BACK

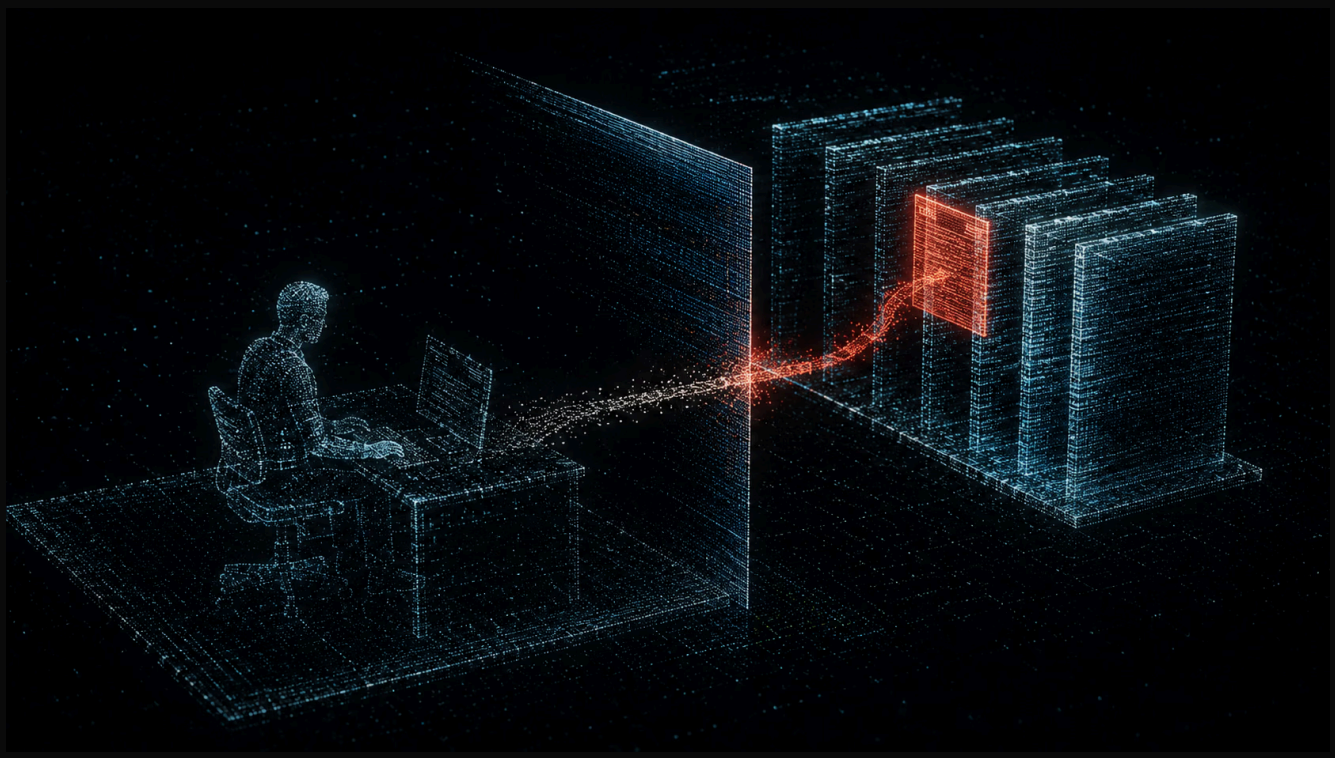
CVE-2026-71508: Dolibarr Mass Assignment to Payroll Tampering

CVE-2026-71508 CVSS 6.5



Amartya Jha

Aug 23, 2026



Research

- The Deny-List Named Five Fields, and Salary Wasn't One
  - Where This Started
  - Where Dolibarr Keeps Salary
  - What We Found
    - Affected code
  - Proof of Concept
  - Who Can Do It, and Why It Is Hard to Catch
  - What This Finding Actually Shows
    - Are You Affected?

BOOK A DEMO

refuses them access, and the change barely shows up in the response?

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

CVE-2026-71508 is a mass-assignment bug in Dolibarr's User REST API. The route that updates a user record copies every field from the request body onto that record, filtered only by a short deny-list of five password-related fields.

