Cross-App Security II

HTTP Cookies

HTTP Cookies

A browser-managed data storage and transmission mechanism where key-value pairs are associated with web applications based on defined scope

HTTP Cookies

Defined in RFC 6265: HTTP State Management Mechanism

> “Although cookies have many historical infelicities that degrade their security and privacy, the Cookie and Set-Cookie header fields are widely used on the Internet.”

https://www.rfc-editor.org/rfc/rfc6265

Key Concepts

Cookies != Sessions

HTTP Cookies are a mechanism for storing and transmitting data often used as part of a session management system

Client-Set Cookies

Cookies are often defined as being set by a server, but JavaScript can also create cookies

Ambient/Implicit Authority

Cookies are automatically sent by the browser according to scope

Scope

The set of rules that determine what URLs cookies are sent to

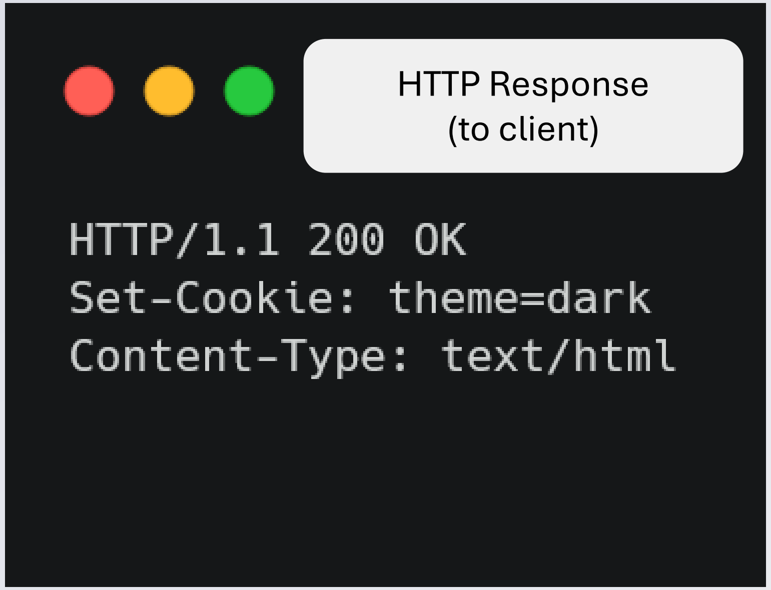
Cookies Are Stateful

While HTTP is stateless, cookies are one mechanism to relate distinct requests

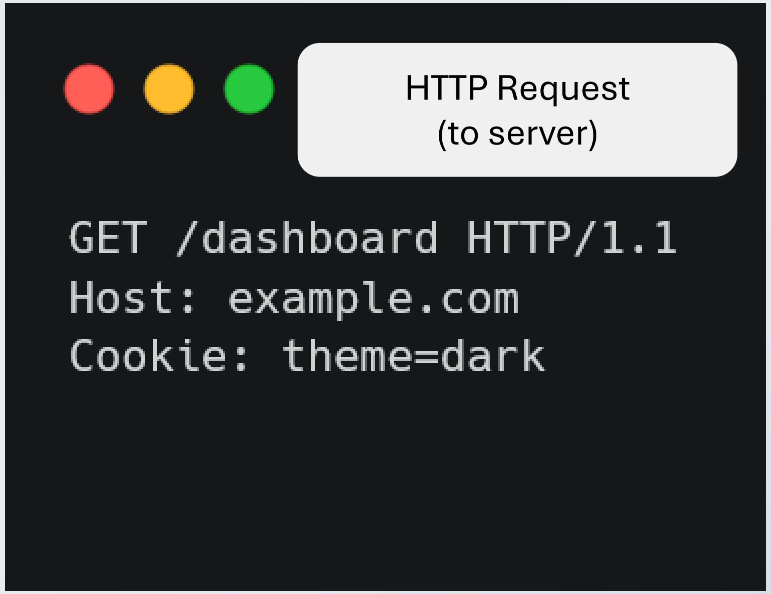
Common Cookie Use Cases

- Session management (both authenticated and unauthenticated)
- Personalization/preferences
- Security mechanisms (anti-CSRF)
- Load balancing
- Trackers and Analytics

