3a9c81".
The server looks up the received identifier.

```
valid session_id
        ↓
user recognized as Giulia
        ↓
access to the page authorized
```

The server then checks that the identifier still corresponds to a valid session (OWASP Foundation, n.d.).

- If the session is valid, the user enters the site as an authenticated user.
- If the session is no longer valid, the user must log in again by entering their credentials.

When the user logs out, the application should (OWASP Foundation, n.d.):

- invalidate the session on the server.
- ask the browser to delete the cookie.

For example:

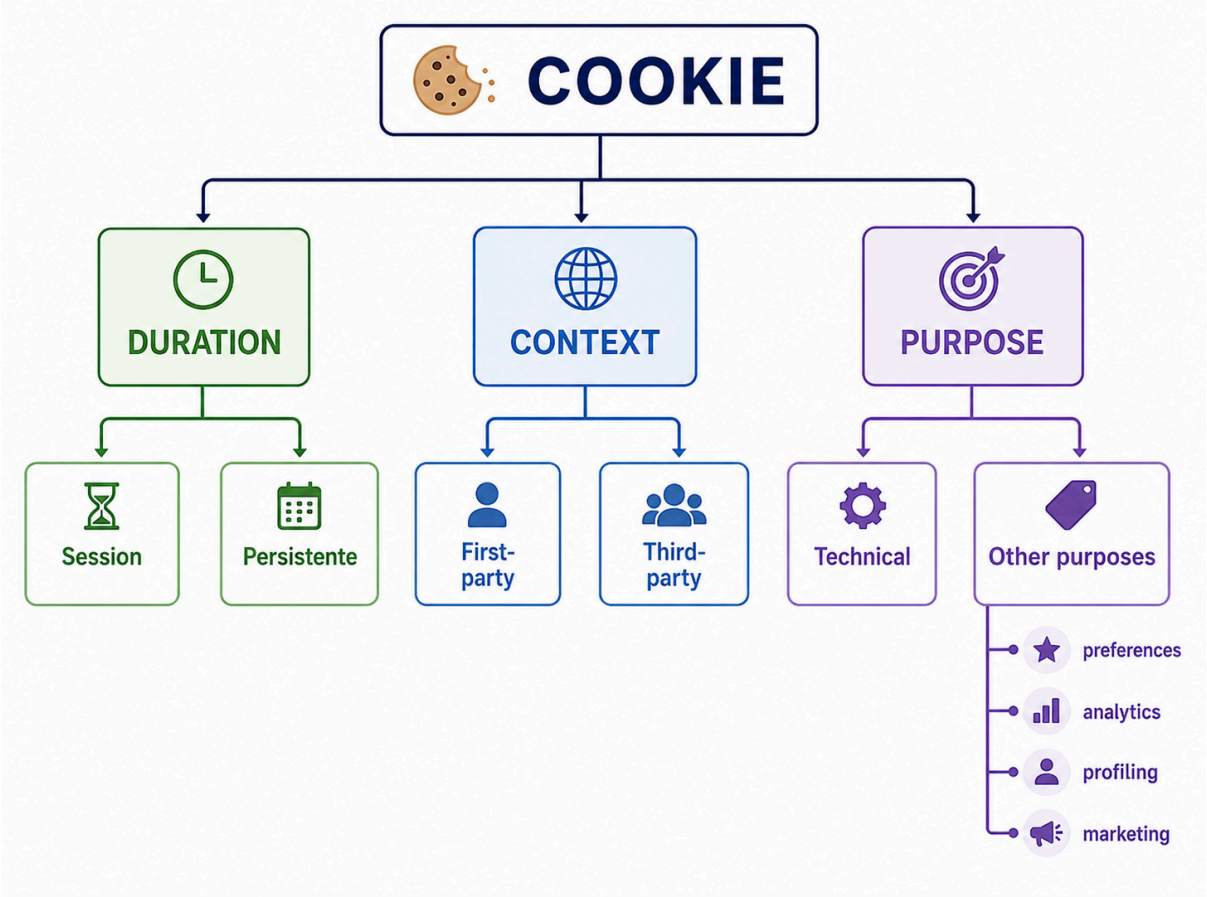
```
Set-Cookie: session_id=; Path=/; Max-Age=0; Secure; HttpOnly;
SameSite=Lax
```

Taxonomy

Cookies can be classified according to different criteria. The main ones are:

- Lifetime, meaning how long they are stored.
- Context of use, meaning whether they are used by the site being visited directly or by an external party.
- Purpose, meaning what they are used for.

