ashton richards

cve-2026-57858: cal.com declines to address stored xss in self-hosted version

expect secure software? please pay $16/mo.

pushed 2026-08-07

CVE: CVE-2026-57858
CWE: CWE-79 — Improper Neutralization of Input During Web Page Generation
Severity: High
Vendor status: No public Cal.diy fix as of August 7, 2026
Last verified: August 7, 2026

0. vulnerability disclosure

> important: this issue remains exploitable. cal.com has confirmed they will not fix this because it does not affect their saas instance. they fixed it on cal.com, but they are unwilling to protect self-hosted users on cal.diy.

stored xss via analytics app tracking id injection into `BookingPageTagManager` in cal.diy

Package: Cal.com, Inc. / Cal.diy

Tested Versions: 2.1.1 (2022-10-24) through 6.2.0 (latest, 2026-03-01).

CVSS 3.1 AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:L = 8.9 HIGH

root cause

`BookingPageTagManager` builds an inline `<script>` tag for the public booking page (`https://{example.com}/<user>/<booking>?overlayCalendar=true`) using `dangerouslySetInnerHTML`, interpolating the analytics app tracking id that an authenticated event owner supplied when installing an analytics app like google tag manager.

the tracking id is stored without sanitization and re-emitted into the inline script body without escaping, which allows attacker javascript to reach the browser as executable code.

the `biome.js` linter explicitly suppresses the `dangerouslySetInnerHTML` warning at this line, but unlike other lint-ignored uses of the same primitive in the codebase, this instance is reachable by attacker-controlled input.

affected files

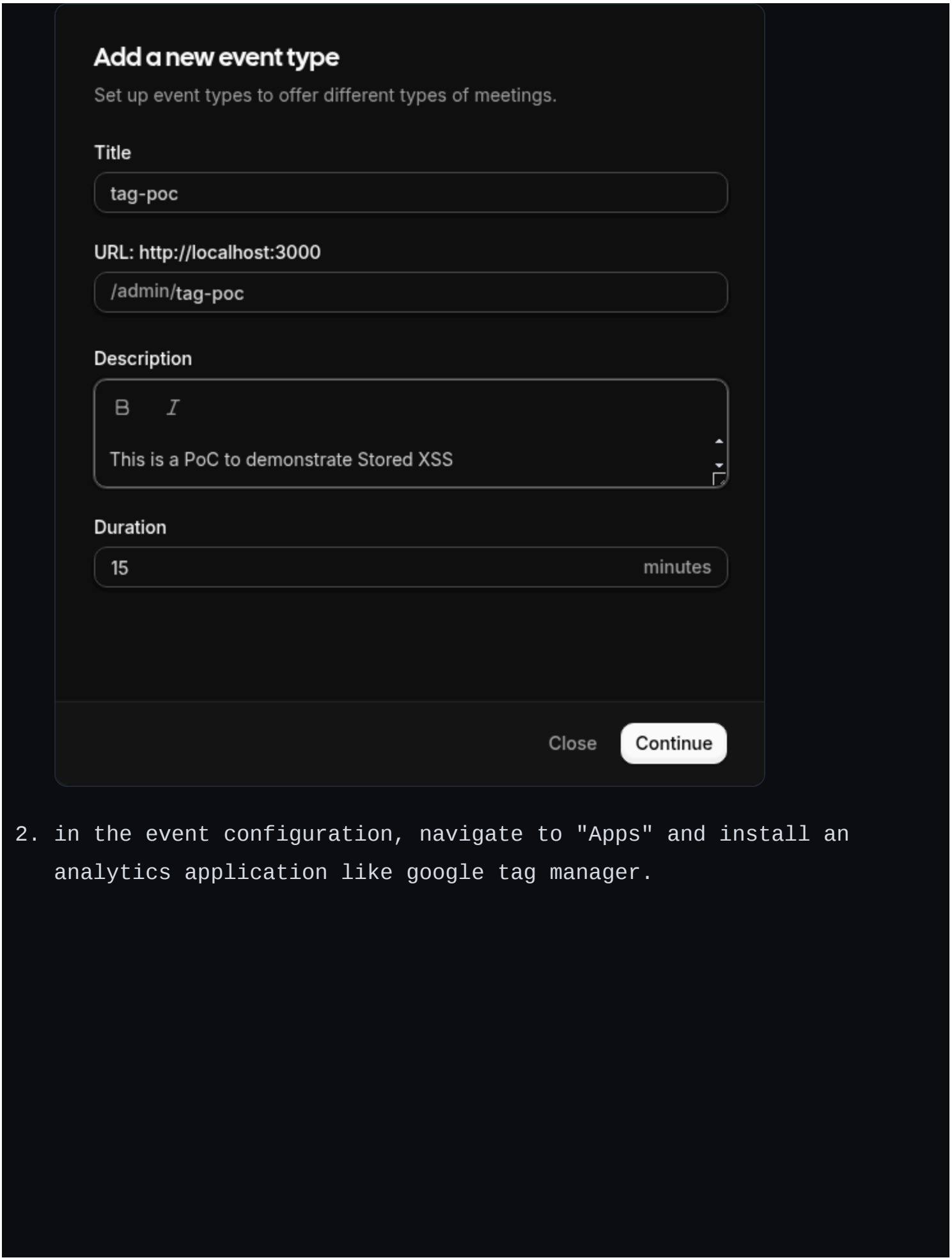
- `packages/app-store/BookingPageTagManager.tsx:92` (vulnerable function) and `:146` (sink) - `BookingPageTagManager` uses `dangerouslySetInnerHTML` with no sanitization on user-controlled

