BACK

CVE-2026-71510: Dolibarr SQLFilters Bypass Exposes Payroll

CVE-2026-71510 CVSS 6.5


Amartya Jha

Aug 23, 2026

What if the same API that carefully refuses to show you anyone's salary would happily answer an unlimited number of yes-or-no questions about it, until you had rebuilt the exact figure?

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-2026-71510 is an incorrect-authorization bug in Dolibarr's User REST API. A key that can only list users can read every employee's salary, hourly cost, and daily cost, and even a scrambled copy of the stored password hash, none of which that key is ever shown in a normal response. It pulls each secret out one bit at a time, through the endpoint's own filter.

It carries a CVSS score of 6.5. This is a disclosure bug, not an account takeover.

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 Endpoint Won't Show You Salary, But It Will Answer Questions About It

The endpoint that lists users strips salary, hourly cost, and daily cost out of every response unless the caller holds the salary-read permission. That part works, and it runs on the way out every time.

The same endpoint also lets you filter the list by any field you name. That filter never runs the check. So you can't read a salary, but you can ask whether it's greater than X, and whether a row comes back is the answer. A few dozen of those questions pin the exact figure.

The key doing this holds only the ordinary right to list users, the grant you'd hand an HR assistant, a helpdesk agent, or a sync integration. The asymmetry is right there in the two requests. Fetch a user by id and the payroll fields are gone from the body. Filter on those same fields and you get a clean answer, every time.

In our test, a low-privilege key reconstructed a user's salary, extra salary, hourly cost, and daily cost exactly, then recovered a full password hash character by character, all from an endpoint that never displayed a single one of those values.

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

Attribute	Details
CVE	CVE-2026-71510
Affected component	Dolibarr User REST API (list endpoint filter)
Issue	Incorrect Authorization (CWE-863)
CVSS	6.5 Medium (`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`)
Required permission	The list-users right (`user->user->lire`)
Impact	Disclosure of salary, cost rates, and a scrambled password hash
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 coming back to ERP platforms, because an ERP holds the numbers a company most wants kept quiet, and payroll is near the top of that list. Dolibarr is exactly that kind of platform, an open-source ERP and CRM that organizations around the world use to run HR, accounting, and operations.

The endpoint that lists users is careful with salary. If you do not hold the salary-read permission, it quietly removes salary, hourly cost, and daily cost from the response before it sends it back. So the API plainly knows those numbers are sensitive, and it knows you are not allowed to see them.

But the same endpoint also lets you filter the list. You can ask it for users where some field is greater than a value, and that got us curious. The API refuses to show us the salary. Does it also refuse to let us filter on it?

It did not. We could not read the number, but we could ask whether it was above a value and get a straight yes or no. A handful of those answers is all it takes to pin the figure exactly. The field the endpoint had just stripped out of its own response, we read anyway, through its front door.

We kept following that trail, one question at a time, until we landed here.

How the Endpoint Is Supposed to Protect Salary

Picture asking a bank teller for a stranger's account balance. They refuse, and rightly, it is not yours to see. But then they cheerfully answer every yes-or-no question you put to them.

Is it over fifty thousand? Yes. Over seventy-five? No. Over sixty? Yes. A dozen questions later you know the balance to the euro, and the teller never told you a single number.

That is the shape of this bug. Dolibarr's user endpoint plays the careful teller on the way out. When it builds a response, it checks whether you hold the salary-read permission, and if you do not, it strips the pay fields out. That check is correct, and it runs on every response.

The gap is that the same endpoint accepts a filter, a way to say "only give me users matching this condition." When it turns your filter into a database query, it never runs that salary check again. It will build a query around any field you name, including the ones it would refuse to show you.

So filtering on salary is the yes-or-no question, and the list of users that comes back, empty or not, is the teller's answer.

What We Found

You cannot read the salary directly, but you can send a filter that says, in effect, "give me this user where their salary is greater than 234,566." If a row comes back, the answer is yes. If nothing comes back, the answer is no. Each request is one bit of information, one yes or one no.

From there it is a guessing game with a strategy. Ask whether the salary is above the midpoint of a range, and whichever way the answer falls, you have halved the range.

Repeat, and a few dozen questions pin any number exactly. The same trick, with a “starts with” comparison instead of “greater than,” walks a text value one character at a time, which is how even the scrambled password hash comes out, letter by letter.

It is worth being clear that this is not SQL injection, because that distinction is the whole diagnosis. Dolibarr scrubs the filter hard. Before it builds the query, it deletes every character an injection would need: the quotes, the spaces, the semicolons, the parentheses.

What survives is only the plain letters and dots of a column name. So you cannot break the query or smuggle in extra logic. The query that runs is exactly the shape the developers intended, well-formed every time.

The only freedom you have is which column name you write into it. Naming the salary column is not an attack on the query; it is a perfectly legal thing to write.

The bug is that no one ever checked whether you, specifically, are allowed to name that column. The query is valid. The permission is missing. That is why this is an authorization flaw and not an injection one.

There is even a bonus. Name a column that does not exist and the database returns its error verbatim, which quietly hands an attacker a map of the real column names before they have read a single value.