These classifications are independent of one another. The same cookie can, for example, be persistent, first-party and used to store a preference.



Lifetime

Session Cookies

Session cookies do not have a persistent lifetime defined through the Expires or Max-Age attributes (Barth, 2011).

For example:

```
Set-Cookie: session_id=ABC123; Secure; HttpOnly
```

They are stored for the duration of the session established by the browser (Barth, 2011).

It is common to say that they are deleted when the browser closes, but this is a simplification: the actual management of the session depends on the browser, and some restore features can allow a session to survive even after the application closes. The cookie specification distinguishes persistent from non-persistent cookies based on the presence of an expiry time, while the actual duration of the session is managed by the browser (Barth, 2011).

Persistent Cookies

Persistent cookies, on the other hand, have a lifetime defined through Expires or Max-Age (Barth, 2011).
For example:

```
Set-Cookie: lingua=it; Max-Age=2592000
```

The browser can store the cookie for about 30 days.
Or:

```
Set-Cookie: lingua=it; Expires=Wed, 18 Aug 2027 10:00:00 GMT
```

In this case, an expiry date is specified.

During its validity period, the browser can send the cookie in requests that meet its conditions, such as domain, path, connection type and SameSite rules (Barth, 2011; Bingler et al., 2025).

A persistent cookie can be used, for example, to remember:

- The selected language.
- Certain interface preferences.
- An identifier that allows a browser to be recognized again.
- Information needed for particular persistent login mechanisms.

The fact that a cookie is persistent therefore does not mean that it is necessarily used for tracking.

Another distinction concerns the relationship between the site the user is visiting and the site to which the cookie belongs.

Context of Use

The common terms are:

- First-party cookies.
- Third-party cookies.

This distinction does not indicate the cookie's purpose: a first-party cookie can also be used for profiling, while a cookie used in a third-party context can also have a technical function (Garante per la protezione dei dati personali [Garante], 2021).

First-Party Cookies

A cookie is used in a first-party context when it belongs to the site with which the user is directly interacting (Bingler et al., 2025).
Imagine that the user visits:
https://negozio.example

The site could respond:

```
Set-Cookie: carrello=ABC123; Path=/; Secure; SameSite=Lax
```

The cookie therefore belongs to the site the user is visiting directly.

It could be used, for example, to remember:

- An authenticated session.
- A shopping cart.
- Language.
- Preferences.

Third-Party Cookies

A web page can embed resources or services from different sites (Bingler et al., 2025).
Imagine that the user visits: https://giornale.example

The page contains an advertising service from: https://pubblicita.example

For example:

```
<iframe src="https://pubblicita.example/banner">
</iframe>
```

We therefore have:

```
giornale.example
       |
       | embeds a service
       ▼
pubblicita.example
```

If pubblicita.example uses its own cookie while embedded in the first site, that cookie is being used in a third-party context (Bingler et al., 2025).

For example:

```
Set-Cookie: advertising_id=XYZ789;
SameSite=None;
Secure
```

SameSite=None allows the cookie to be used in cross-site contexts as well, when permitted by the other rules and browser policies; it also requires Secure (Bingler et al., 2025).

Third-party cookies have also been used to recognize the same browser as it visits different sites (Bingler et al., 2025).

Imagine that two sites use the same advertising service:

```
giornale.example
       |
       ▼
pubblicita.example
```

and subsequently:

```
negozio.example
       |
       ▼
pubblicita.example
```

If the browser allows the external service to use the same identifier, “advertising_id=XYZ789”, the service could observe:

```
giornale.example
       ↓
XYZ789
```

```
negozio.example
       ↓
XYZ789
```

and recognize that the interactions come from the same browser.

In this way, it may be possible to link activities carried out on different sites and build information about browsing behavior (Bingler et al., 2025).

This phenomenon is called **cross-site tracking**, meaning tracking across different sites (Bingler et al., 2025).

However, it is important to remember that the use of cookies in third-party contexts can be restricted or blocked by browser settings and policies (Bingler et al., 2025).