Server-Side Cookie Management

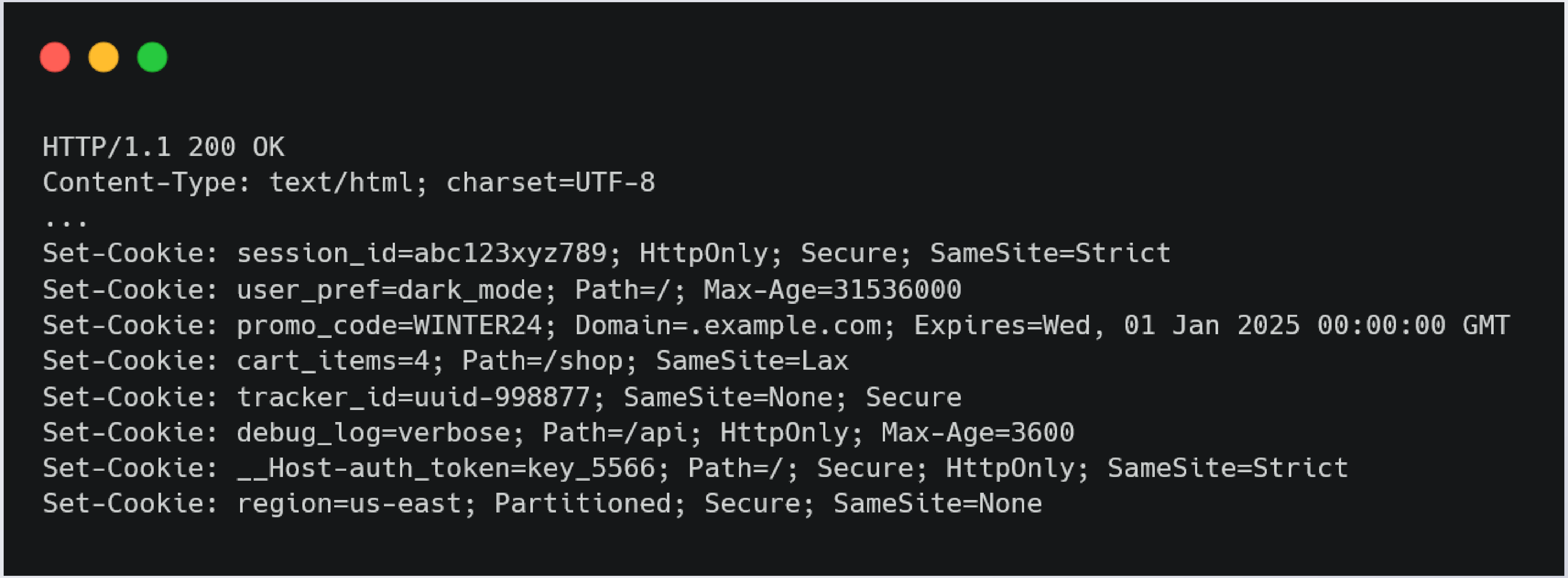


```
HTTP/1.1 200 OK
Set-Cookie: theme=dark
Content-Type: text/html
```



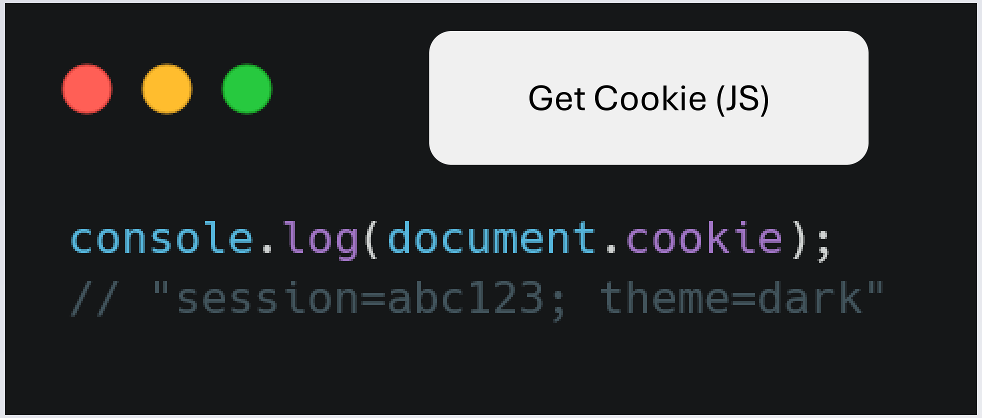
```
GET /dashboard HTTP/1.1
Host: example.com
Cookie: theme=dark
```

Server-Side Cookie Management

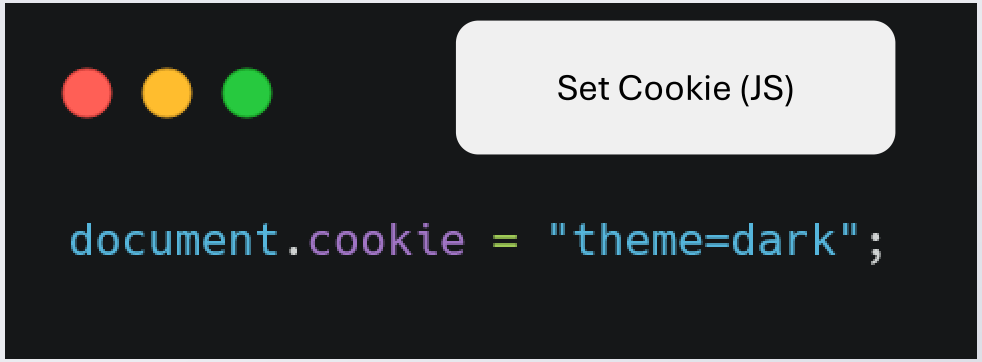


```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
...
Set-Cookie: session_id=abc123xyz789; HttpOnly; Secure; SameSite=Strict
Set-Cookie: user_pref=dark_mode; Path=/; Max-Age=31536000
Set-Cookie: promo_code=WINTER24; Domain=.example.com; Expires=Wed, 01 Jan 2025 00:00:00 GMT
Set-Cookie: cart_items=4; Path=/shop; SameSite=Lax
Set-Cookie: tracker_id=uuid-998877; SameSite=None; Secure
Set-Cookie: debug_log=verbose; Path=/api; HttpOnly; Max-Age=3600
Set-Cookie: __Host-auth_token=key_5566; Path=/; Secure; HttpOnly; SameSite=Strict
Set-Cookie: region=us-east; Partitioned; Secure; SameSite=None
```

Client-Side Cookie Management



```
console.log(document.cookie);
// "session=abc123; theme=dark"
```



```
document.cookie = "theme=dark";
```

Cookie Attributes (Flags)

HTTPOnly

- Can only be set by the server, prevent JS access (blocks *document.cookie*)