tracking-id values injected into the inline analytics `<script>` block on the public booking view.

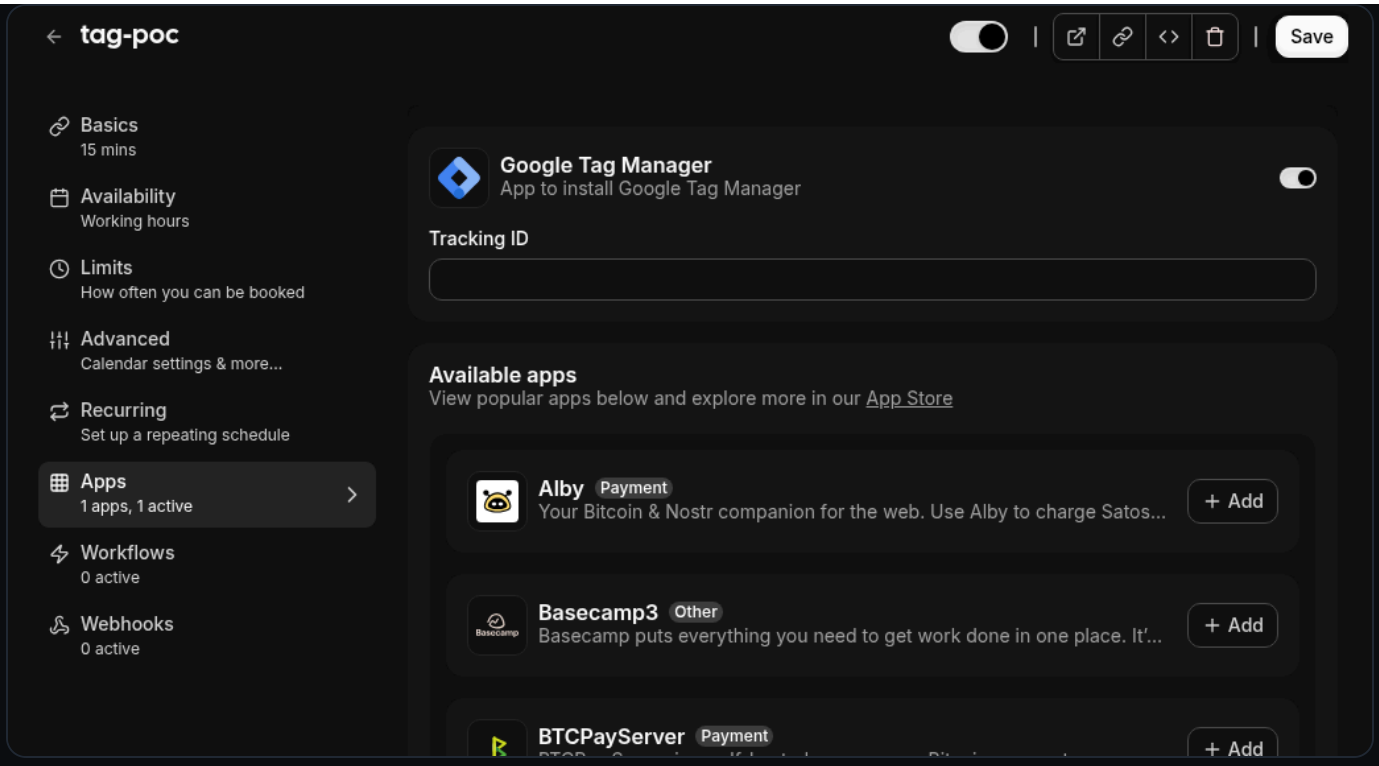
- analytics app tracking id input handler (google tag manager, and other analytics apps in the cal.com app store that plug into the same manager). the value is neither validated at write nor escaped at render.
- the line is annotated to ignore the `biome.js` linter's `dangerouslySetInnerHTML` warning. the annotation only silences the check and does not sanitize the input.

