FREE CHEAT SHEET · WEB AUTH · 2026

Stop storing tokens wrong.

Where auth tokens should live, and the copy-paste config to put them there.

> **The problem:** anything in `localStorage` or `sessionStorage` can be read by any JavaScript running on your page. One cross-site scripting (XSS) bug, whether in your own code, a dependency or a third-party widget, and an attacker can copy every user's token and sign in as them. XSS is the #1 weakness in MITRE's 2025 CWE Top 25.

Inside:

① Where each kind of data should live (one table)
② The cookie settings that block token theft
③ Copy-paste code for Express and Next.js
④ The CSRF trade-off, and how to handle it
⑤ A 10-minute audit for your app



Afeez Badmos
@afeez_sdbytes · SoftwareDevBytes × GMSS

Where should it live? ①

DATA	STORE IT IN	WHY
Session / refresh token	**HttpOnly cookie**	JavaScript can't read it, so injected scripts can't steal it. The browser sends it automatically with requests to your site.
Short-lived access token (SPA calling an API)	**Memory (a JS variable)**	Disappears on reload, so you get a new one from the refresh-token cookie. Keep it short-lived (for example, 5–15 minutes).
Any token	**localStorage**	Readable by any script on the page, and it **never expires**, even after the tab or browser is closed.
Any token	**sessionStorage**	Also readable by any script. It's cleared when the tab closes, which doesn't help if XSS steals it while the tab is open.
Theme, language, UI prefs	**localStorage**	Not sensitive. Persisting it across visits is exactly what localStorage is for.
Non-sensitive per-tab state (form drafts, wizard step)	**sessionStorage**	Lasts per tab and is cleared when the tab closes.

Cookies bring CSRF. Handle it. ④

Because the browser attaches cookies automatically, another site can try to trigger requests that carry your user's cookie. That's cross-site request forgery (CSRF), #3 in the 2025 CWE Top 25. Cover it with:

- [ ] **SameSite=Lax** (or **Strict**). With Lax, cross-site requests only get the cookie on top-level GET navigations, not on cross-site `fetch()` or POST/PUT/DELETE.
- [ ] **A CSRF token** on state-changing requests (forms, POST/PUT/DELETE), or a check of the `Origin` header on the server.
- [ ] **Never change state on GET.** GET routes should only read.

Be honest about the limits: HttpOnly stops token *theft*, not XSS itself. While the page is open, an injected script can still send requests as the user. You still need the XSS hardening in ⑤.

10-minute audit ⑤

- [ ] Search your code for `localStorage.setItem` / `sessionStorage.setItem` containing "token", "jwt" or "auth", and move those tokens to cookies.
- [ ] In DevTools → Application → Cookies, check that the session cookie shows ✓ HttpOnly and ✓ Secure, and that SameSite is set.
- [ ] Add a Content-Security-Policy (no inline scripts, `script-src 'self'`) and avoid `innerHTML` / `dangerouslySetInnerHTML` with user input (sanitize it if you must).
- [ ] Make logout clear the cookie **and** revoke the session on the server.

The cookie settings ②

```
Set-Cookie: __Host-session=<token>; HttpOnly; Secure; SameSite=Lax; Path=/; Max-Age=900
```

`HttpOnly`	JavaScript can't read the cookie (`document.cookie` won't show it). This is what blocks token theft by XSS.
`Secure`	The cookie is only sent over HTTPS.
`SameSite=Lax`	Cuts off most cross-site sending (see the CSRF section). `None` requires `Secure` .
`__Host-` prefix	The browser only accepts the cookie if it has `Secure` , `Path=/` and no `Domain` , which locks it to your exact host.
`Max-Age`	Keep sessions short and rotate refresh tokens on each use.

Copy-paste code ③

```
// Express — after a successful login
res.cookie("__Host-session", token, {
  httpOnly: true, secure: true, sameSite: "lax", path: "/",
  maxAge: 15 * 60 * 1000,            // milliseconds in Express
});
```

```
// Next.js (App Router) — Route Handler or Server Action
import { cookies } from "next/headers";
(await cookies()).set("__Host-session", token, {
  httpOnly: true, secure: true, sameSite: "lax", path: "/",
  maxAge: 60 * 15,                   // seconds in Next.js
});
```

```
// Front end: no token handling at all. The cookie rides along.
await fetch("/api/me");    // cross-origin API? add { credentials: "include" }
```

WANT A SECOND PAIR OF EYES?

Get your app's auth flow reviewed.

Global McLien Software Solutions audits login, token storage, cookies, CSP and CSRF in web and mobile apps, then fixes what it finds.

globalmcliensoftwaresolutions.com →

Sources: MDN, Set-Cookie header: developer.mozilla.org · MITRE 2025 CWE Top 25: cwe.mitre.org · OWASP HTML5 Security Cheat Sheet (local storage): owasp.org · OWASP CSRF Prevention Cheat Sheet: owasp.org