Secure

- When set, cookies will only be sent over HTTPS connections

SameSite

- Further restricts when cookies are sent if the request is cross-site

Domain

- Sets the scope according to a chosen domain

Path

- Sets the scope according to a chosen path

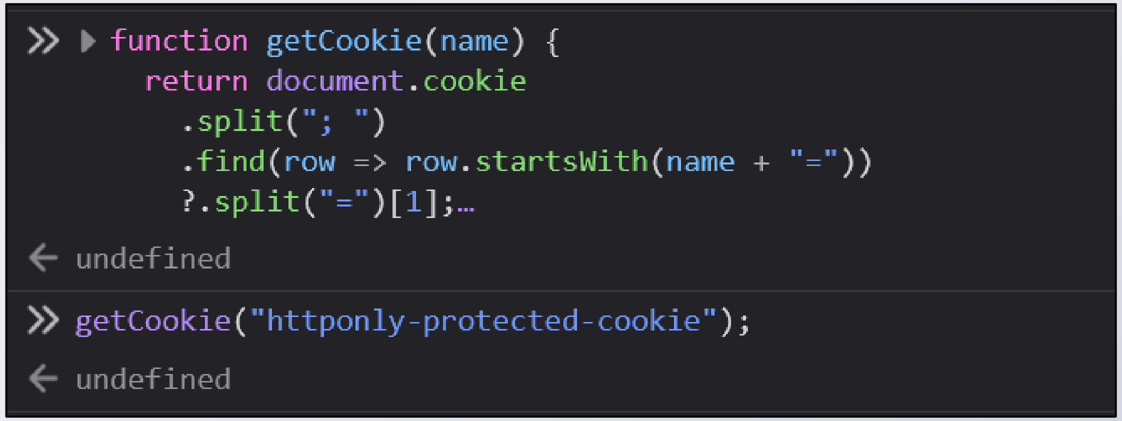
Cookie Attributes

HTTPOnly

Can only be set by the server, prevent JS access (blocks *document.cookie*)

Secure

When set, cookies will only be sent over HTTPS connections



```
>> ▸ function getCookie(name) {
       return document.cookie
         .split("; ")
         .find(row => row.startsWith(name + "="))
         ?.split("=")[1];…
← undefined
>> getCookie("httponly-protected-cookie");
← undefined
```

DBG App Test Finding
Session Cookie Without HttpOnly Flag Set

DBG App Test Finding
Session Cookie Without Secure Flag Set

Aside: Canonical Domains

- Should *ex.com* be equivalent to *www.ex.com*?
- These are different Origins AND have implications for cookie handling!
- The subdomain *www* emerged to denote a web service as distinct from other services hosted by *apex* domain (eTLD+1 / root domain)
- Best practice: choose and enforce *www* as the *canonical* subdomain for the app and redirect all traffic to it

DBG App Test Finding

Canonical World Wide Web Subdomain Policy Not Implemented

Domain and Path Attributes

- These can broaden OR restrict cookie scope
  - Implication: applications share security context with subdomains and sub-paths
  - “Cookie Tossing” – subdomain sets cookies on its parent via *domain* attr.
- Domain value restrictions:
  - Must be a suffix of the setting host
  - Cannot set against an eTLD (such as *.com*)

DBG App Test Finding

Session Cookie Scoped to Parent Domain/Path

DBG App Test Finding

Shared Application Site or Origin Scope

Traditional Cookie Scope Examples

Cookie Set By	Attributes	Cookie May Be Sent To (non-exhaustive)	Cookie NOT Sent To (non-exhaustive)

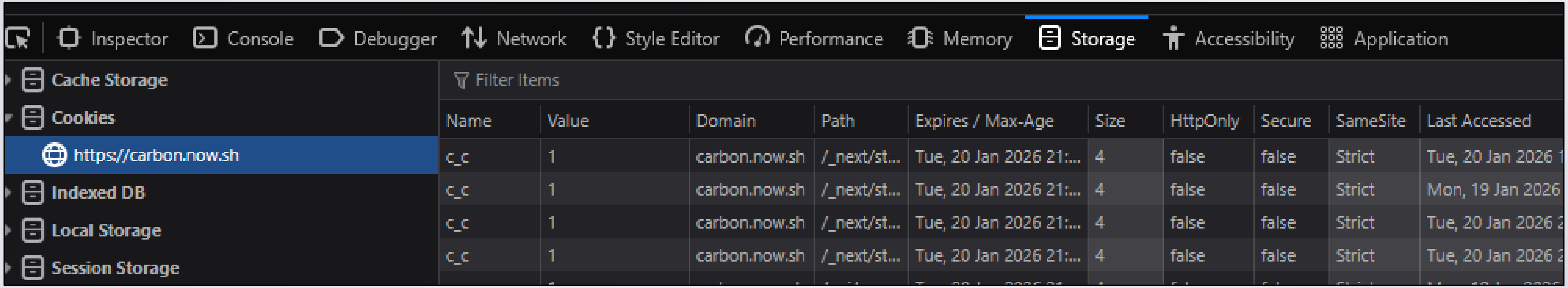
https://www.ex.com	<none>
	
	(implicit:
	Path=/
	Domain=www.ex.com)
		https://www.ex.com
		http://www.ex.com
		http://www.ex.com:8080
		https://www.ex.com/path1
		https://www.ex.com/path2
			https://www.ex2.com
			https://api.ex.com
			https://ex.com

https://ex.com	<none>
	
	(implicit:
	Path=/
	Domain=ex.com)
		https://ex.com
		http://ex.com
		https://www.ex.com
		https://api.ex.com
			https://www.ex2.com
			https://ex.org

https://www.ex.com	Domain=ex.com
	
	(implicit:
	Path=/)
		https://www.ex.com
		https://ex.com
		https://api.ex.com
			https://www.ex2.com
			https://ex.org