poc

1. as an authenticated user, create a new event type with any attributes desired.



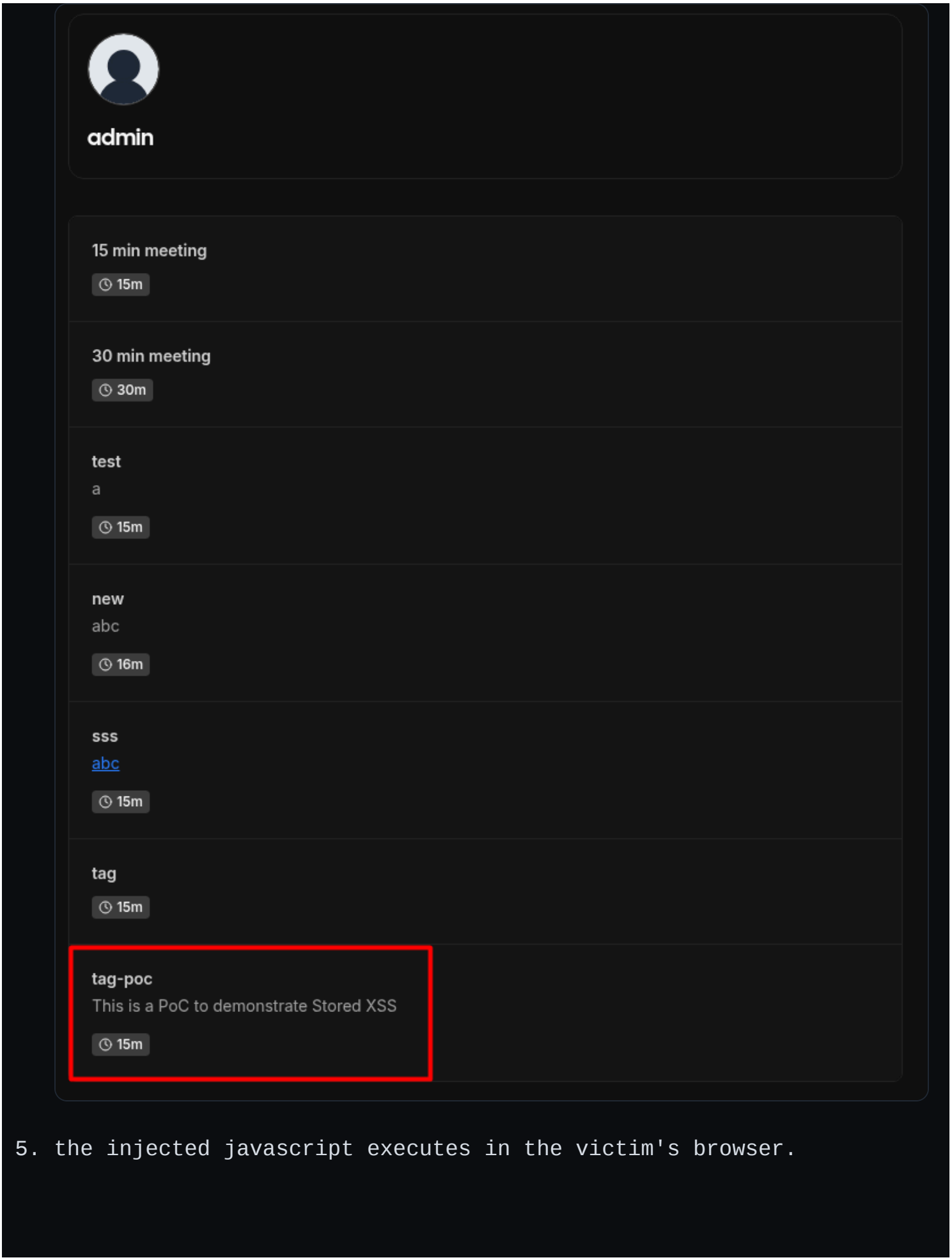
2. in the event configuration, navigate to "Apps" and install an analytics application like google tag manager.