Salary, bonus, and the hourly and daily cost rates are plain columns on the same record, and none of them are on that list.

CodeAnt found the gap by asking a simple question of every mass-assignment loop in Dolibarr's API: what sensitive field is missing from the list of forbidden ones?

Salary was missing from this one. The result: an account holding only the ordinary edit-your-own-profile right, the same account the Salaries API refuses with a 403, can rewrite its own salary, bonus, and cost rates in a single request, and the 200 response hides four of the five values it just changed.

It carries a CVSS score of 6.5.

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 Deny-List Named Five Fields, and Salary Wasn't One

The route that updates a user record copies every field in the request body onto that record. Its only gate is a deny-list of five names, all of them passwords and API keys.

Reading those fields back out is another story. Four of the five are stripped from the response unless the caller holds salary-read rights. Writing them is guarded by nothing at all, which means the write lands and the response never shows it happened.

The account needed here is unremarkable. The ordinary edit-your-own-profile right, or the user-administration write right. Neither has anything to do with payroll. Send those payroll fields to the Salaries API and the same account gets 403 Forbidden. Send them to the Users API and it gets 200 OK.

In our test, an account with no payroll rights at all set its own salary, bonus, hourly rate, daily rate, and weekly hours to whatever it wanted, and the numbers flowed straight into Dolibarr's own payroll export report.

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

Attribute	Details
CVE	CVE-2026-71508
Affected component	Dolibarr User REST API (`Users::put()`)
Issue	Mass Assignment (CWE-915), OWASP API3:2023
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 edit-your-own-profile right, or the user-administration write right
Exposed fields	`salary`, `salaryextra`, `thm`, `tjm`, `weeklyhours`
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?

safe to write. The User API's loop blocked exactly five fields, all of them passwords and keys.

We weren't just checking whether that deny-list existed. We were asking what it left out. So we asked the obvious question: what sensitive field is not on that list of five?

Salary was not on it. Neither were the hourly and daily cost rates, or contractual hours. They're plain columns on the user record, and nobody had named them. So we sent a PUT that named salary, from an account with no payroll rights at all. It came back 200, and almost empty. That was the unsettling part. The response showed nothing had changed.

We read the database directly. All five payroll columns held the new numbers, written by an account that the Salaries API had refused with a 403 seconds earlier.

That is where we landed.

Where Dolibarr Keeps Salary

Before the bug, one detail of how Dolibarr is built makes the whole thing possible.

Dolibarr has a Salaries module with its own permission family, its own screens, and an export-gated report. What it doesn't have is a table of its own for the base figures. The reference salary, extra pay, hourly cost rate, daily cost rate, and contractual weekly hours are plain columns on the user record itself. The same row is reachable from the Users screen and from the Salaries screen.

That shared row is the whole problem. A control placed on the Salaries side means nothing if the Users side can write the same columns without it.

What We Found

The update loop in the User API walks the request body and assigns each key onto the user object. Its only gate is a five-name deny-list.

```
        if (in_array($field, array('pass_crypted'
            continue;
        }
        $this->useraccount->$field = $this->_chec
    }
```

The five payroll columns aren't on that deny-list, and they aren't declared in the object's field map either, so the sanitizer treats them as generic text and their numeric content arrives intact.

The update then writes them with no permission consideration anywhere on that path. Between the end of the authorization block and this assignment, there is no salaries or HR check at all. That's the missing guard, and a deny-list that names credentials while forgetting payroll is why it's missing.

The proof that these fields were meant to be protected sits in the same class, on the read path. When the API returns a user, it builds a read-salary flag from the salary-read rights and strips four of the five payroll fields from the response unless the caller holds them. Reading four of the five is guarded.

Writing any of the five is not. Same fields, same file, opposite answers. That asymmetry is also what hides the attack: a caller without the salary-read right gets a 200 whose body omits the four monetary values it just wrote, so a test that reads only the HTTP response looks like nothing happened.

Affected code

- Vulnerable update loop: `htdocs/user/class/api_users.class.php`, `Users::put()`, the field-assignment loop
- Deny-list that omits the payroll columns: `pass_crypted`, `pass_indatabase`, `pass_indatabase_crypted`, `pass_temp`, `api_key`, five names, none of them payroll
- Exposed fields: `salary`, `salaryextra`, `thm` (hourly cost rate), `tjm` (daily cost rate), `weeklyhours`
- The read-side guard that shows these fields were meant to be protected: `_cleanObjectDatas()`, same file, strips four of the five unless the caller holds salary-read rights

Proof of Concept

export.

