BUILD SAFELY · BUILDER · 15 MIN READ

Web security basics: access checks, XSS, CORS and cookies

Most attacks on small apps use a few old tricks. Check every request on the server, keep what people type as text, keep data out of your commands, know what CORS really does, and set sign-in cookies safely.

Pass the check (4 of 5) online to earn the “Web Guardian” badge:
https://digitalbuilders757.com/university/study/web-security-basics

Never trust the browser

Everything that arrives from a browser can be changed by the person using it: form fields, the URL, headers, cookies, even a price in a hidden input. DevTools or a one-line script can send anything.

So the server decides: who is this, are they allowed to do this, and does the data make sense? Checks in the browser are a convenience, never the lock.

Check access on every request

- Broken access control is number one in the OWASP Top 10, as it was in 2021: apps that let people see or change things that aren't theirs.
- Signing in answers who someone is. Each request must also ask whether they may do this: /api/orders/42 must check that order 42 belongs to the person asking.
- Hiding a button isn't protection. Anyone can call the URL behind it.
- Admin pages need a server-side check that the person is an admin, not a secret URL.
- In Supabase, row level security runs this check inside the database for every query.

Cross-site scripting (XSS): text that becomes code

XSS happens when something a person typed goes into a page as HTML, and the browser runs it as code for whoever views it. That code acts as the victim: it can read the page, call your API and change their settings.

- React shows {text} as plain text, so the default is safe.
- The risky doors are innerHTML, outerHTML, document.write and React's dangerouslySetInnerHTML. Use textContent for text.

- If you must show HTML that people wrote, such as rich comments, clean it with a sanitiser like DOMPurify first.
- Links can carry code too. Only accept http: and https: links from people, never javascript:.

Injection: keep data out of commands

- Never glue what people type into a SQL string. Use parameters, or a client such as supabase-js that sends values separately.
- The same goes for shell commands and file paths: a name like ../../secrets can walk out of the folder you meant.
- AI prompts are commands too. Text from a page or an email can carry instructions, so treat it as data and limit what your AI feature can do.

CORS is a browser rule, not a lock

- The same-origin policy stops a page on one site from reading responses from another. CORS headers are how a server says which other sites may read it.
- A CORS error comes from the browser: the request often reached the server, but the browser wouldn't hand your page the answer.
- CORS doesn't protect your API. Scripts, curl and other servers ignore it, so your API still needs sign-in and access checks.
- The usual fix for a third-party API is to call it from your own server route, which also keeps the key secret. Don't send Access-Control-Allow-Origin: * with private data.

Cookies and forged requests

- Sign-in cookies should be HttpOnly (JavaScript can't read them), Secure (sent over HTTPS only) and SameSite=Lax or Strict.
- Cross-site request forgery (CSRF) tricks someone's browser into sending a request to your site with their cookies. SameSite cookies stop most of it.
- Change data with POST, PUT, PATCH or DELETE, never GET, so a link or an image can't change anything.
- Next.js Server Actions turn away requests whose Origin doesn't match your site. Route handlers you write don't do that for you.

Security bugs to spot in AI-written code

- innerHTML or dangerouslySetInnerHTML with anything a person typed.
- A route that reads an id from the URL and returns the row without checking who owns it.
- SQL built from template strings with user input.
- Access-Control-Allow-Origin: * on routes that return private data.
- Admin checks only in the page, not on the server.
- Error messages that show visitors stack traces or database details.

Key terms

Broken access control
When an app lets people see or change things that aren't theirs, usually because the server never checked. Number one in the OWASP Top 10.

Cross-site scripting (XSS)
When text someone typed goes into a page as HTML and runs as code in other people's browsers.

Sanitiser (HTML)
A tool, such as DOMPurify, that strips anything that could run as code out of HTML before you show it.

SQL injection
When text someone typed is glued into a SQL query and changes what it does. Parameters prevent it.

Same-origin policy
The browser rule that stops a page on one site from reading responses from another site.