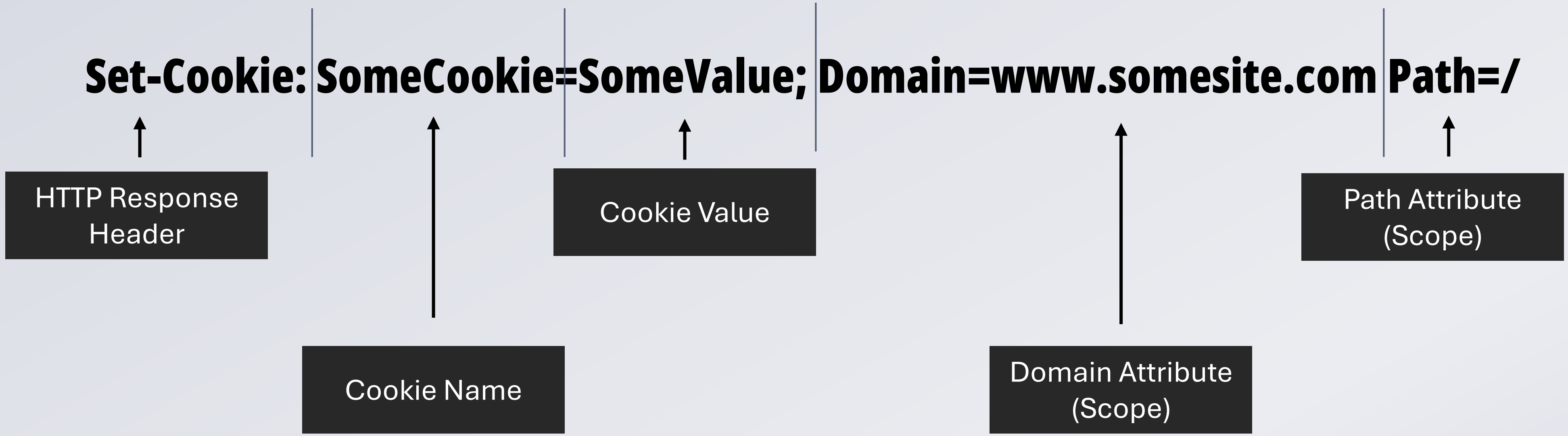
https://www.ex.com/app/profile	<none>
	
	(implicit:
	Path=/app/
	Domain=www.ex.com)
		https://www.ex.com/app/update
		https://www.ex.com/app/update
			https://www.ex.com/app2/profile
			https://ex.com/app/update
			https://sub.ex.com/app/update

Cookie Attributes: Browser Dev Tools



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
c_c	1	carbon.now.sh	/_next/st...	Tue, 20 Jan 2026 21:...	4	false	false	Strict	Tue, 20 Jan 2026 1
c_c	1	carbon.now.sh	/_next/st...	Tue, 20 Jan 2026 21:...	4	false	false	Strict	Mon, 19 Jan 2026
c_c	1	carbon.now.sh	/_next/st...	Tue, 20 Jan 2026 21:...	4	false	false	Strict	Tue, 20 Jan 2026 2
c_c	1	carbon.now.sh	/_next/st...	Tue, 20 Jan 2026 21:...	4	false	false	Strict	Tue, 20 Jan 2026 2

Cookie Identity and Uniqueness



Set-Cookie: SomeCookie=SomeValue; Domain=www.somesite.com Path=/

HTTP Response Header

Cookie Name

Cookie Value

Domain Attribute (Scope)

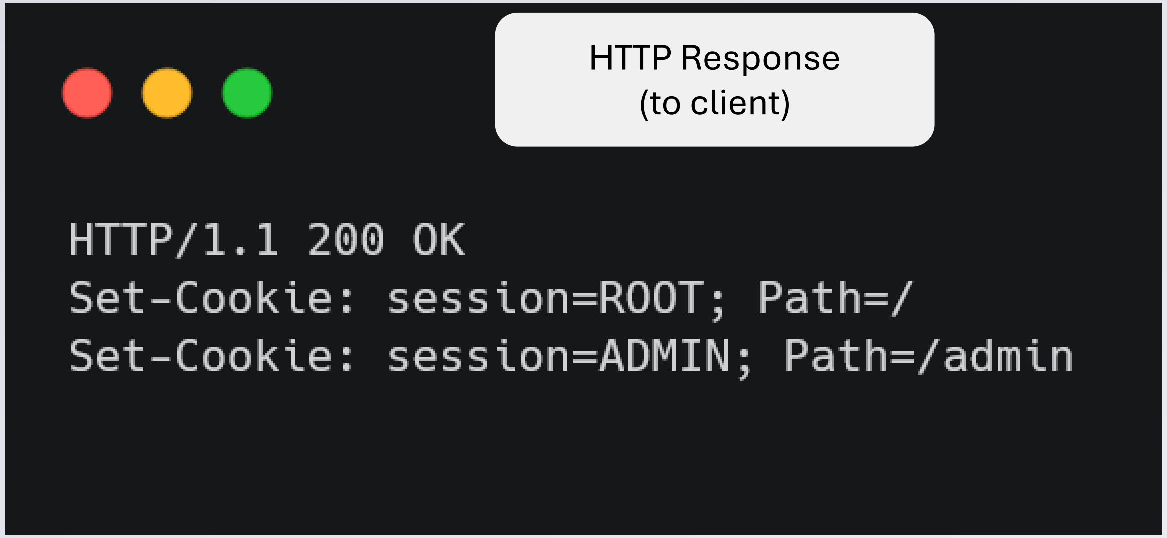
Path Attribute (Scope)

