BACK

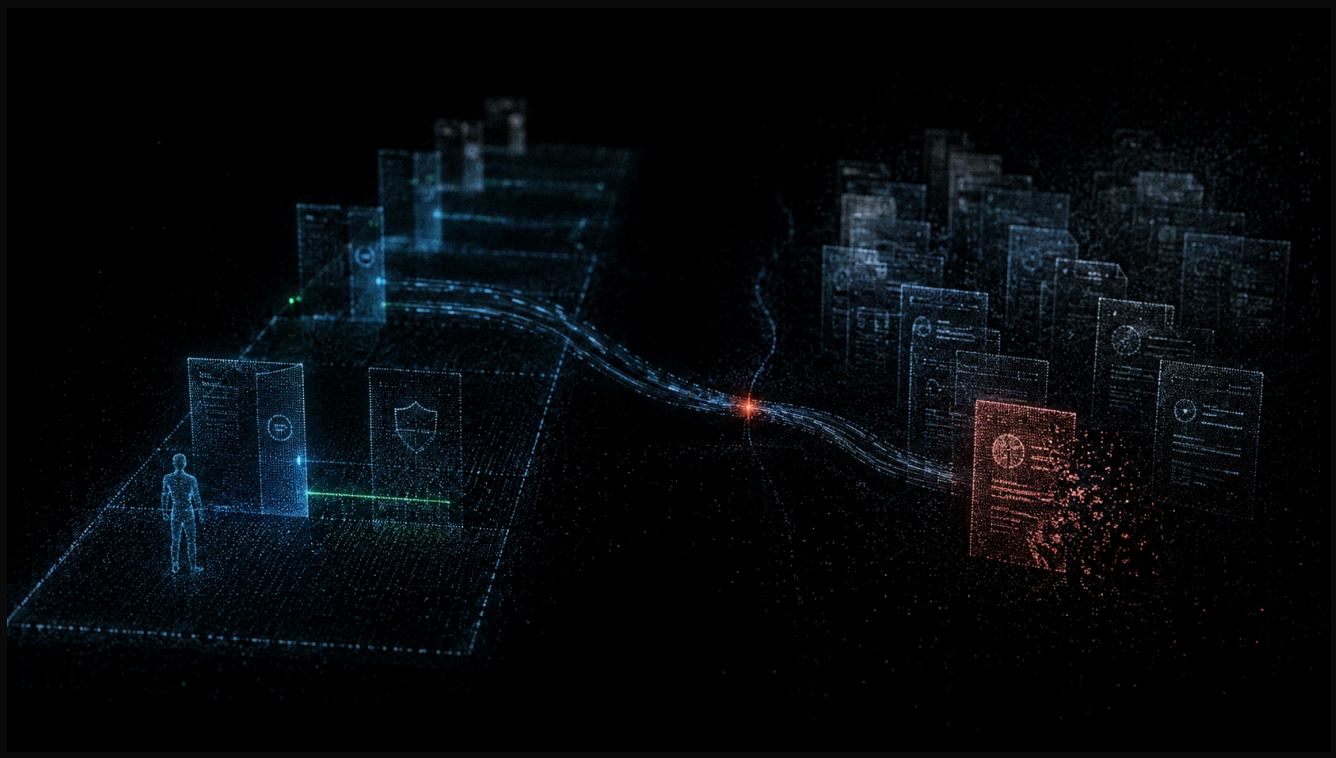
CVE-2026-71506: Dolibarr Payment Deletion via Incorrect Authorization

CVE-2026-71506 CVSS 6.5



Amartya Jha

Aug 23, 2026



Research

- The Check Ran, It Just Asked the Wrong Question
- Where This Started
- How Dolibarr Splits Money From Invoices
  - Affected code
- What We Found
  - Affected code
- What Actually Disappears
- Proof of Concept
- Why This Matters Beyond This One Route
- What This Finding Actually Shows
  - Are You Affected?

BOOK A DEMO

money they already sent?

That is what CodeAnt AI Security Research found in Dolibarr, the open-source ERP and CRM that runs invoicing and accounting for small and mid-size businesses.

CVE-2026-71506 is an incorrect-authorization bug in Dolibarr's payments REST API. The vulnerable `Paiements::delete()` route checks `facture.supprimer` (the permission to delete invoices) instead of `facture.paiement`, the permission that governs payments. The check runs. It just asks the wrong question.