Every request below ran against a throwaway Dolibarr in local Docker, hardened to a production profile with install locked, production mode on, HTTPS forced, and CSRF at its strictest. None of that blocked the write.

We tested two principals, each holding zero salaries rights, no group membership, and no admin flag. One holds the right to write another user's record. The other holds only the right to write its own.

Control. Prove the Salaries API refuses this account.

```
GET /api/index.php/salaries

→ HTTP 403 Forbidden
```

Exploit. Write another user's payroll through the Users route instead.

```
PUT /api/index.php/users/{USER_A}
{"salary":777777,"salaryextra":888888,"thm":7

→ HTTP 200, body echoes only: {"salary":false
```

The response cleaner strips the four monetary fields for this caller, so we confirmed the write against the database directly, where all five columns held the new values.

Confirm. Repeat it with the narrower, self-edit-only account. A second principal whose entire authority is editing its own profile, refused the Salaries API with a 403 seconds earlier, raises its own pay:

```
{"salary":999999,"salaryextra":888888,"thm":9
```

In this run, the record moved from 1000 / 100 / 10 / 100 / 35 to 999999 / 888888 / 9999 / 7777 / 1. Reading the corrupted row back through Dolibarr's built-in Salaries export, the same report finance actually runs, showed the tampered figure carried straight into a report this principal has no permission to see.

The expectation was straightforward. A request carrying any of the five payroll fields, from a caller with no salaries or HR right, should have been rejected, or those fields silently discarded, which is exactly what the web interface does and what this same API class already does for four of the five on the read side.

Instead the request returned 200 and all five columns were rewritten, moments after the Salaries API refused the same key with a 403.

Who Can Do It, and Why It Is Hard to Catch

The self-edit right is a routine grant, the kind of thing every employee has to edit their own contact details. The user-administration right is the ordinary user-management write.

Neither implies any authority over compensation, and this bug hands both exactly that. The self-edit variant is the one that squarely crosses a boundary: that principal has no legitimate route to any salaries right at all, and it can still set its own salary, bonus, and cost rates to any value it likes.

The damage doesn't stay in the user table. The altered salary is emitted by the export-gated Salaries and Payments report, so finance acts on a number written by a principal the same instance denies payroll access to.

And detection is hard: the 200 response conceals four of the five written values, and Dolibarr's security-event log is off by default, so the change realistically surfaces only by querying the user table directly or spotting a discrepancy later.

CVE-2026-71508 wasn't caused by a missing authorization check on an obviously sensitive endpoint. It was caused by a deny-list that named the fields someone remembered were dangerous, credentials and API keys, and missed the ones nobody thought to name.

The read path in the same file proves Dolibarr's own developers knew these fields needed protecting. They guarded them on the way out. They just never carried that same rule to the way in.

We found this not by asking whether the PUT returned 200, but by asking whether the fields it accepted could cross a permission boundary and change stored payroll data. That's the difference between checking whether a write succeeds and testing what a successful write is actually allowed to touch.

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 account holds either the self-edit or user-administration write right, that account can set payroll fields through the Users API today. Upgrade to Dolibarr 24.0.0.

Dolibarr 24.0.0 closes the gap the way the read side already modeled it. Before the assignment, the update route now refuses the five payroll keys unless the caller satisfies a write-salary check built from the salary-read rights, the same rule the web form already wraps around exactly these five fields.

The deeper correction is to stop filtering by exclusion. A five-name deny-list fails the moment a sensitive column is added that nobody remembers to list. An allow-list inverts the default, so only fields explicitly declared writable get written, kept as two sets: a profile set for self-write that reaches no HR or cost field, and an administrative set for user-write.

When a mass-assignment write is filtered by naming known-bad fields rather than known-good ones, the safe default is off by exactly the fields nobody thought to name, and payroll on a shared user record is precisely the kind of field that gets forgotten.

fields that fall through are payroll, and the result is an employee setting their own pay.