>> Cookies are unique by: Name + Domain + Path <<

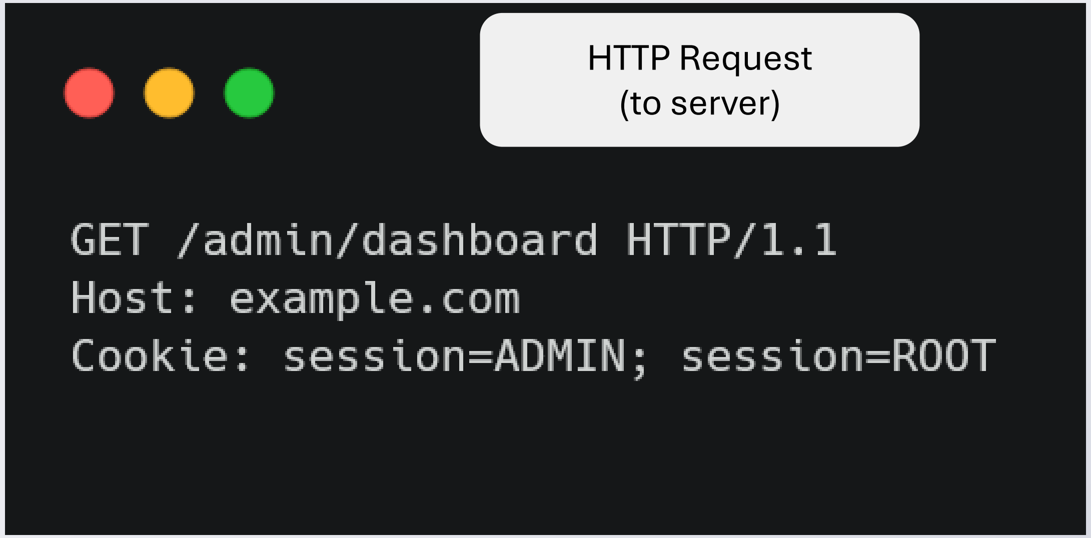
- ***Default domain (attribute absent): cookie is set to exact host that set it***
  - ***NOT sent to different subdomains or higher domains***
- ***Default path (attribute absent): cookie is set to the path truncated to the last '/'***
  - ***For ex, cookies set on '/account' will apply to '/'***

Cookie Identity and Uniqueness

(Cookie Shadowing)



```
HTTP/1.1 200 OK
Set-Cookie: session=ROOT; Path=/
Set-Cookie: session=ADMIN; Path=/admin
```



```
GET /admin/dashboard HTTP/1.1
Host: example.com
Cookie: session=ADMIN; session=ROOT
```

Cookie ordering by browsers:

1. Path length, long to short
2. Last updated, least recent to most

See: https://blog.ankursundara.com/cookie-bugs/

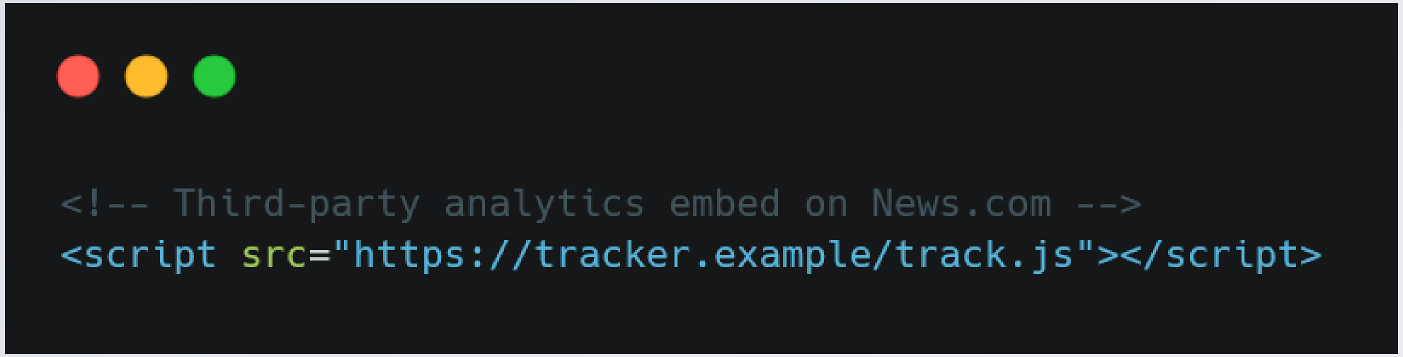
When and Where are Cookies Sent?

- Cookies *mostly* follow Same-Site scope (they are shared cross-origin)
- Cookie identity and scope is determined by *name*, *domain*, and *path*
- The *secure* attribute ensures cookies are sent over HTTPS only
- The *SameSite* attribute restricts scope depending on the source and destination and *how* the request was initiated
- Browser-specific behaviors may also impact when “third-party” (effectively cross-site) cookies are sent
    - ex: Firefox implements Enhanced Tracking Protection that includes Total Cookie Protection (behavior depends on configuration)
    - Other exs: Safari ITP, Chrome CHIPS (see also *Partitioned* attribute)