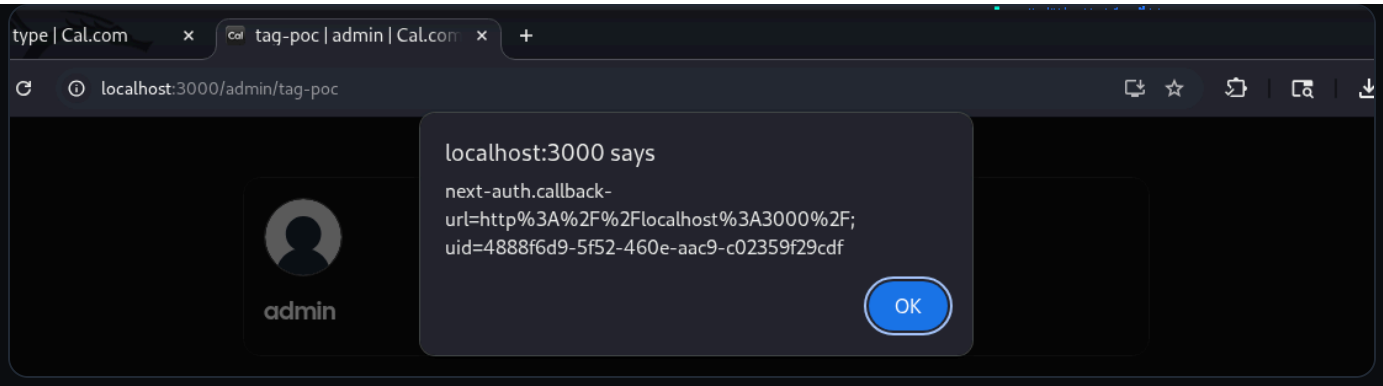
3. set the tracking id to a payload that closes the inline string and appends malicious javascript, then save the event:

```
GTM-abc');alert(document.cookie);//
```

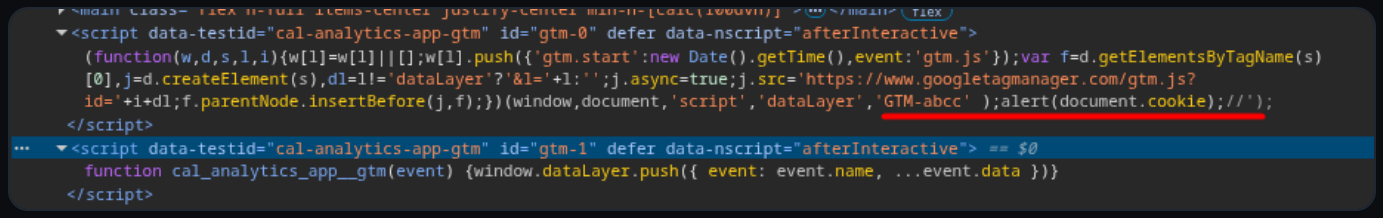
4. in a separate session, navigate to the public booking page (`https://{example.com}/<owner>/<booking>?overlayCalendar=true`).