[FAQ]

Frequently Asked Questions

Can CVE-2026-71508 change more than an employee's salary? +

What is the difference between mass assignment and broken object-level authorization in Dolibarr? +

Which Dolibarr versions are affected, and how is CVE-2026-71508 fixed? +

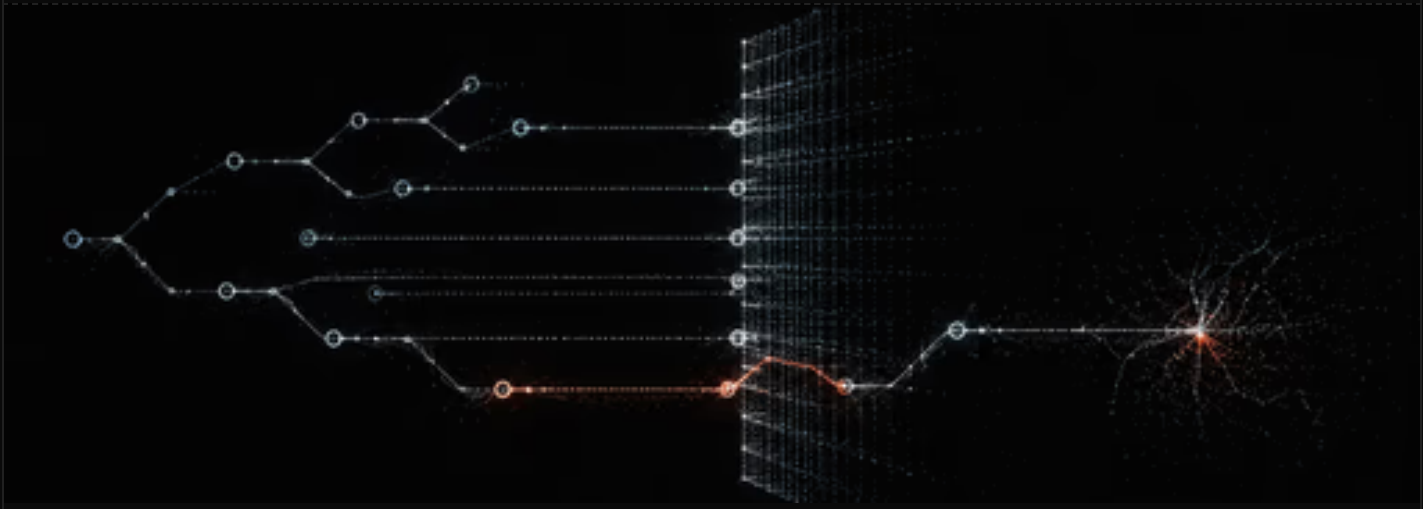
What privileges does an attacker need to exploit CVE-2026-71508? +

What is a mass-assignment vulnerability, and how does CVE-2026-71508 work? +

Why does the API response make CVE-2026-71508 hard to notice? +

More from CodeAnt

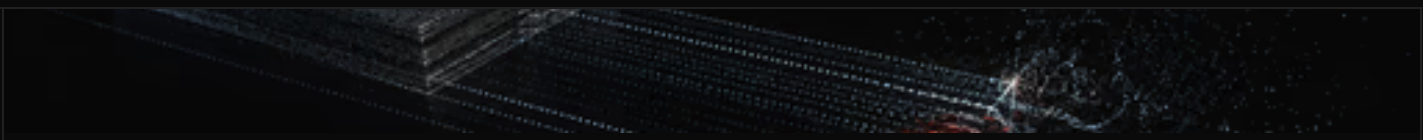
VIEW ALL



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

CodeAnt AI Product Resources Security Reseach Customer Enterprise

Find out what's already exploitable in your codebase.

START PENTEST BOOK A DEMO

CodeAnt AI

PRODUCT
Blackbox Pentesting
Whitebox Pentesting
Graybox Pentesting
Code Security

COMPANY
About
Pricing
Customer Story
Blogs
Glossary
Enterprise
Contact Sales

DEVELOPERS
Documentati
Integrations
Security Res
Vulnerability
System Statı

355 Bryant St, San Francisco, CA 94107

Copyright © 2026 CodeAnt AI. All rights reserved.