BACK

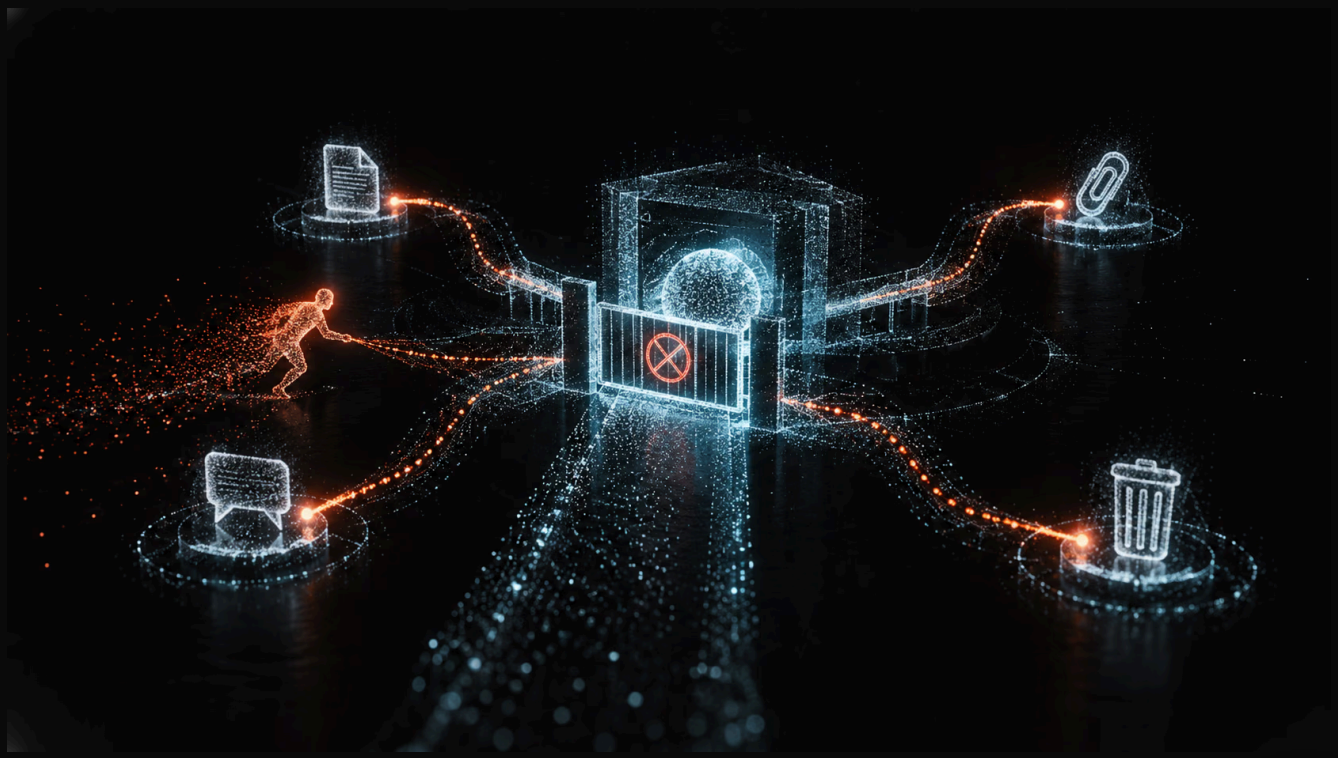
CVE-2026-109431: The Authorization Check Four Camunda Task Endpoints Skip

CVE-2026-109431 8.6



Hemanshu Baviskar

Oct 15, 2026



Research

- TL;DR
  - At a Glance
  - Two Helpers, One Letter Of Difference In Intent
  - Which Methods Call Which
  - Nothing Behind It
  - The Lab, Including The Part That Works Correctly
  - What This Does Not Do
  - Severity, And a Scoring Mistake Worth Describing
  - Fix And Detection
    - Credit & Scope

BOOK A DEMO

invoke only the harmless one, and behind them the engine commands contain no authorization check at all.

Severity	High, 8.1 (CVSS 3.1) / 8.6 (CVSS 4.0)
Confirmed on	`camunda/camunda-bpm-platform:run-7.24.0`, source commit `ee4826e5e7`
Status	CVE-2026-109431, reported through VulnCheck as CNA