5. the injected javascript executes in the victim's browser.



6. observe injected input in source html.



impact

an authenticated cal.diy user can install an analytics app on an event they own to persist an arbitrary javascript payload in the analytics tracking id field. this payload executes in the browser of any visitor to that booking url for the event.

reachable consequences on the cal.diy origin include theft of cookies and state, forced actions via csrf-able requests, and compromised user actions on the vulnerable page.

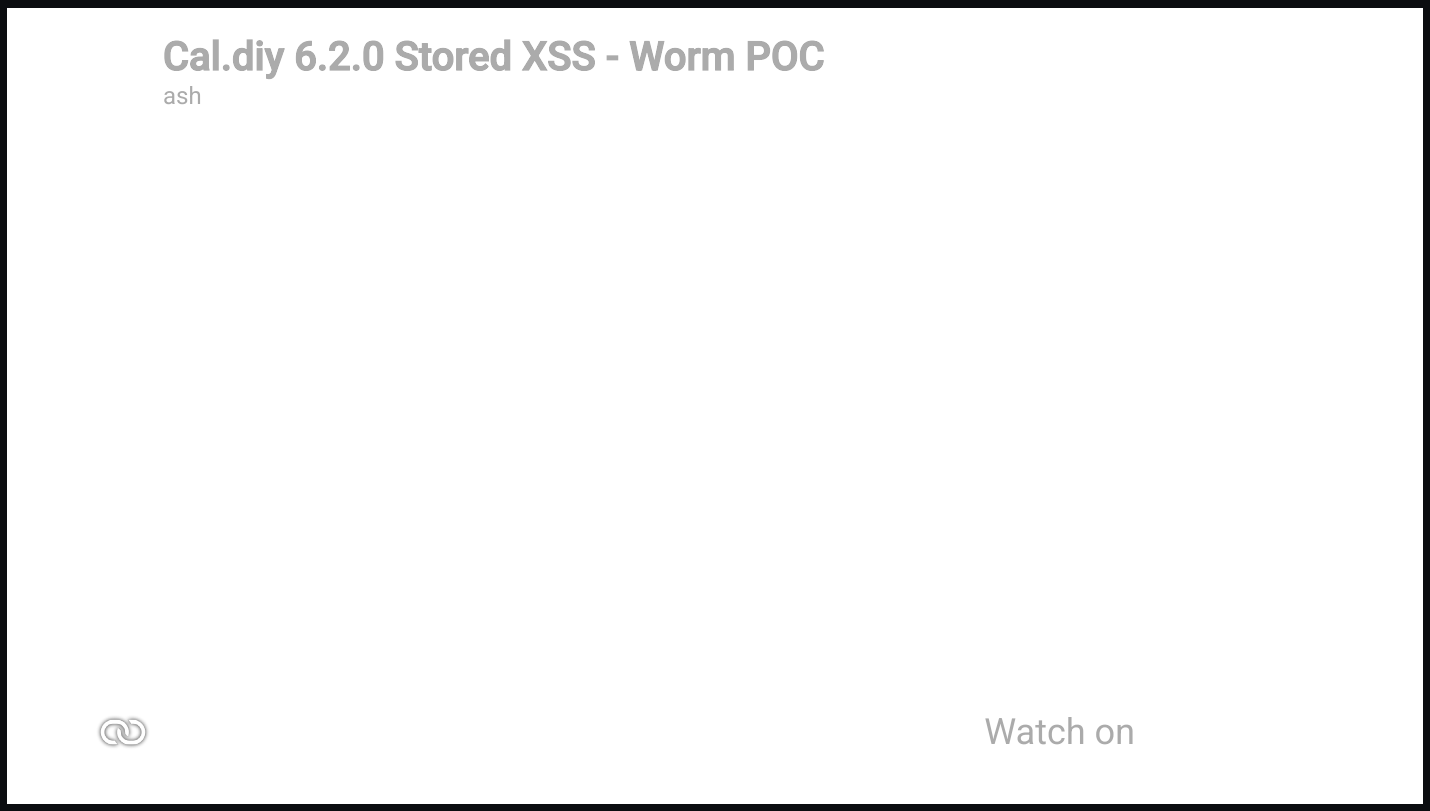
the payload is stored and re-fires for every visitor on the booking. when combined with csrf to create events and store a new payload, this leads to an xss worm.

