BACK

CVE-2026-71507: Dolibarr BOLA Redirects Supplier Payments

`CVE-2026-71507` `CVSS 6.5`


Amartya Jha

Aug 23, 2026

What if a junior sales login could change the bank account a supplier gets paid into, and your next payment run quietly sent the money to an attacker, with the totals, transaction count, and every payee name exactly as expected?

That is what CodeAnt AI Security Research found in Dolibarr, the open-source ERP and CRM that thousands of small and mid-size businesses use to pay their suppliers.

CVE-2026-71507 is a broken object-level authorization bug in Dolibarr's Third Parties REST API. Three routes, `createCompanyBankAccount()`, `updateCompanyBankAccount()`, and `deleteCompanyBankAccount()`, check whether the caller can edit companies in general, but never whether they can touch this particular company. Their sibling read route already performs that object-level check.

CodeAnt found the mismatch by comparing authorization decisions across the read and write paths, then followed the successful write all the way into a generated SEPA payment file. The result: a low-privilege API key could replace a supplier's IBAN, and the next payment run sent funds to the attacker's account while leaving the visible payment details unchanged.

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.

Two Routes Into the Same Bank Record

The routes that create, update, and delete a supplier's bank details check whether the caller is generally allowed to edit companies. They stop there.

Whether the caller may touch this particular company never comes up. The sibling route that reads those same bank details does run that check. So a caller refused when trying to read a supplier's bank account can turn around and rewrite it.

The key doing this holds nothing unusual, just the routine create-companies right, the kind a salesperson or a CRM-sync integration carries. Nothing about the target supplier has to be within its reach. The split is stark in the responses. Reading a supplier's bank account comes back 403 Forbidden. Rewriting it comes back 200 OK.

From there the rewritten IBAN becomes the creditor on the next generated SEPA payment file. In our test the totals, transaction count, and payee names all stayed the same, so an approver checking the usual payment metadata would see the batch they expected with only the destination accounts changed underneath it.

Severity is Medium, CVSS 6.5, with the full vector in the table below. Dolibarr 24.0.0 closes it.

Attribute	Details
CVE	CVE-2026-71507
Affected component	Dolibarr Third Parties REST API bank-account write routes (`createCompanyBankAccount()`, `updateCompanyBankAccount()`, `deleteCompanyBankAccount()`)
Issue	Broken Object Level Authorization (CWE-639), with Missing Authorization (CWE-862) as a secondary weakness
CVSS	6.5 Medium (`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N`)
Required permission	The routine create-companies right, plus a valid API key
Confirmed build	25.0.0-alpha develop build
Fixed version	Dolibarr 24.0.0
Discovered by	CodeAnt AI Code Security Team

Where This Started

Working through Dolibarr's Third Parties REST API, our research team had already seen one version of a pattern. The route that reads a company's portal accounts ran a careful per-object check, and a sibling write route in the same class skipped it.

So we kept walking the write routes in that file, one at a time, looking for the same gap. We weren't just checking whether a permission existed; we were comparing what each route actually required against what its closest sibling required.

The bank-account routes had it. The read route confirmed you owned the company before it handed back its bank details. The three routes that create, update, and delete those details did not.

We rewrote a supplier's IBAN with a key that could not even read that supplier. The API returned 200. But a 200 is just a row in a database. The question we could not put down was simpler: does anything downstream actually trust that IBAN?

So we followed it. We generated the supplier payment run, and the SEPA file it produced paid the attacker's account, with the same total, the same three transactions, and the same supplier names as before. Only the destination had moved.

That is where we landed.

How Dolibarr Is Supposed to Guard This

Before the bug, the design it breaks. Dolibarr authorizes a REST call to a company in two parts.

The first is a role check: does this user hold the general right to create or modify a third party? That's coarse. It says you're the kind of user who edits companies.

The second is an object check: does this user own the specific company named in the request? That's the check that separates "you may edit companies" from "you may edit this company."

The read route for bank accounts runs both, refusing anyone who is not a commercial contact of the company:

That's why, in the lab, reading a supplier's bank accounts returned 403 for our test user.

Affected code

- Read route, runs both checks correctly: `DolibarrApi::_checkAccessToResource()` at `api_thirdparties.class.php:1904-1906`

What We Found

The three write routes that change the same rows sit a couple hundred lines down in the same file:

- `createCompanyBankAccount()`
- `updateCompanyBankAccount()`
- `deleteCompanyBankAccount()`

Their entire authorization story is a coarse right check.

The primary flaw is the missing per-object check. The right check only asks whether you may modify companies at all, never whether you may touch this company.

The row-versus-path check just below it looks like a guard, but it only confirms the bank row belongs to the company named in the path; it says nothing about whether the caller may reach either one.

From there, every supplied field, including the IBAN, BIC, and default flag, gets copied onto the object with no allowlist. The other two routes, `createCompanyBankAccount()` and `deleteCompanyBankAccount()`, open with the identical right check and the same missing per-object guard.

There is a second authorization gap worth noting. Dolibarr's web UI also enforces a dedicated payment-information permission for editing bank details. The REST routes do not check that permission either.

That is separate from the primary flaw above: even if the API added the missing object-level check, it would still not mirror the UI's dedicated permission model.

So the check is not missing from the codebase. It's missing from three functions that live right next to the one that has it. One surface enforces object access. A second surface writing the identical records forgets to.