CORS
Cross-Origin Resource Sharing: headers a server sends to tell browsers which other sites may read its responses. It isn't a lock on your API.

Cross-site request forgery (CSRF)
A trick that makes someone's browser send a request to your site with their cookies. SameSite cookies and origin checks stop it.

Cookie
A small piece of data a site stores in your browser and sends back with requests, often to keep you signed in.

Row level security (RLS)
Database rules that decide which rows each user may read or change.

Prompt injection
Instructions hidden in content (an email, a web page) that try to hijack an AI.

Try it: See XSS happen, then stop it

1. Open the 'Text, not code' starter in the code playground. Add hi both ways. What's different?
2. Add the <img src=x onerror=…> line from the task with innerHTML, and watch the page run code someone typed. Then add it with textContent.
3. In your own app, search the code for innerHTML and dangerouslySetInnerHTML. Where does each value come from?
4. Pick a route that takes an id. Signed in as one test user, ask for another test user's id. Does it say no?
5. Ask your AI helper to review one route for access checks, injection and XSS, then check each thing it finds.

KNOWLEDGE CHECK

Web security basics: access checks, XSS, CORS and cookies

5 questions. Circle one answer for each; 4 or more correct passes. Answers are at the back.

1. The Delete button only shows for admins. Is the delete route safe?

A. Yes, because people can't click a button they can't see
B. Only if the button is also greyed out with CSS for everyone else
C. No: anyone can call the route, so the server must check they're an admin
D. Yes, as long as the route's web address isn't written anywhere on the page

2. Which line safely shows a comment someone typed?

A. box.innerHTML = comment
B. document.write(comment)
C. box.outerHTML = ' ' + comment + ' '
D. box.textContent = comment

3. Your page calls a weather API and the console shows a CORS error. What's true?

A. The browser wouldn't hand your page the answer; call it from your server
B. The API is down for everyone, so try again later
C. Your API key is wrong, so the server refused to answer
D. Your site has been hacked and needs to be taken offline

4. /api/orders/42 returns order 42 to anyone who's signed in. What's missing?

A. A longer, harder-to-guess number for every single order
B. A check that order 42 belongs to the person asking
C. A CORS header that only allows your own site
D. HTTPS, so nobody can read the order on its way

5. Which settings suit a sign-in cookie?

A. Readable by JavaScript, so the page can check it
B. SameSite=None, so it's sent from every site
C. HttpOnly, Secure and SameSite=Lax
D. No settings at all, so every browser accepts it

Official sites

OWASP Top 10:2025: https://owasp.org/Top10/2025/0x00_2025-Introduction/
OWASP: Web Frontend Security Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Web_Frontend_Security_Cheat_Sheet.html
OWASP: Next.js Security Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Nextjs_Security_Cheat_Sheet.html

MDN: Cross-Origin Resource Sharing (CORS):
https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS
MDN: Secure cookie configuration:
https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/Cookies

Last reviewed 2026-11-25. AI tools change fast; check each tool's own site for current features, plans and prices.

ANSWER KEY

Web security basics: access checks, XSS, CORS and cookies

1. C: No: anyone can call the route, so the server must check they're an admin
Hiding a button changes the page, not the server. Every route that changes data checks, on the server, that this person may do it.

2. D: box.textContent = comment
textContent treats the comment as text, so <script> or an onerror trick shows up as characters instead of running.

3. A: The browser wouldn't hand your page the answer; call it from your server
CORS is the browser enforcing the other site's rules about who may read its responses. Calling the API from your own server route avoids it and keeps the key secret.

4. B: A check that order 42 belongs to the person asking
Being signed in says who someone is, not what they may see. Check ownership on the server, or let row level security do it.

5. C: HttpOnly, Secure and SameSite=Lax
HttpOnly keeps scripts (including XSS) from reading it, Secure keeps it on HTTPS, and SameSite stops most forged cross-site requests.