this has been present for nearly 4 years. shodan reflects at least 263 affected self-hosted instances based on favicon hash.

bonus: wormable impact

after cal.com incorrectly claimed the issue was fixed, i took the impact to the next level and demonstrated a wormable exploit that propagates the malicious payload and creates new events for infected users.

the exploit will be published on github if a fix becomes available in a future release.



mitigation

no vendor fix is available as of 2026-08-07.

until an upstream fix is available, self-hosted users should:

- restrict or disable analytics integrations where possible
- remove untrusted analytics tracking ids
- review stored analytics configurations

- apply a local patch to escape/sanitize tag manager values
- validate any local mitigations against the deployed version

---

1. timeline

disclosure took over 7 months, and ended with cal.com claiming they will not fix the issue because it does not affect their saas instance.

between disclosure and this final claim, cal.com rebranded their open-source repository to `cal.diy`, and apparently absolved the responsibility of delivering secure software to self-hosted users in doing so.

from their blog:

> *"Our goal was to hand Cal.diy to the community in a solid state from a security perspective. That meant treating the public repo as a production codebase, not an afterthought, even while the commercial product was being built in the private repo."*

ironically, this issue is not present in their saas instance, but has existed for over 4 years and counting in cal.diy. they're right security was not an afterthought for cal.diy, it was a neverthought.

January 17, 2026

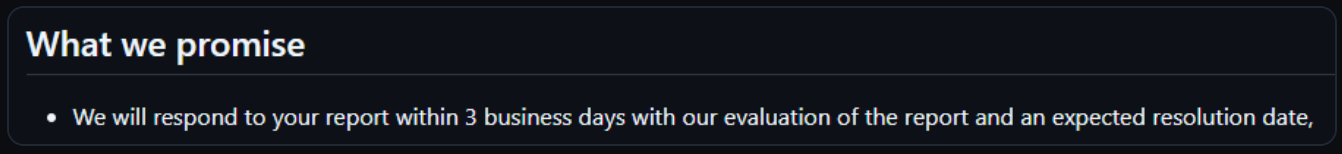
initial disclosure

reported the vulnerability with reproduction steps via email to cal.com's listed security contact.

January 23, 2026

follow-up #1

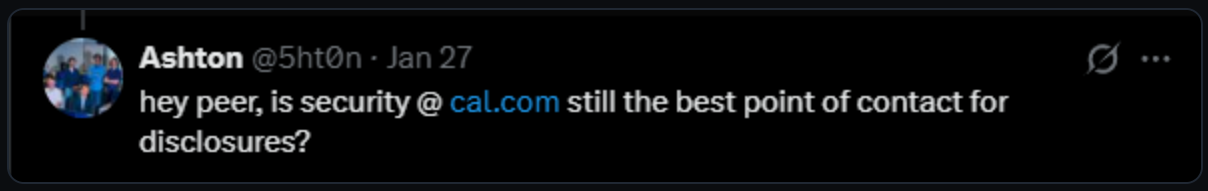
despite "promising" a response within 3 business days in their security disclosure process, cal.com does not respond after 5 business days.



January 27, 2026

tweet #1

an out-of-band attempt was made to contact cal.com without disclosing details.