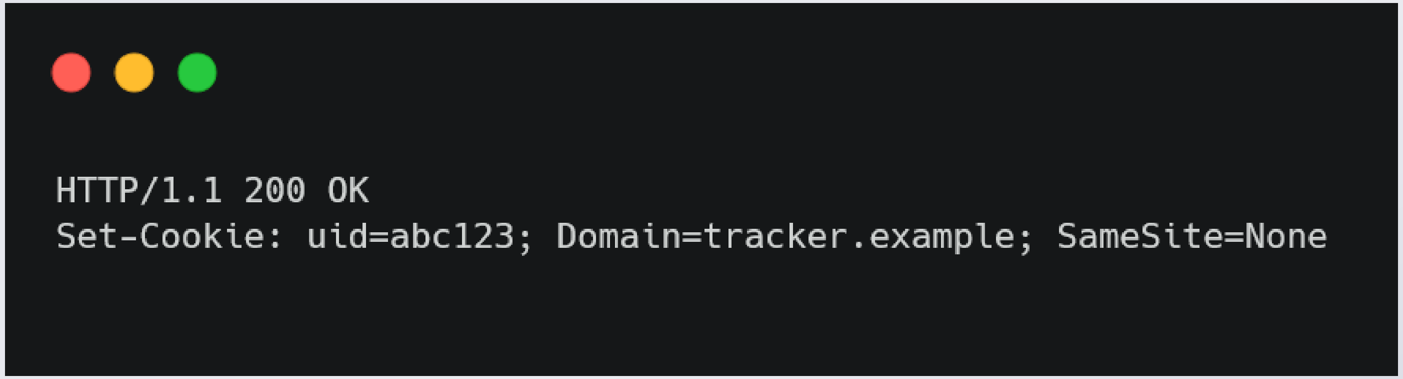
Understanding Tracking Cookies

Top-level site:
news.com

Cookie set by:
tracker.example

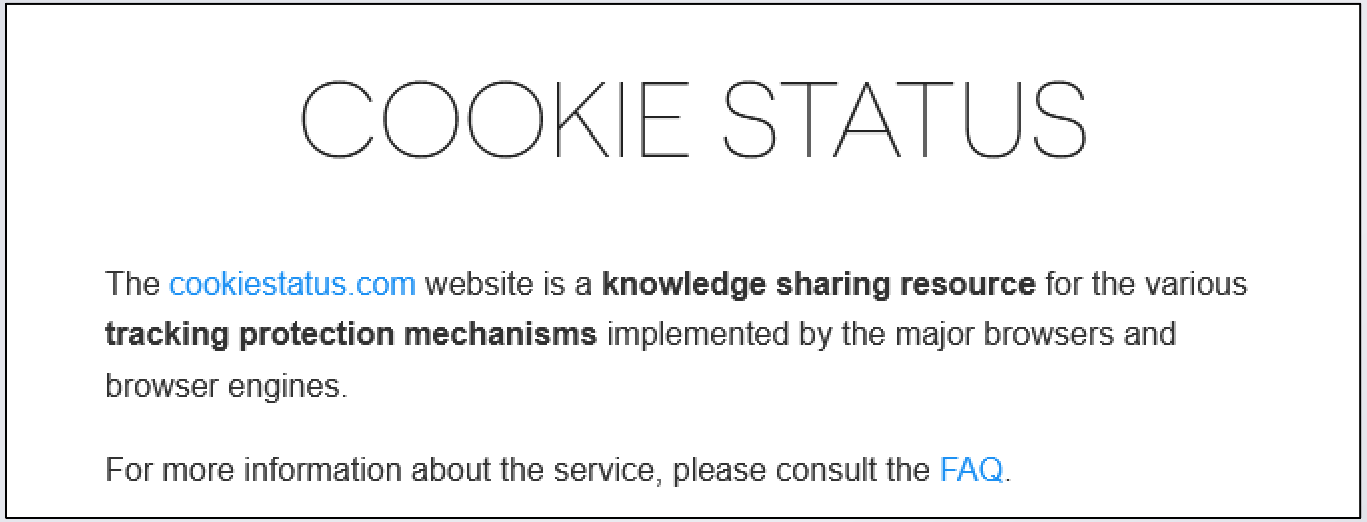


```
<!-- Third-party analytics embed on News.com -->
<script src="https://tracker.example/track.js"></script>
```



```
HTTP/1.1 200 OK
Set-Cookie: uid=abc123; Domain=tracker.example; SameSite=None
```

Understanding Tracking Protections



COOKIE STATUS

The cookiestatus.com website is a **knowledge sharing resource** for the various **tracking protection mechanisms** implemented by the major browsers and browser engines.

For more information about the service, please consult the FAQ.

https://www.cookiestatus.com/

Cookies Having Independent Partitioned State (CHIPS)

Partitioned Cookies

Cookies with *Partitioned* attribute that live in isolated cookie jar per top-level site context

Purpose: Allow limited third-party state without cross-site tracking

Example:

- ***shop.ex*** **loads** ***payments.ex*** **resource, setting cookies for payments.ex domain**
- ***news.ex*** **loads** ***payments.ex*** **resource, setting cookies for payments.ex domain**
- **Both sets of** ***payments.ex*** **cookies are isolated!**

Only recently implemented across browsers as explicit attr. (Dec. 2025)

DBG App Test Finding
Third-Party Cookie Without Partitioned State

Cookie Prefixes

Less well-known and further restrict cookie creation rules to guarantee integrity (protects against maliciously set cookies)

__Secure-

Any cookie with this prefix must be set with the *Secure* attribute and from a URI considered secure by the user agent