It carries a CVSS score of 6.5. A non-admin holding only invoice-deletion rights can hard-delete recorded customer payments and inflate the apparent debt on those invoices, without ever holding the permission that actually controls receipts.

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.

The Check Ran, It Just Asked the Wrong Question

Dolibarr splits two permissions apart on purpose. One deletes invoices, the kind of access you hand a junior clerk cleaning up mistaken paperwork. The other manages payments, and it belongs to whoever reconciles the bank. The route that deletes a recorded payment was supposed to demand the second one. It demands the first. The check is there. It runs. It just names the wrong permission, and that one word is the whole finding.

So the clerk permission, the one deliberately kept away from the ledger, deletes real customer receipts. Every other payment path in Dolibarr gets this right. The web UI asks for the payment permission. The canonical API asks for it, and goes further, checking that the caller owns the invoices a payment touches. This one route asks for the wrong key.

below.

Attribute	Details
CVE	CVE-2026-71506
Affected component	Dolibarr payments REST API (`DELETE /api/index.php/paiements/{id}`)
Issue	Incorrect Authorization (CWE-863)
CVSS	6.5 Medium (`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N`)
Wrongly required permission	`facture.supprimer` (Delete invoices)
Correct permission	`facture.paiement` (Issue payments on invoices)
Fixed version	Dolibarr 24.0.0
Discovered by	CodeAnt AI Code Security Team

Where This Started

At CodeAnt AI, we keep chasing one question: which open-source packages actually run real businesses, and what critical bugs are hiding inside them?

One of our researchers kept circling ERP platforms, and Dolibarr in particular, because an ERP is where a company's money actually lives. So we started reading the paths that move money.

Dolibarr is careful about this. It splits the authority to edit invoices from the authority to touch the payments recorded against them, and the money paths are supposed to demand the money permission.

We checked, and the careful places agreed: the web interface asks for it, and the canonical invoice API asks for it, even checking that the caller owns the invoices a payment is allocated to.

So we asked the question that keeps turning up findings in this codebase: if every careful path demands the payment permission, does every path? We walked the payment routes one at a time. Then we reached the one that deletes a payment. It asked for the permission to delete invoices instead.

We handed it a key belonging to a low-privilege clerk, a key with invoice-deletion rights and nothing to do with money. The delete returned 200.

The payment row was gone, its allocation gone with it, and the invoice that had been part-paid now read zero paid against the full amount, still marked validated, as if the customer had never sent a thing.

That is where we landed.

How Dolibarr Splits Money From Invoices

Before the bug, the design it breaks.

Dolibarr deliberately keeps two permissions apart. One is the right to delete invoices, the kind of thing you'd hand a junior clerk so they can clean up mistaken or duplicate paperwork.

The other is the right to issue and manage payments, the actual money a customer has sent, which belongs to whoever reconciles the bank.

These are two separate, independently grantable permissions, and that separation is the point. It's how you let someone tidy invoices while keeping them away from the ledger.

Every careful path in Dolibarr honors it.

The web interface asks for the payment permission on both creating and deleting a payment.

Two doors write the same payment records, the web UI and the API, and they're supposed to demand the same key. One of them had the wrong lock fitted.

Affected code

- Payment creation and deletion in the web UI, both correctly gated on `facture.paiement`: `htdocs/compta/paiement/card.php:73` (create), `:132` (delete)
- Canonical REST endpoints, all correctly gated on `facture.paiement`: `htdocs/compta/facture/class/api_invoices.class.php:1752, 1879, 2022`
- The object-level check that confirms the caller owns the specific invoices touched: `api_invoices.class.php:2036-2042` (`putPayment()`, via `getBillsArray()` and `_checkAccessToResource('facture', $id)`)

What We Found

Here's the method that has the wrong lock on it, trimmed to the load-bearing lines.

```
// htdocs/compta/facture/class/api_paiements.
public function delete($id)
{
    if (!DolibarrApiAccess::$user->hasRight('
        throw new RestException(403);
    }
    // no object-scope check on the invoices
    $result = $this->paiement->fetch($id);
    if (!$result) { throw new RestException(4
    if ($this->paiement->delete(DolibarrApiAc
        throw new RestException(409, ...);
    } elseif ($this->paiement->delete(Dolibar
        throw new RestException(500, ...);
    }
}
```

The route also lacks the object-scope authorization used by the canonical payment path, the check that confirms you own the invoices a payment touches.

That's a real gap, but it's secondary to the headline issue: even with that check in place, a caller holding the wrong permission should never have reached this far at all.