February 9, 2026

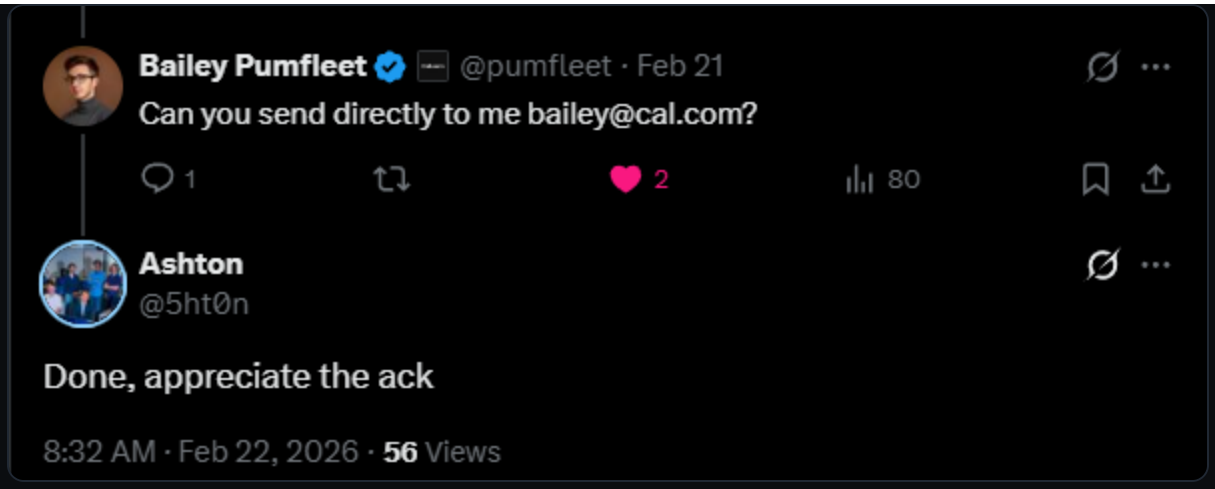
follow-up #2

another follow-up sent via email 16 business days later.

February 21, 2026

tweet #2

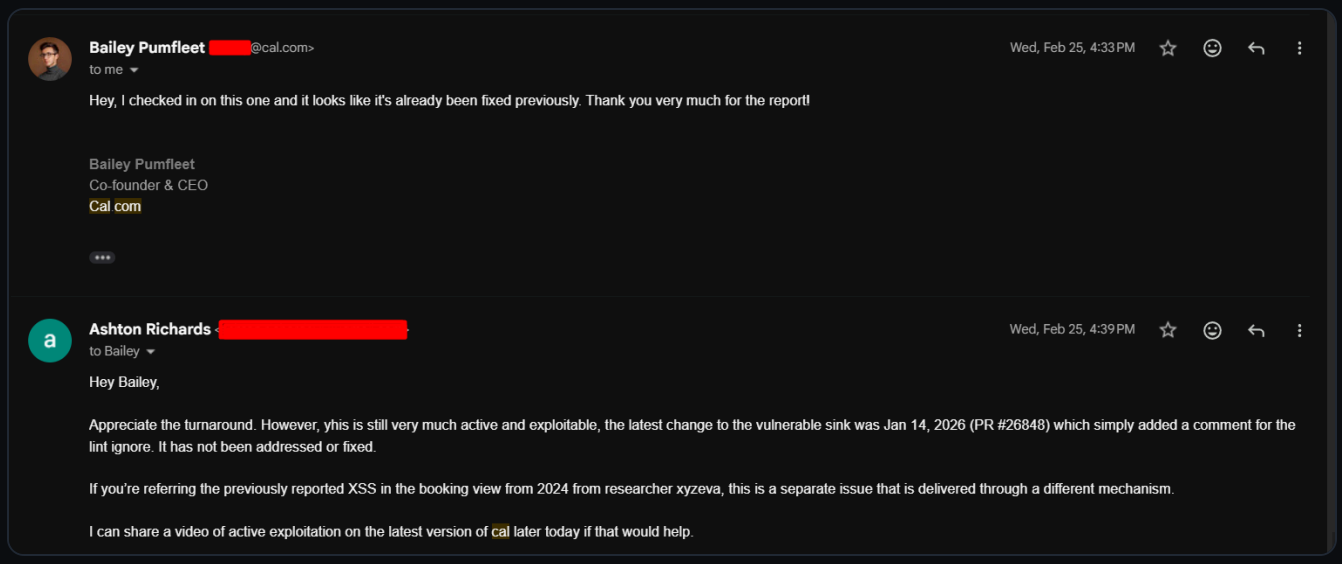
a new thread received acknowledgement from the co-founder requesting a personal email.



February 22, 2026

secondary disclosure

as requested, a new email was sent to the co-founder of cal.com with the same details as the initial report.



