BACK

CVE-2026-71503: Dolibarr Reflected XSS to Admin Takeover

CVE-2026-71503 CVSS 9.3


Amartya Jha

Aug 23, 2026

What if a single link, opened once by one of your admins, quietly minted a second administrator account in your system, with a password and an API key the attacker picked?

That is what CodeAnt AI Security Research found in Dolibarr, the open-source ERP and CRM that runs the accounting, invoicing, HR, and payroll for a whole small business.

CVE-2026-71503 is a reflected cross-site scripting bug in Dolibarr through 25.0.0-alpha. An administrator opens a crafted URL, nothing on the screen changes, and a second administrator account appears in the database. The interesting part is how it gets there. Stealing the session cookie was the obvious move, and it failed. It also turned out not to matter.

It carries a CVSS score of 9.3, Critical.

The fix is live in Dolibarr 24.0.0. If you run an affected build, stop reading and upgrade.

Now here is how we found it, and how it works.

How One Click Became an Admin Account

Dolibarr's admin setup pages take one URL parameter and print it straight into a `<script>` block. The value does pass through a filter first, but the filter is watching for the wrong characters. It catches a handful of dangerous HTML characters and lets a single quote through.

That quote closes the JavaScript string Dolibarr just built, and everything typed after it becomes new instructions. Stealing the session cookie, the obvious next move, goes nowhere here because Dolibarr's cookie is unreadable to JavaScript.

What works instead is the anti-forgery token, which Dolibarr prints into every page so its own scripts can use it. Read it off the page, attach it to a request, and the request goes through as the administrator.

Getting to that point requires nothing more than a logged-in admin who opens one link. The attacker needs no account, no forged CSRF token, and no further interaction after the click.

The page answers with `200 OK` and no Content-Security-Policy header. The injected script fires on load, with no confirmation prompt and nothing on screen to suggest anything happened.

In our test, that one click produced a second administrator account with a password and API key the attacker chose. It survives long after the victim closes the browser, which is what turns a reflected XSS into standing access. Severity lands at critical, CVSS 9.3, with the full vector in the table below. Dolibarr 24.0.0 closes it.

Attribute	Details
CVE	CVE-2026-71503
Affected component	Dolibarr shared extra-fields admin template (reflected via the `type` parameter)
Issue	Reflected XSS (CWE-79)
CVSS	9.3 Critical (`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:N`)
Requires	A logged-in administrator opens a crafted link once
Affected versions	Dolibarr through 25.0.0-alpha
Fixed version	Dolibarr 24.0.0
Discovered By	CodeAnt AI Code Security Team

Where This Started

At CodeAnt, we're trying to find the security flaws that matter in the software developers actually use. That means taking popular open-source applications, reading how they are built, and then testing whether the weaknesses we find in the code can be turned into something real. Dolibarr was one of those applications.

Most of our findings there followed one pattern: two parts of the application were supposed to enforce the same permissions, but didn't. This finding was different.

While going through Dolibarr's admin setup pages, one URL parameter came back reflected inside a `<script>` block. The reflex move with an XSS is to steal the session cookie, so we tried. It came back empty. Dolibarr sets the cookie so JavaScript can't read it at all, so the payload had nothing to grab.

That failure is where it got interesting. The thing that made cookie theft pointless, a security token Dolibarr prints into every page for its own use, turned out to be the exact thing that made a far worse attack possible.

Instead of stealing a session, the injected script could read that token straight out of the page and use it to create a brand-new administrator.

That is where we landed. A 9.3, from one click.

What We Found

Picture a page that takes something from the web address and places it directly inside JavaScript running in the browser, the same way a template drops your name into the blank on a form letter.

Dolibarr did filter that value, but the filter was checking for the wrong thing. It could block some dangerous HTML characters, but it did not safely handle a single quote.

That single character was enough to close the piece of JavaScript Dolibarr had created and add new instructions of the attacker's choice.

In other words, Dolibarr was checking the input before it came in, but it wasn't safely escaping the value when it put it back into JavaScript. The frustrating part is that Dolibarr already had the right protection elsewhere in the codebase. It simply wasn't being used here.

Affected code

- Vulnerable value reflected in: `htdocs/core/tpl/admin_extrafields_add.tpl.php:153`
- Existing filter, checking the wrong thing: `htdocs/core/lib/functions.lib.php:1443`

- Dolibarr's own JavaScript-escaping function, `dol_escape_js()`, already used elsewhere in the codebase and never applied here

How One Click Became an Admin Account

The attack comes down to three steps: the admin opens the link, the injected script runs in their browser, and the script uses the admin's own permissions to create another administrator. Nothing about step one looks unusual. The link leads to a screen for adding a custom field, the kind of page an admin might open as part of normal work.

Two ordinary settings are what let steps two and three happen without anything stopping them. Dolibarr also treats this page as a safe action that does not need the usual anti-forgery protection. That makes sense if the page only displays a form. The problem is that the same page can be reached with the malicious value already attached to the URL.

Most fresh Dolibarr installs also don't turn on a Content Security Policy, the browser-level setting that would otherwise block an injected script from running at all. Because it's off by default, nothing stands in the way once the script is on the page.

This is where the page's own security token matters. Dolibarr prints that token into every page so its built-in scripts can use it, which means an injected script can read it too, without needing to steal anything. It's already sitting on the page. The script reads it, then uses it to submit its own request as if it were legitimate, and the browser attaches the admin's login automatically because it's a request to the same site.

That's step three: a request that looks like it came from the admin, asking Dolibarr to create a new administrator account.