Warning: An External Script Does Not Necessarily Imply a Third-Party Cookie
The origin of the JavaScript script and that of the cookie must not be confused (Bingler et al., 2025; Google, n.d.-a, n.d.-b).
A page could load:

```
<script src="https://servizio-esterno.example/script.js">
</script>
```

but the script could still create or use a cookie associated with the site being visited.

Example
Imagine a website: https://miosito.com.

The owner uses Google Analytics 4 (a service that collects statistics such as the number of users, etc.).
The Google tag is inserted into the page and loads a JavaScript script from Google's servers:

```
<script async
  src="https://www.googletagmanager.com/gtag/js?id=G-XXXXXXXXXX">
</script>
```

We therefore have:

```
miosito.com
     |
     | requests an external script
     ▼
www.googletagmanager.com
     |
     ▼
gtag.js
```

The script is therefore an external resource because it is downloaded from a domain other than miosito.com.
Once downloaded, however, the JavaScript code runs within the page on miosito.com.
Google Analytics can thus use a cookie such as: _ga=GA1.1.123456789.987654321 associated with miosito.com and not with googletagmanager.com

The "_ga" cookie is used by Google Analytics to store a browser identifier, called the "Client ID", which allows browser instances/clients to be distinguished in measurements.
We can therefore have:

```
VISITED PAGE
```



Purpose

Cookies can also be distinguished according to the purpose for which they are used (Garante, 2021).
This classification is particularly important from a privacy perspective (Garante, 2021).

Under Italian regulations, the fundamental distinction is primarily between tools used for technical purposes, necessary for communication or the provision of a requested service, and tools used for other purposes, such as profiling (European Parliament & Council of the European Union, 2002; Garante, 2021).
Website interfaces and consent management systems often use the following categories:

- Technical or necessary.
- Preferences and functionality.
- Statistical or analytics.
- Marketing and profiling.
- Social services and external content.

These categories are useful for understanding the different purposes, but do not necessarily represent completely separate technical or legal categories (Garante, 2021).

Technical Cookies
Technical cookies are used to enable a service to operate or to provide a function requested by the user (European Parliament & Council of the European Union, 2002; Garante, 2021).
They can be used, for example, to:

- maintain an authenticated session.
- manage certain security functions.
- maintain the state of a shopping cart during a purchase.
- remember information needed to provide a service.

For example, after login:

```
User enters credentials
          ↓
Server authenticates the user
          ↓
creates a session
          ↓
Set-Cookie: session_id=ABC123
```

In subsequent requests:

```
Cookie: session_id=ABC123
          ↓
server recognizes the session
          ↓
user remains authenticated
```

Preference and Functionality Cookies
Some cookies are used to remember choices made by the user (Garante, 2021).
For example: lingua=it-
or: tema=scuro-

Imagine that the user selects:

```
Italian
    ↓
cookie:
lingua=it
```

On the next visit:

```
Cookie: lingua=it
       ↓
the site recognizes the preference
       ↓
interface displayed in Italian
```

These cookies therefore allow certain features of the service to be personalized without asking for the same choice again on every visit.

Their classification from a privacy perspective, however, depends on the function they actually perform: calling a cookie “functional” does not automatically determine whether or not it requires consent (Garante, 2021).

Statistical and Analytics Cookies
Cookies and other analytics tools can be used to measure how a site is used (Garante, 2021; Google, n.d.-b).

They can help collect the following information:
- Number of visits.
- Pages visited.
- Session duration.
- Interactions.
- Conversions.
- Traffic sources.

This information can help the operator understand how the site is used and improve its structure and content.

Under Italian regulations, however, analytics cookies are not automatically equivalent to technical cookies (Garante, 2021).

The Garante provides that they can be treated as technical cookies only under certain conditions relating, among other things, to the production of aggregate statistics and the minimization of users’ identifiability (Garante, 2021).

Profiling and Marketing Cookies
Cookies and other profiling tools can be used to observe users’ behavior and derive information about, for example, interests or habits (Garante, 2021).
Imagine that a browser frequently visits content relating to:
- Cameras.
- Lenses.
- Photography.

A system could associate the identifier with the value “XYZ789” with a possible interest:

```
XYZ789
   ↓
inferred interest:
photography
```

This information could be used to:
- Personalize advertising messages.
- Create audience segments.
- Personalize certain content.
- Measure advertising campaigns.
- Recognize certain behaviors over time.