Affected code

- Vulnerable routes, primary flaw, missing per-object check: `updateCompanyBankAccount()` (`api_thirdparties.class.php:2042`), `createCompanyBankAccount()` (`:1985`), `deleteCompanyBankAccount()` (`:2098`)
- Row-versus-path check that isn't a substitute for the missing guard: `api_thirdparties.class.php:2055`
- Secondary gap: the dedicated payment-information right (id 130, `societe.thirdparty_paymentinformation.write`, `modSociete.class.php:202-206`, enforced in the UI at `paymentmodes.php:95`) is never checked by any of the three write routes

Proof of Concept

CodeAnt AI did not stop at proving that the API would accept an unauthorized write. We followed the modified bank record into the business workflow that consumes it: supplier payment generation.

Every request below ran against a throwaway Dolibarr in local Docker, on the hardened 25.0.0-alpha develop build, with install locked, production mode on, HTTPS forced, and CSRF at maximum. No public or third-party system was involved, no file was ever handed to a bank, and no money moved.

The attacker principal was a non-admin with exactly the read and create-companies rights, deliberately without the payment-information right or the right that would let it see restricted customers.

Control. Prove the read boundary is real.

Baseline. Generate the legitimate payment run before touching anything. Three validated EUR invoices produced a SEPA payment file totaling EUR 6,600 across 3 transactions, a test fixture, not the scale of the risk, with the legitimate supplier IBANs as creditors: `FR1420041010050500013M02606`, `DE89370400440532013000`, and `BE68539007547034`.

Exploit. Rewrite a bank account the attacker key cannot even read.

The same key that had been refused a read on this supplier a moment earlier was allowed to rewrite where its money goes. Repeating it for the other two suppliers swapped in `NL91ABNA0417164300` and `ES9121000418450200051332`.

An administrator's decrypted read-back confirmed the database now held the attacker IBANs, while the attacker key still got 403 reading the very rows it had just authored.

Confirm. Regenerate the payment file and diff it. That's the punchline. The new SEPA file was byte-for-byte the same batch: same transaction count, same total, same supplier names, same invoice references. Two fields changed per line: the creditor IBAN and its BIC.

In this run, an approver reviewing the usual payment metadata would have seen precisely the batch they expected, with only the destination account swapped underneath it.

The security impact here is not the specific total in this lab run. It's that the payment workflow trusted an attacker-controlled bank account while preserving every other piece of legitimate transaction metadata.

A Second Impact: Object Enumeration

The same missing guard on delete also let the attacker remove a supplier's bank record outright, dropping the row count from 1 to 0.

And the 403-versus-200 split on the write route doubles as a one-bit oracle: an attacker can map bank-row IDs to companies they cannot otherwise read, just by watching which writes succeed.

Why a 200 Becomes Payment Fraud

The create-companies right is not an exotic privilege. It's the grant you hand a salesperson, a purchasing clerk, or a CRM-sync integration, precisely the kind of account an organization also fences out of viewing sensitive or unrelated customer records.

That combination is the whole problem. The role that edits company data is often the one kept from reading it, and here the fence only stands on the read side.

The payoff is silent supplier-payment redirection. That means an approver reviewing the usual payment metadata could see the expected batch while the creditor accounts had been changed underneath it.

Catching it needs a line-by-line creditor reconciliation against an out-of-band vendor master. The audit trail doesn't save you either: the change event records the new label but not the previous IBAN, so the original account can't be reconstructed after the fact.

This is one node in the same pattern behind the rest of this research: two write paths into the same set of rows disagreeing about who may touch them. We call it split-brain authorization.

What This Finding Actually Shows

What this finding shows is why authorization testing cannot stop at asking whether an endpoint returns 403.

CodeAnt found a route that had a permission check, returned the expected success response for authorized users, and looked ordinary under a conventional endpoint-level review.

The flaw appeared only when we compared that write decision with the authorization decision made by its sibling read route, then followed the resulting state change into the SEPA payment workflow.

That is the difference between checking whether an authorization check exists and testing whether the application actually enforces the boundary it claims to.

If you want that kind of test run against your own APIs, start with a free CodeAnt pentest.

Are You Affected?

If you run Dolibarr through the affected builds and any non-admin holds the create-companies right, that user can rewrite supplier bank details through the REST API today. Upgrade to Dolibarr 24.0.0.

Version 24.0.0 does the obvious thing: it adds the same per-object check the read route already had to all three write routes, so an unauthorized caller now gets the same 403 whether it reads or writes, which also closes the object-enumeration oracle noted above.

Related research. This finding is one node in the split-brain authorization pattern, the pillar behind all nine findings. The same missing object check on Third Parties write routes, taken all the way to a full customer-portal takeover, is documented in CVE-2026-71505. Read the two together to see how far one absent guard reaches.

[FAQ]

Frequently Asked Questions

Can CVE-2026-71507 redirect payments without changing the invoice amount? +

CVE-2026-71507 is a CWE-639 broken-object-level-authorization vulnerability in Dolibarr's Third Parties REST API. A low-privilege API user can modify the IBAN and BIC of a supplier it cannot read. When Dolibarr later generates a SEPA pain.001 payment file, it uses the modified bank details, allowing the payment destination to be redirected without changing the supplier name, invoice amount, or payment total. +

What is a BOLA vulnerability, and how does CVE-2026-71507 illustrate CWE-639? +

Which Dolibarr versions are affected, and how is CVE-2026-71507 fixed? +

How does editing a company's bank details become payment fraud? +

Why does the redirected SEPA file pass a four-eyes approval? +

What privileges does an attacker need to exploit CVE-2026-71507? +

More from CodeAnt

VIEW ALL

[GET STARTED]

START PENTEST BOOK A DEMO