One limit is worth being precise about: this only works inside an administrator's own logged-in session. Anyone opening the same link while logged out, or logged in as a regular user, sees nothing happen.

That's exactly why the attack has to arrive as a link for an admin to open, rather than something that works on its own.

Affected code

- Page restricted to admins only: `htdocs/product/admin/product_extrafields.php:55`
- Action exempted from the anti-forgery check: `htdocs/main.inc.php:373-395`
- Content Security Policy header, off by default: `htdocs/main.inc.php:1635`
- Confirmed reflected on: Products, Agenda, and Third-parties setup pages, the same shared template included by 80 files in total, so disabling one module only moves the door

Proof of Concept

Every request below ran against an isolated Dolibarr container we stood up at `http://127.0.0.1:18080`. No public or third-party install was ever contacted.

Control. Confirm the reflection runs unescaped, with no CSP in the way. Requesting the following inside the administrator session returns `HTTP 200`, with no Content-Security-Policy in the response:

The inline `<script>` block now reads:

The literal is closed and the attacker's statements run as code.

Exploit. Swap the alert for the real payload. The weaponized `type` value replaces it with a fetch to `/user/card.php` carrying `action=add`, the token read live from the page, `admin=1`, an attacker-chosen password, and an attacker-chosen API key.

Driving the logged-in administrator in headless Chromium, the request returned 200 and a new user row appeared, `dol16pwn`, with admin rights and the attacker's key encrypted at rest.

Confirm. Use the credentials cold. We tested three keys side by side. The key that existed only because the victim clicked read the entire user directory. The two ordinary, non-administrator keys were refused.

The chosen password logged straight into the admin panel. We deleted the created account through the UI afterward, and the key then returned 401, confirming it was that account, and only that account, doing the work.

What That Second Administrator Can Do

The important part is that this isn't a stolen browser session. It's a real administrator account. It has a password chosen by the attacker. It has an API key chosen by the attacker. And it survives after the victim closes their browser.

The admin only had to open the link once. The link itself can arrive by email, ticket, or chat, and it looks like an ordinary internal setup page.

An operator who resets the victim's password in response, but never checks the user table for a new account, leaves the attacker's API key working. Seconds after the click, that key can read the entire user directory through Dolibarr's own API.

A couple of things do not fall in the same breath, and that detail matters for research credibility.

Records tied to specific modules, like invoices, customers, and payroll, stay protected by Dolibarr's ordinary permission checks, which have no special bypass for administrators.

But the account is still a full administrator, and one further request lets it grant itself any permission it wants, because simply being marked an administrator is enough to unlock that one action.

And it doesn't depend on any single module. The same shared template sits behind 80 different pages, and we confirmed the same reflection on the Agenda and Third-parties setup screens as well as Products. Turning off one module only moves the door to another.

What This Finding Actually Shows

CVE-2026-71503 is a useful example of why a severity score doesn't tell you the impact by itself. On paper, it's a single reflected XSS.

The real damage only shows up once the browser's own protections, the anti-forgery design, a routine admin workflow, and the built-in account-creation feature are chained together. That's the gap between finding a vulnerability and proving an attack path.

This finding is also a deliberate exception in our Dolibarr research. The rest of that work documents a recurring access-control pattern we call split-brain authorization: two surfaces writing the same records and disagreeing about who's allowed to.

This one is a missed output-encoding step rather than a missing permission check, yet it earns its place in the same body of work for the same underlying reason. The safe tool already existed in the codebase, and this one screen simply skipped it.

Escaping output for the context where it's used matters more than filtering input ever will: filtering an input does not make it safe to place inside JavaScript, HTML, SQL, or another execution context.

A Content Security Policy is a useful second layer, not a primary fix, but it can stop one missed escaping mistake from becoming executable code. And a page considered "read-only" can still become dangerous if attacker-controlled input reaches it and the browser executes that input, which is exactly why exempting it from anti-forgery checks deserves a second look.

We found this not by stopping at a suspicious line of code, but by going from "this parameter is reflected in JavaScript" to "one admin opening one link creates another administrator with credentials controlled by the attacker."

If you want to test your own application the way an attacker would, start with a free CodeAnt pentest.

See If You Are Affected

If you run Dolibarr on an affected build, any admin who opens a crafted link can be turned into a second-admin factory. Upgrade to Dolibarr 24.0.0.

The patch closes the gap right where it lives: the raw output now goes through Dolibarr's own JavaScript-escaping function before it's printed, and because the fix sits in the shared template, it protects all 80 pages that use it at once.

If you can't upgrade right away, turn on a Content Security Policy that blocks inline scripts and instead trusts only scripts carrying the security token Dolibarr already generates for this purpose. That one setting alone stops the injected script from running, even without the code fix.

Related research. This finding joins our split-brain authorization pillar as its deliberate exception, the same shape expressed as an encoding bug rather than an access-control one. For the strongest of the access-control findings, see how a low-privilege caller reaches customer records through the Third Parties REST API in CVE-2026-71505, where the web UI checks ownership and the API forgets to.

[FAQ]

Frequently Asked Questions

What is CVE-2026-71503? +

Is CVE-2026-71503 remotely exploitable? +

Which Dolibarr versions are affected by CVE-2026-71503? +

Can CVE-2026-71503 lead to account takeover? +

Does the HttpOnly cookie protect Dolibarr against CVE-2026-71503? +

How do I fix CVE-2026-71503? +

How can CVE-2026-71503 be mitigated without upgrading? +

More from CodeAnt

VIEW ALL

[GET STARTED]

START PENTEST BOOK A DEMO