February 25, 2026

official acknowledgement

the co-founder acknowledges the report and claims it has been previously fixed. a follow-up was immediately sent challenging this assertion.

April 15, 2026

cal.com goes closed-source and rebrands cal.diy

cal.com renames the repo and refactors all code to point to cal.diy for self-hosting use cases.

July 1, 2026

fresh disclosure attempt

given the vulnerable code path still existed, i recorded a more severe, wormable impact and attempted communications via a new email thread.

July 8, 2026

third-party disclosure process initiated

i reported this issue to vulncheck to receive assistance with disclosure, trying my hardest to have cal.com reconsider and fix the vulnerability.

July 16, 2026

cve reserved

vulncheck reserved a cve for the issue and attempted further communications with cal.com. these communication attempts continued throughout the course of july.

July 30, 2026

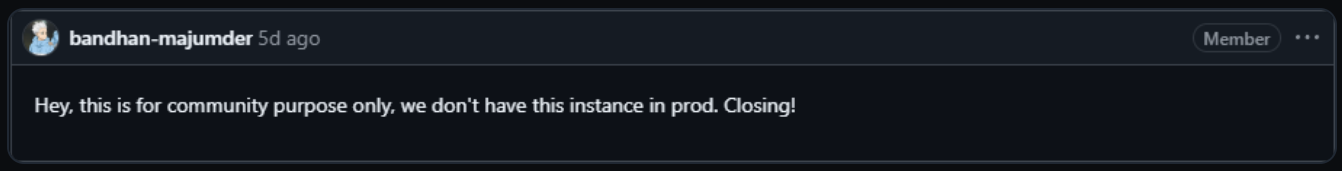
github issue raised

vulncheck raised a github issue outlining the report.

August 2, 2026

issue closed

a software intern at cal.com closed the issue since it doesn't affect their enterprise instance.



August 5, 2026

vulnerability disclosed via github issue

vulncheck discloses the full report of the finding in response to cal.com confirming they will not fix.

August 7, 2026

public disclosure

this post is published as public disclosure of cve-2026-57858 to help protect users that are running self-hosted versions of cal.com, as they deserve secure software just as much as enterprise users do.

2. personal thoughts

i would not trust cal.com with running a calendar based on how this disclosure was handled.

originally, i believed they had waived this issue because they didn't understand the report or impact. i thought maybe the `biome.js` linter had baited them into thinking it wasn't exploitable.

in hindsight, i suspect they immediately waived this report as a non-issue because they knew their transition to cal.diy was coming, and wanted nothing to do with responsibility of user security there.

despite this transition to cal.diy occurring 4 months after initial report, *someone* still needs to be responsible for the security of the self-hosted version. they call it a community-led effort, but an official cal.com employee closes the report due to it not affecting their enterprise production version.

regardless of if it affects your paying customers, it does affect your self-hosted users. they deserve secure software just the same. disclosures are not always to protect your company - they protect your users.

---

3. gallery

during this timeline, there were a few cal.com communications that were quite ironic given the circumstances.

transparency

> unless it affects cal.diy security, then we won't bother.

receiving a link

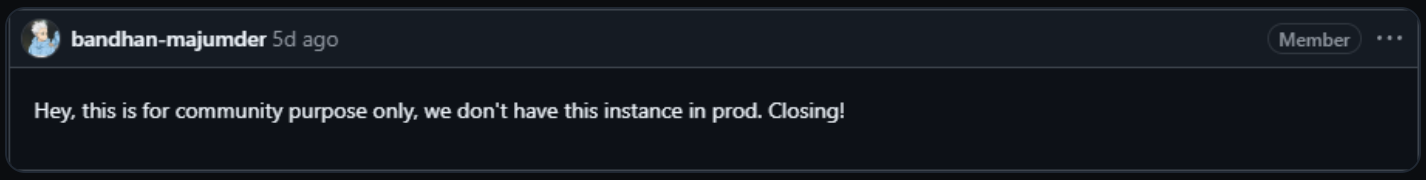
> it's not homework, it's actually an exploit vector!

is it production, or not?

their official blog on cal.diy:

> *"That meant treating [cal.diy] as a production codebase..."*

their intern:



© 2026 ashton richards