In Italy, the use of cookies and other tools for profiling purposes normally requires the user's prior consent, under the conditions established by the applicable legislation and the Garante's guidance (European Parliament & Council of the European Union, 2002; Garante, 2021).

Supplementary material

Exercises

Maintaining Login Across Different Requests
A user logs in at: https://example.com/login

After successfully checking the email and password, the server creates a session associated with the user and responds:

```
Set-Cookie: session_id=7f3a9c81; Path=/; Secure; HttpOnly; SameSite=Lax
```

The user then opens: https://example.com/profilo

Answer the following questions:

- Explain why the server should not necessarily put the user's name, email and other data in the cookie.
- State what information can be stored on the server by associating it with the value 7f3a9c81.
- Conceptually write the header that the browser could send in the next request.
- Explain how the server uses session_id to recognize the user again.
- Explain why this mechanism allows state to be maintained across different requests without making HTTP inherently "stateful".
- Explain the function of the Path=/, Secure, HttpOnly and SameSite=Lax attributes.
- Explain why HttpOnly does not prevent the browser from sending the cookie in HTTP requests.
- The user logs out. Describe the two operations the application should perform: one on the server and one in the browser.
- Suggest a possible Set-Cookie header to ask the browser to delete the session cookie.
- Explain what would happen if the browser continued to send an old session_id but the server had already invalidated that session.

When Is the Cookie Sent?
A banking website sets this cookie:

```
Set-Cookie: session_id=ABC123; Path=/; Secure; HttpOnly; SameSite=Lax
```

The user is already authenticated on https://banca.example.
Consider these situations:

- A. While on banca.example, the user opens /profilo.
- B. The user is on notizie.example and clicks a normal link to https://banca.example/profilo.
- C. A page on notizie.example contains: <img src="https://banca.example/operazione">.
- D. A script on notizie.example executes: fetch("https://banca.example/operazione").

For each case:

- Determine whether the request is same-site or cross-site.
- In cross-site cases, distinguish between top-level navigation and a resource request.
- Predict whether the SameSite=Lax cookie can be attached to the request and justify your answer.
- Repeat the reasoning assuming that the cookie uses SameSite=Strict.
- Repeat the reasoning with SameSite=None.
- Explain why SameSite=None also requires Secure.
- Explain why cross-site navigation is not equivalent to loading an image across sites or making a fetch() request.
- State which configuration among Strict, Lax and None offers the most restrictive behavior.
- Explain why Lax is presented in the handout as a possible compromise between security and normal browsing.

Classifying a Website's Cookies

An online store uses these cookies:

- Cookie A, session_id=ABC123. It keeps the user authenticated during the session.
- Cookie B, lingua=it; Max-Age=2592000. It remembers the selected language.
- Cookie C, carrello=XYZ456. It stores the state of the shopping cart.
- Cookie D, advertising_id=QW789; SameSite=None; Secure. It belongs to an advertising service embedded in the site and can be used to recognize the same browser on different sites.
- Cookie E, analytics_id=123456. It is used to collect statistics on the use of the site.

Analyze each cookie according to the different taxonomies in the handout.

- Classify it, where possible, as a session or persistent cookie.
- Classify it as first-party or third-party according to its context of use.
- Identify its possible purpose as technical, preference/functionality, analytics or profiling/marketing.
- Explain why these three classifications are independent of one another.
- Explain why a persistent cookie is not automatically a tracking cookie.
- Explain why a first-party cookie is not automatically a technical cookie.
- Explain how Cookie D could enable cross-site tracking if the same advertising service were embedded in different sites.
- Explain why the mere fact that a page loads a JavaScript script from an external domain does not necessarily mean that the cookie created is third-party.
- Using the Google Analytics example described in the handout, explain how a script downloaded from an external domain can run in the context of the site being visited and use a cookie associated with that site.
- Conclude by explaining why truly understanding a cookie requires more than knowing its name: its lifetime, context of use, attributes and actual purpose must be considered.