There's a second bug we observed in the same method, separate from the authorization flaw. The underlying delete function never actually returns 0, so the first branch above is dead code.

On a successful delete, control falls through into the second branch and causes the method to attempt the deletion a second time. That double call left a fingerprint in our testing: a bank-linked control payment was refused with a 500 traced to the second branch, not the first, confirming the method really does try to delete twice.

Affected code

- Vulnerable route, primary flaw: `htdocs/compta/facture/class/api_paiements.class.php:227-250`, `Paiements::delete()`, checks `facture.supprimer` (line 229) instead of `facture.paiement`
- Secondary gap: missing the object-scope loop that `putPayment()` runs elsewhere in the same file
- Separate implementation bug in the same method: the underlying delete function never returns 0, making the first branch dead code: `htdocs/compta/paiement/class/paiement.class.php:699`

What Actually Disappears

Deleting a payment in Dolibarr does more damage than it looks like, because of how the app stores what a customer has paid.

An invoice does not carry its own paid figure. What a customer has paid is the running sum of the allocation rows in a separate table. So deleting a payment does not cleanly mark an invoice unpaid. It silently raises the balance the customer still appears to owe.

Nothing was reversed. The financial records were hard-deleted, with no tombstone left behind on this path.

Proof of Concept

Every request here was fired at an isolated Dolibarr container on loopback: no public host, no third party, nothing but a Docker instance built to be thrown away afterward.

The actor is an internal, non-admin user (`admin = 0`) whose rights we read straight out of the database rather than take on trust: exactly the invoice-read and invoice-delete permissions, nothing else. No payment permission.

Control. Prove the payment permission is actually required elsewhere. A principal holding only third-party read rights sends the identical DELETE and is refused `403`, confirming the guard is real and firing.

Then the actor offers its own key to the canonical payment-update route. Dolibarr refuses that too, forbidding this same user from so much as renaming a payment.

Exploit. Send DELETE using only the invoice-deletion permission. The request is unremarkable. The response:

```
HTTP/1.1 200 OK
{"success":{"code":200,"message":"Paiement de
```

We ran it for the other two payment ids. Three 200s, three payments gone.

Confirm. Verify the payment and allocation rows are gone. A database read-back showed zero remaining rows in either the payments table or its invoice-allocation table, and all three invoices back at 0.00 paid.

Why This Matters Beyond This One Route

Separation of duties is exactly why the invoice-deletion and payment permissions are kept distinct. You let a junior clerk clean up mistaken or duplicate invoices, and you deliberately keep them away from recorded receipts, which belong to whoever reconciles the bank.

Read the permission by its plain name, and invoice deletion says nothing about money. This flaw quietly folds payment destruction into it.

That's a practical abuse primitive: one request per payment can remove receipts, against ids the account may enumerate legitimately, leaving open invoices dunning customers for money they already paid, and ledger entries pointing at receipts the ERP no longer knows about.

Unless the optional Unalterable Archives module is on, and it's off by default, nothing reaches the event log at all.

This is one node in a pattern we saw across Dolibarr's REST surface: two write paths into the same set of rows that end up disagreeing about who may touch them.

We call it split-brain authorization, and it turns up across nine findings from this research effort. Its closest sibling here is CVE-2026-71510, where Dolibarr's sqlfilters parser leaks column data one boolean at a time. Both are a check that genuinely executes, and then decides the wrong thing, seen from opposite sides.

What This Finding Actually Shows

CVE-2026-71506 wasn't caused by a missing authorization check. Dolibarr checked a permission before deleting payments, and the check genuinely runs.

The failure was more specific: it checked `facture.supprimer` instead of `facture.paiement`. That's the difference between an authorization check existing and an authorization check being correct.

The result was that a low-privilege invoice clerk could delete recorded customer payments without ever holding the payment permission.

Are You Affected?

If you run Dolibarr through the affected builds and any non-admin holds the invoice-deletion right, that user can delete payments through the REST API today. Upgrade to Dolibarr 24.0.0.

The fix is small, because the correct behavior already existed twenty lines away: align the permission check with the payment right, add the object-scope loop after the fetch, and capture the delete result once so the routine stops running twice. The 24.0.0 release closed the gap.

The broader lesson is the one this whole cluster keeps arriving at. When two surfaces independently write the same records, the permission each one demands drifts apart unless something forces them to agree. A single table-driven map from object action to required right, consulted by both layers, is what stops a route from quietly asking for the wrong key.