__Host-

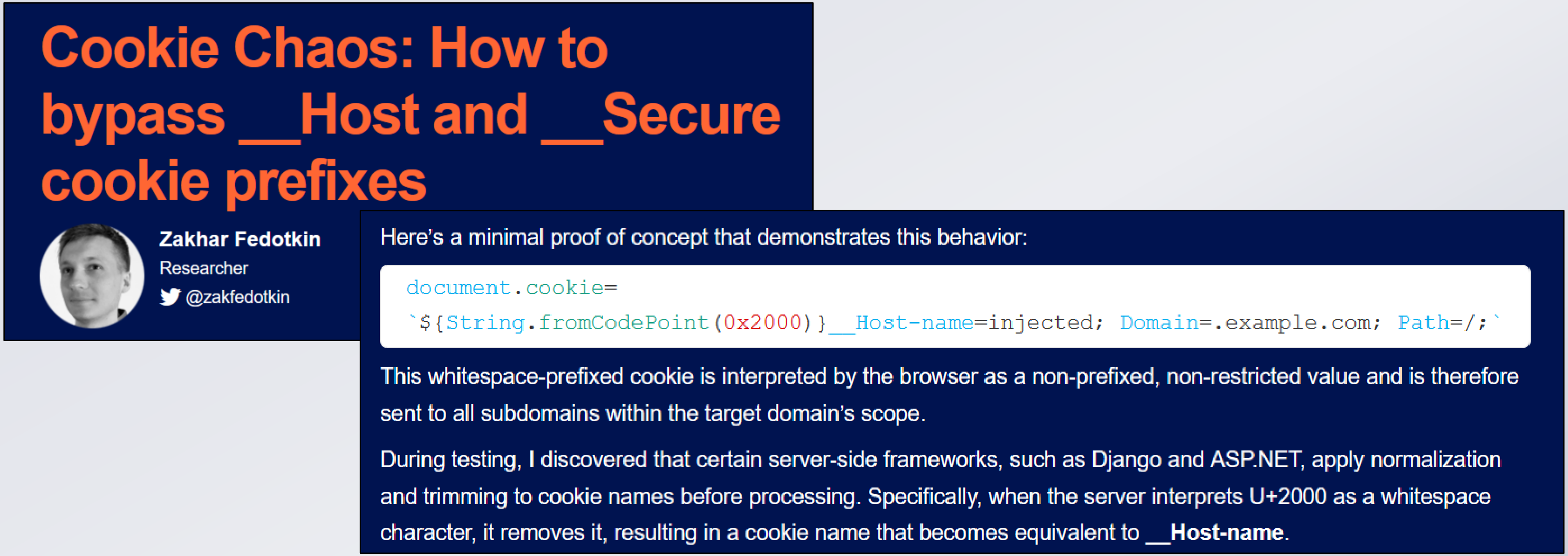
In addition to *__Secure-* requirements, the cookie is sent only to the host that set the cookie, must not include a *Domain* attribute, and *Path* must be '/'

DBG App Test Finding

Session Cookie With Missing or Misconfigured Prefix

Cookie Prefixes

Like many cookie features, security relies on coordination of browser AND server!



Cookie Chaos: How to bypass __Host and __Secure cookie prefixes

Zakhar Fedotkin
Researcher
@zakfedotkin

Here's a minimal proof of concept that demonstrates this behavior:

```
document.cookie=
`${String.fromCodePoint(0x2000)}__Host-name=injected; Domain=.example.com; Path=/;`
```

This whitespace-prefixed cookie is interpreted by the browser as a non-prefixed, non-restricted value and is therefore sent to all subdomains within the target domain's scope.

During testing, I discovered that certain server-side frameworks, such as Django and ASP.NET, apply normalization and trimming to cookie names before processing. Specifically, when the server interprets U+2000 as a whitespace character, it removes it, resulting in a cookie name that becomes equivalent to **__Host-name**.

https://portswigger.net/research/cookie-chaos-how-to-bypass-host-and-secure-cookie-prefixes

Cookie Storage and Lifetime

Max-Age

Time in seconds after which the cookie should be deleted and not sent (takes precedence over *Expires*)

Expires

Date after which the cookie should be deleted and not sent

Important notes:

- **With no date/age set, the cookie should be removed when the browser is closed (historically referred to as a *session* cookie)**
  - **Not *strictly* followed by browsers**
- **With either *Expires* or *Max-Age* set, the cookie should persist restarts (historically referred to as a *persistent* cookie)**