The response cleaner already knows salary is sensitive and gates it on the caller’s rights:

That salary-read decision is the one that matters. The class computed exactly who may see a salary, and used it to strip four columns from the outgoing response. That same decision never runs on the way in. The list endpoint checks one permission and then splices the filter straight into the query:

That first check authorizes listing users. Nothing between it and the filter call authorizes the columns the filter names. And the filter builder is what scrubs the input down to a bare column name:

That pattern removes every character an injection would need, which is why the result is always a valid query of the shape the developer drew. What it never does is ask whether this caller may reference the operand it left behind.

Affected code

- Vulnerable route: `htdocs/user/class/api_users.class.php`, `Users::index()`, the list-and-filter endpoint
- Check it runs: `hasRight('user', 'user', 'lire')`, whether the caller may list users at all
- Check it never runs on the filter: the salary-read decision computed in `_cleanObjectDatas()` (lines 1289-1336), which strips `salary`, `salaryextra`, `thm`, and `tjm` from the response but is never consulted before the filter reaches the query
- Where the filter is built unchecked: `htdocs/core/lib/functions.lib.php`, `dolForgeSQLCriteriaCallback()`, which sanitizes the column name's characters but never authorizes the column itself

Proof of Concept

Every request below ran against an isolated Dolibarr container on laptop loopback. No hosted or third-party instance was ever touched. The attacking key holds exactly the list-users and create-users rights, admin off, an ordinary account that can look people up.

Control. Confirm the endpoint actually hides the fields. Fetching the victim by id returns a body with salary, extra salary, hourly cost, daily cost, and the password hash all absent. The endpoint's stated position is that this key may not see them.

Exploit. Move the same column into the filter. Three requests, all HTTP 200:

A returned row means the condition held. An empty array means it did not. Those three bits pin the salary to 234,567.

Confirm. Scale the same technique up. Wrapped in a binary search over 0 to 16,777,216, each numeric column falls in 24 questions. Salary, extra salary, hourly cost, and daily cost came back as 234567 / 765432 / 2345 / 5432, matching the database exactly.

Switching the comparison to a "starts with" match walks the password hash one character at a time, and 1,286 questions recover the full 60-character value, in a lowercased form because the database's default collation makes the match case-insensitive.

The full run was 1,382 requests, every one an ordinary authenticated GET. Drop the targeting and a single filter for salary above a threshold returns everyone over that band at once.

What Someone Can Actually Read

The list-users permission is not a privileged grant. It is what you hand an HR assistant, a helpdesk agent who looks people up, or an integration account that syncs users into another system.

Any one of those keys can now read the whole workforce's pay, base and extra salary and both cost rates, from an endpoint built to withhold exactly those numbers from exactly those callers.

It is worth stating the ceiling plainly. This is disclosure, not takeover. What actually comes out is the four payroll figures and a scrambled, lowercased copy of the password hash.

That copy is not a usable credential, because logging in needs the exact original string and the lowercased version matches a huge number of possible hashes, so there is no shortcut to cracking it.

The same yes-or-no trick can answer questions about the admin flag and the API key too, but no key value was extracted and no account is taken over here. What leaks is confidential data the application had already decided this caller could not read.

What This Finding Actually Shows

CVE-2026-71510 wasn't caused by a filter that let attackers break the query. Dolibarr's filter sanitizer does its job: it strips every character an injection would need, and the query that runs is always well-formed.

The failure was that a filter which guarantees your query parses is not an access control. Keeping the SQL well-formed and deciding who may reference a column are two different jobs, and passing the first tells you nothing about the second.

Because this same filter splice appears across dozens of Dolibarr's REST classes, the durable version of the fix puts the column check in one shared place, so a new endpoint inherits it instead of having to remember it.

We found this by proving a low-privilege key could reconstruct data the API deliberately refused to return, not by spotting suspicious-looking SQL.

If your own APIs expose filter, search, sort, or query parameters, the same question applies: sanitizing what a user can write into a query does not decide what they are allowed to read, so authorize the fields they can reference, not just the syntax they can use.

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 anyone holds the list-users permission, that account can read salaries and cost rates through the filter today. Upgrade to Dolibarr 24.0.0.

Dolibarr 24.0.0 stops trusting the caller to name columns. The fix runs the same authorization decision on the filter that the response already runs on the output.

It allowlists which columns a filter may reference, reuses the existing salary-read check to reject the pay columns for callers who lack it, makes the never-returned credential columns off-limits as filter targets for everyone including admins, and stops handing the raw database error back so the schema map goes quiet.

Related research. This finding is one node in the split-brain authorization pattern, the pillar behind all nine findings, where one filter can pass and still leak. A close cousin is CVE-2026-71506, where the check also runs and still lands on the wrong answer, because the payments API validates the wrong permission. Both are the same underlying weakness: a control that executes but does not protect what it should.

[FAQ]

Frequently Asked Questions

Is CVE-2026-71510 SQL injection? +

What is a blind boolean oracle, and how does it leak data in CVE-2026-71510? +

Can the password hash leaked by CVE-2026-71510 be used to log in? +

What privileges does an attacker need to exploit CVE-2026-71510? +

Which Dolibarr versions are affected, and how is CVE-2026-71510 fixed? +

Why does sqlfilters bypass the protection on hidden user fields? +

Can CVE-2026-71510 be exploited without knowing the victim's user ID? +

More from CodeAnt

VIEW ALL

[GET STARTED]

START PENTEST BOOK A DEMO