Bibliography

Barth, A. (2011). *HTTP state management mechanism* (RFC 6265). RFC Editor. https://www.rfc-editor.org/rfc/rfc6265.html

Bingler, S., West, M., & Wilander, J. (2025). *Cookies: HTTP state management mechanism* (Internet-Draft draft-ietf-httpbis-rfc6265bis-22). Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/

European Parliament & Council of the European Union. (2002). *Directive 2002/58/EC concerning the processing of personal data and the protection of privacy in the electronic communications sector* (consolidated text as of December 19, 2009). EUR-Lex. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:02002L0058-20091219

Garante per la protezione dei dati personali. (2021). *Linee guida cookie e altri strumenti di tracciamento*. https://www.garanteprivacy.it/home/docweb/-/docweb-display/docweb/9677876

Google. (n.d.-a). *Configuration: Google Analytics*. Retrieved September 2, 2026, from https://developers.google.com/analytics/devguides/collection/ga4/reference/config

Google. (n.d.-b). *[GA4] Cookie usage on websites*. Retrieved September 2, 2026, from https://support.google.com/analytics/answer/11397207?hl=en

OWASP Foundation. (n.d.). *Session Management Cheat Sheet*. Retrieved September 2, 2026, from https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Further reading

- David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy, HTTP: The Definitive Guide, O'Reilly Media, 2002 — Probably the most directly relevant reference for understanding the fundamental cookie mechanism. Chapter 11 examines client identification, cookies, session tracking, cookie storage, domain and path restrictions, and the relationship between cookies and HTTP headers. It is particularly useful for understanding how Set-Cookie and Cookie allow applications to associate successive requests with previously stored state. Because the book predates modern attributes such as SameSite, it should be supplemented with current specifications.
- Andrew Hoffman, Web Application Security: Exploitation and Countermeasures for Modern Web Applications, 2nd ed., O'Reilly Media, 2024 — The strongest technical complement for cookie security and browser protection mechanisms. It examines authentication, authorization, browser security, cross-origin restrictions, and secure application configuration. Chapter 22 explicitly covers creating, securing, and testing cookies. It is particularly relevant to understanding how cookie attributes and browser security policies help protect authenticated sessions and reduce exposure to attacks.
- Malcolm McDonald, Web Security for Developers, No Starch Press, 2020 — An accessible and practical introduction to session security, authentication, cross-site request forgery, and other common web vulnerabilities. It explains how attackers exploit weaknesses in web applications and how developers can implement appropriate protections. It is especially useful for understanding why session

identifiers must be protected, why authentication state should be validated on the server, and why cookie configuration matters for application security.

- Markus Vollmert, Google Analytics 4: The Practical Guide, Rheinwerk Computing/Packt Publishing, 2026 — The most directly relevant book for the handout's discussion of analytics cookies, tracking identifiers, third-party services, and consent management. It explains how Google Analytics 4 collects information through tags and events, how tracking systems are implemented, and how privacy requirements affect data collection. It includes dedicated sections on cookies, browser privacy, Google Tag Manager, and Consent Mode. It is particularly useful for understanding the practical example involving an externally loaded Google Analytics script and a first-party cookie. Packt
- Nishant Bhajaria, Data Privacy: A Runbook for Engineers, Manning Publications, 2022 — The most appropriate complement for the privacy and data protection perspective. It examines personal data, privacy risks, data classification, governance, regulatory compliance, and privacy engineering. Although it is not specifically a book about HTTP cookies, it provides a broader framework for understanding the responsibilities involved when websites collect, store, associate, and use information about visitors. It is particularly relevant to the handout's distinction between technically necessary functions and processing activities involving analytics, profiling, or marketing.

Editorial notes

In producing these materials, I have used generative artificial intelligence tools to support the writing process, particularly to improve the form of the text, reorganize content, refine wording, and speed up certain editorial tasks.

As I work independently on the production of these materials, I try to automate any activities that can reasonably be automated, so that I can devote more time to research, design, and content development.

However, artificial intelligence does not determine the content of this work: the choice of topics, structure, ideas, interpretations, examples, and teaching approach are developed by me. AI is therefore used primarily as a tool to support production and formal revision, while authorship and responsibility for the design and content remain mine.

Unless otherwise indicated, this material is distributed under the Creative Commons Attribution–NonCommercial–ShareAlike 4.0 International license (CC BY-NC-SA 4.0).

You are therefore free to share, redistribute, adapt, and create derivative works based on this material, provided that appropriate credit is given, the material is not used for commercial purposes, and any modified or derivative versions are distributed under the same license.