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

Related research. This finding is one node in the split-brain authorization pattern across Dolibarr's REST surface. Its closest sibling is CVE-2026-71510, where the sqlfilters parser leaks column data one boolean at a time, the same failure mode from the opposite side.

[FAQ]

Frequently Asked Questions

What is the impact of CVE-2026-71506?

What is incorrect authorization (CWE-863), and how does CVE-2026-71506 illustrate it?

What privileges does an attacker need to exploit CVE-2026-71506?

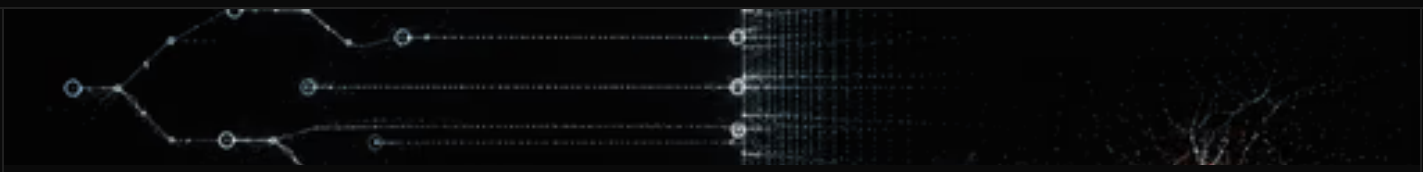
Which Dolibarr versions are affected, and how is CVE-2026-71506 fixed?

What can an attacker do with CVE-2026-71506?

How is a wrong-permission check different from a missing authorization check?