Cookie Storage and Lifetime

Browser Cookie Storage

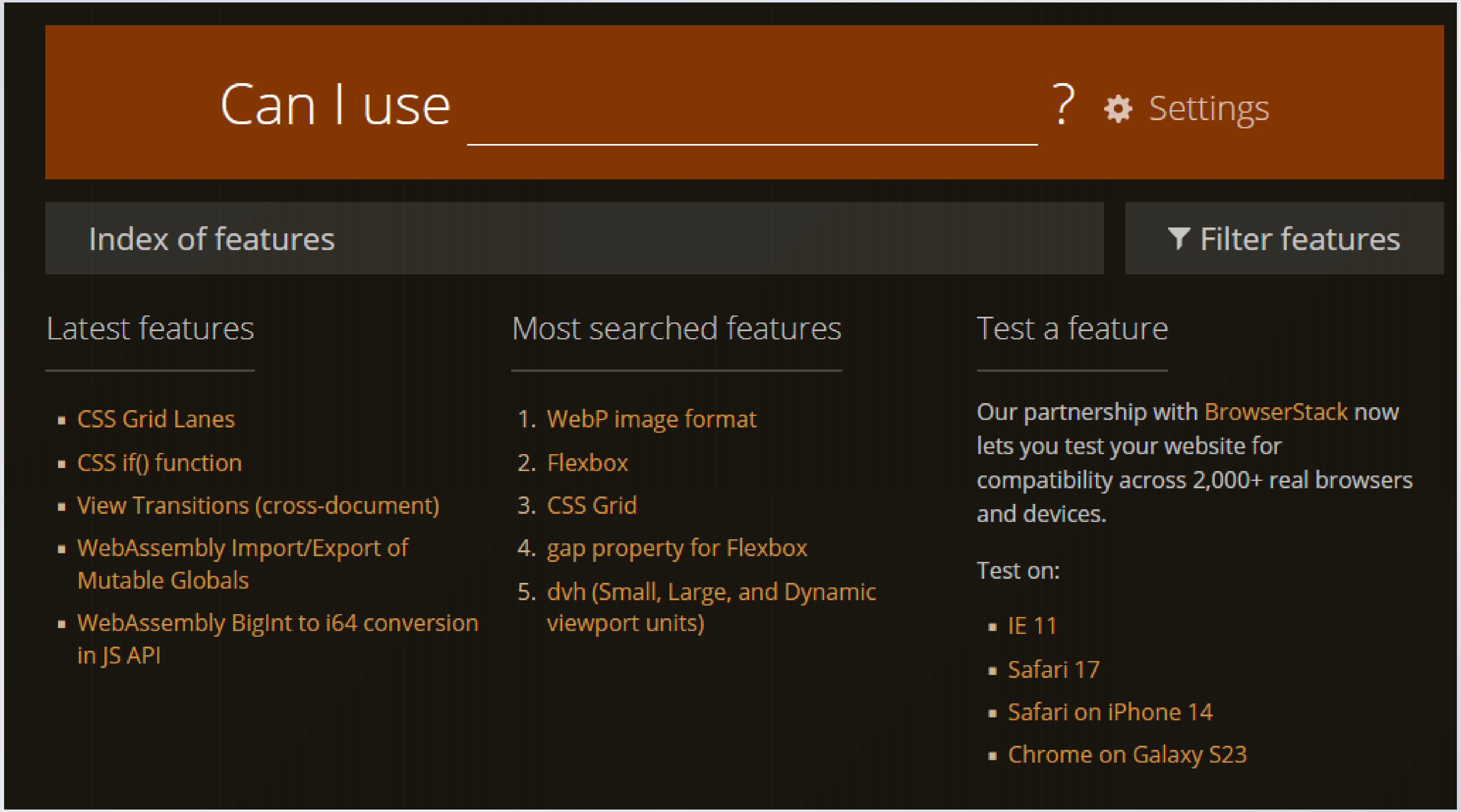
Modern browsers utilize a complex set of storage techniques for cookies

(Approximate) Storage Limits

- **Cookie size: 4096 bytes**
- **Cookies per domain: ~150-180**
- **Total cookies: ~3000+**

- **Numerous attacks rely on length/size limitations and behavior!**

Aside: Can I Use ___?



Can I use ___ ? ⚙ Settings

Index of features

Filter features

Latest features

- CSS Grid Lanes
- CSS if() function
- View Transitions (cross-document)
- WebAssembly Import/Export of Mutable Globals
- WebAssembly BigInt to i64 conversion in JS API

Most searched features

1. WebP image format
2. Flexbox
3. CSS Grid
4. gap property for Flexbox
5. dvh (Small, Large, and Dynamic viewport units)

Test a feature

Our partnership with BrowserStack now lets you test your website for compatibility across 2,000+ real browsers and devices.

Test on:

- IE 11
- Safari 17
- Safari on iPhone 14
- Chrome on Galaxy S23

https://caniuse.com/

SameSite Attribute

Restricts when cookies are sent based on the context of the request

SameSite=None

- **No cross-site restriction is applied**

SameSite=Lax

- **Cookies are only sent same-site OR when the following are BOTH true:**
    - **The request is top-level navigation (clicking a link or navigating to a URL)**
    - **The request uses a “safe” (read-only) method (GET, HEAD, OPTIONS)**

SameSite=Strict

- **Cookies are not sent cross-site**

DBG App Test Finding

Session Cookie Without Cross-Site Restriction

SameSite Attribute: Browser Behavior

When *SameSite* is <u>absent</u>, browsers apply default protections

(likely to vary over time, so always test!)

Firefox

- **Sets *SameSite=Lax***

Chrome/Chromium

- **Sets *SameSite=Lax* but only AFTER 2 minutes from the time the cookie is set**
- **"Such cookies will also be sent with non-idempotent (e.g. POST) top-level cross-site requests … Support for this intervention ("Lax + POST") will be removed in the future."**

Safari

- **Does not set Lax-by-default, but has third-party cookie blocking (ITP)**

Common Cookie Abuse

Cookie Overflow

Forcing browser to evict important cookies by malicious setting (via, for ex, controlled subdomain)

Cookie Bomb

These attacks target the servers processing requests with cookies by using JS to set very large cookie values

Cookie Tossing

Setting cookies via XSS or controlled subdomain to collide with target application cookies

Session Fixation

Cause a user to authenticate with an attacker-set cookie value

DBG App Test Finding
Session Fixation Potential

Common Cookie Abuse: CSRF

To be covered in a CSRF Session…

OWASP ASVS 5.0

V3 Web Frontend Security: V3.3 Cookie Setup

- 3.3.1: Set *Secure* and set __*Secure-* prefix if __*Host-* is not implemented
- 3.3.2: Restrict scope using *SameSite*
- 3.3.3: Unless designed to be shared, set __*Host-* prefix
- 3.3.4: Use *HttPOnly* if possible
- 3.3.5: Ensure cookies (names and values) are not too long

https://github.com/OWASP/ASVS/blob/master/5.0/en/0x12-V3-Web-Frontend-Security.md

Security Testing Considerations

- What is the purpose of cookies assigned by the application?
- How are cookies scoped/protected? When will they be sent?
- Do multiple applications share Site scope?
- Can browser differences and quirks be abused?
- Are cookies handled in an unconventional way?