More from CodeAnt

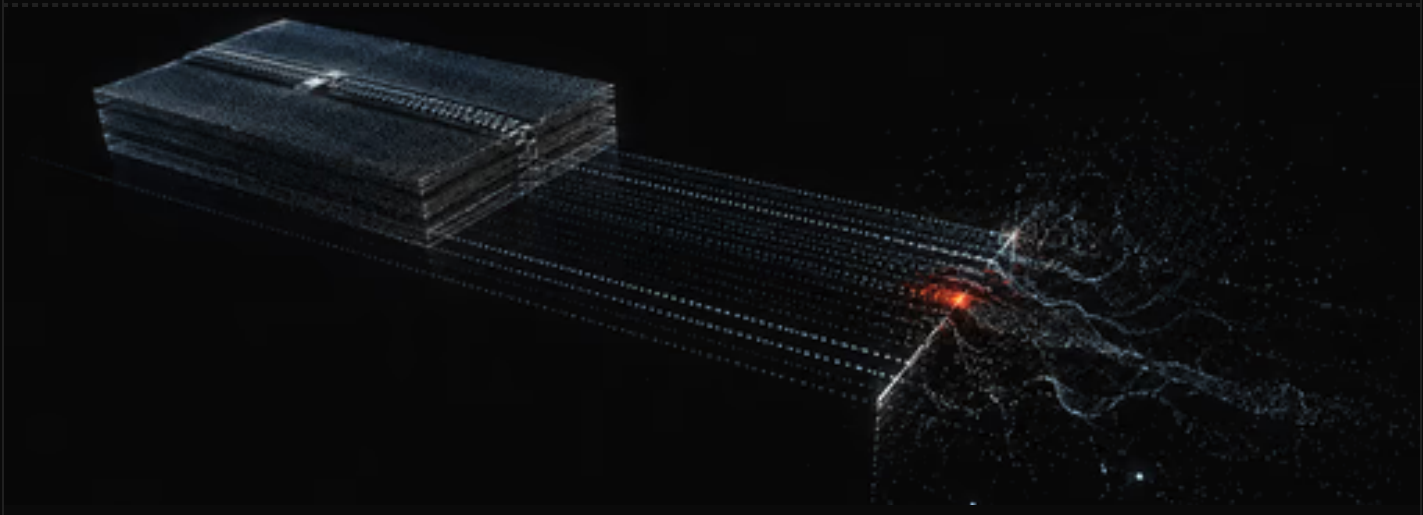
VIEW ALL



Mar 3, 2026

CVE-2026-29000: pac4j-jwt Auth Bypass PoC With a Public Key

CVE-2026-29000 CVSS 10



Mar 9, 2026

CVE-2026-28292: simple-git RCE via Case-Sensitivity Bypass

CVE-2026-28292 CVSS 9.8



Mar 11, 2026

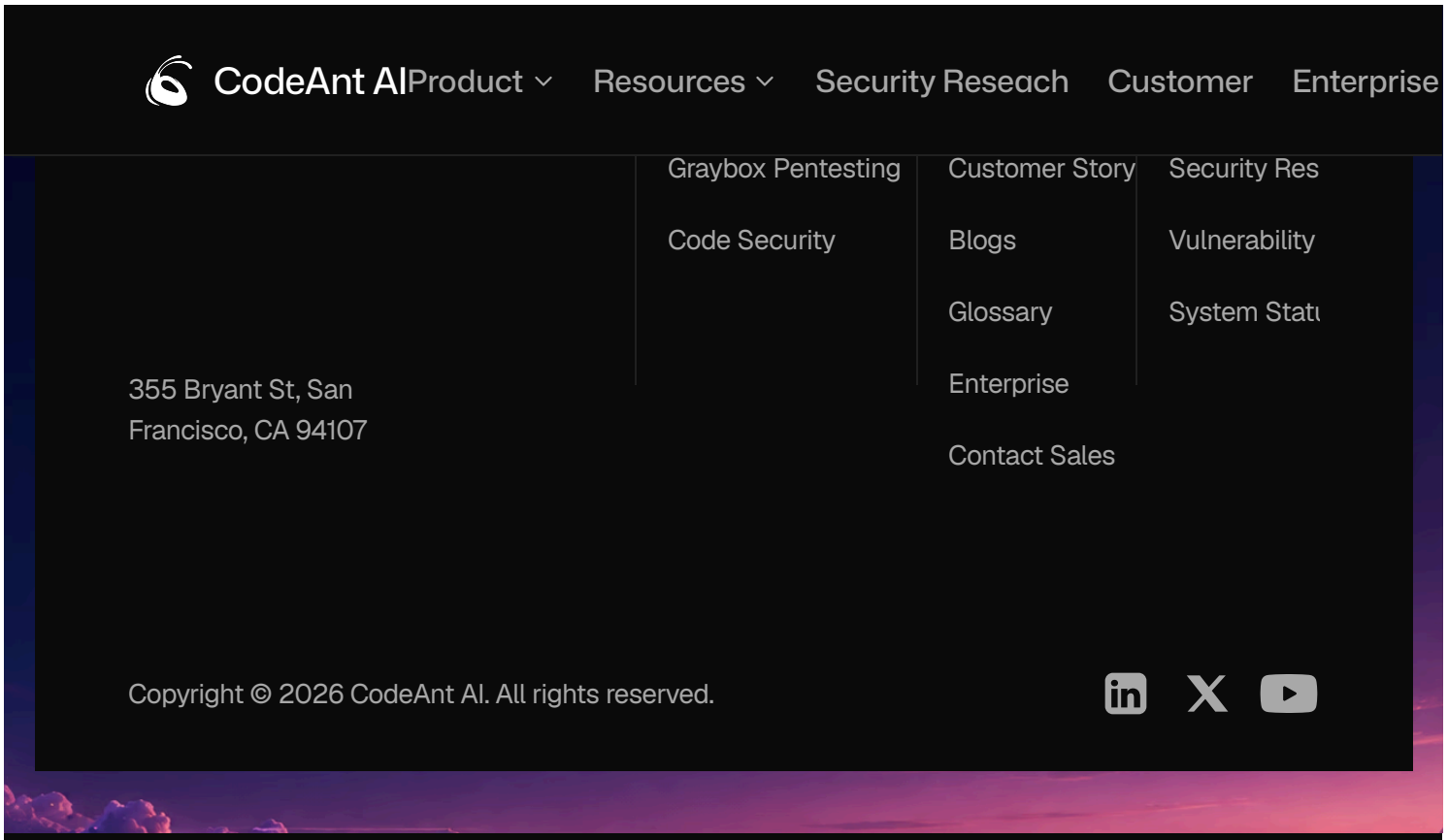
CVE-2026-31988: jszip DoS via Malformed ZIP File

CVE-2026-31988 CVSS 6.9

[GET STARTED]

Find out what's already exploitable in your codebase.

START PENTEST BOOK A DEMO


CodeAnt AI
Product
Resources
Security Reseach
Customer
Enterprise
Graybox Pentesting
Code Security
Customer Story
Blogs
Glossary
Enterprise
Contact Sales
Security Res
Vulnerability
System Statı
355 Bryant St, San
Francisco, CA 94107
Copyright © 2026 CodeAnt AI. All rights reserved.