TL;DR

- **Defect:** `ensureHistoryEnabled()` is a feature check. `ensureTaskExists()` is the authorization boundary, because the history query behind it is authorization-filtered. `getComment()`, `getAttachment()`, `getAttachmentData()` and `deleteAttachment()` call only the first.
- **Impact:** An authenticated account with no permission on any resource read a task comment and an attachment it was refused seconds earlier through the list endpoints, then permanently deleted the attachment. The delete removes the row and the content blob.
- **Precondition:** Authentication and authorization enabled, plus a task id and object id that correctly pair. Ids are not enumerable through the API, which we verified rather than assumed.
- **Status:** CVE-2026-109431, 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 comment and attachment endpoints
Issue	High: four task endpoints skip their authorization check, letting a zero-grant user read comments and attachments and permanently delete attachments
CVE	CVE-2026-109431
Class	Missing authorization: a feature check used where the authorization guard was needed

	skip `ensureTaskExists()`, the authorization-filtered guard. The engine commands behind them have no authorization check of their own
Impact	Read a task comment body, attachment metadata and file contents; permanently delete an attachment (row and content blob)
Requires	Authentication and authorization enabled, an authenticated account with no grants, and a task ID plus object ID that pair correctly. IDs are not enumerable through the API
Origin	Commit `536442d842` (November 29, 2024) added the guard to new comment-mutation methods only, leaving these four without it
Tested on	`camunda/camunda-bpm-platform:run-7.24.0`, source commit `ee4826e5e7`
Severity	8.6, High CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N 8.1, High CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N
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**

Two Helpers, One Letter Of Difference In Intent

Camunda's task comment and attachment endpoints are small REST wrappers. Before touching data, each one calls a guard or two off the top of the method. There are two guards available, and from a call site they look interchangeable.

`ensureHistoryEnabled()` checks that the engine's history level is not `none`. It is a feature-availability test and it carries no security weight whatsoever. `ensureTaskExists()` looks like a sanity check on a path parameter, and is actually the entire authorization boundary for these endpoints:

```
    HistoricTaskInstance historicTaskInstance =
        .createHistoricTaskInstanceQuery().taskI
    if (historicTaskInstance == null) {
      throw new InvalidRequestException(status,
    }
  }
```

That query runs through the engine's authorization filter, so for a caller without permission on the task it returns null and the request is refused. The method name says nothing about any of this.

A reviewer skimming a call site sees two `ensure*` calls and reads them as a pair of preconditions. Only one of them denies anybody.

Which Methods Call Which

Across the two resource classes, nine methods call both guards and four call only the feature check. The split does not follow read versus write, or old versus new, or anything else that would suggest a decision:

Method	History check	Authorization check	Outcome for a zero-grant calle
`getComments() :53`	yes	yes `:58`	404, correct
`deleteComment() :81`	yes	yes `:83`	404, correct
`updateComment() :95`	yes	yes `:97`	404, correct
`getAttachments() :57`	yes	yes `:62`	404, correct
`getComment() :70`	yes	**no**	200, full comment body
`getAttachment() :75`	yes	**no**	200, attachmer metadata

`ment()` `:102`			destroyed

`deleteAttachment()` at `:101-103` and `deleteComment()` at `:81-83` are the same shape minus one line. That is the whole bug, and it is why comment deletion is correctly refused today while attachment deletion is not.

The asymmetry is recent and datable. Commit `536442d842` (2024-11-29, *"feat(core): add update and delete comment Java/Rest APIs"*) added `ensureTaskExists(Status.NOT_FOUND)` to the three comment-mutation methods it was introducing, and to nothing else. It did not touch the pre-existing `getComment()` and it did not touch the attachment resource at all. The same commit added `TaskCommentAuthorizationTest`, which asserts `AuthorizationException` across twelve methods, every one of them an update or a delete. Read is never covered, and no attachment authorization test exists anywhere in the tree.

> A feature commit added the guard to the methods it was writing and tested exactly those. The guard's absence next door was never a decision anybody made.

Nothing Behind It

A missing check in a REST layer is usually survivable, because the engine command underneath typically runs its own `CommandChecker`. That is the pattern across most of Camunda. Not here. All four engine commands reached by these endpoints, `GetTaskCommentCmd`, `GetTaskAttachmentCmd`, `GetTaskAttachmentContentCmd` and `DeleteAttachmentCmd`, contain zero `CommandChecker` calls.

The missing REST line was not one layer of defence out of two. It was the only one.

The Lab, Including The Part That Works Correctly

the finding is the difference between them.

```
$ curl -u lowpriv:lowpriv "$B/engine-rest/tas
404
$ curl -u lowpriv:lowpriv "$B/engine-rest/tas
404
$ curl -u lowpriv:lowpriv "$B/engine-rest/tas
404      # <- the list endpoints refuse, cor
```

Three correct refusals. The caller cannot see the task and cannot list either collection on it. The singular endpoints then hand over what the list endpoints just withheld, to the same caller, in the same session:

```
$ curl -u lowpriv:lowpriv "$B/engine-rest/tas
{"id":"a3a85400-...","userId":"demo","taskId"
 "message":"SECRET-comment-body", ...}

$ curl -u lowpriv:lowpriv "$B/engine-rest/tas
SECRET-attachment-body

$ curl -u lowpriv:lowpriv -X DELETE "$B/engi
204

$ curl -u demo:demo "$B/engine-rest/task/$TI
404      # <- the owner's own object is gone
```

Read, download, destroy. The last line is the owner, not the attacker.

And the contrast that proves this is a missing call rather than an absence of authorization in general. The same caller, the same session, the operation that received its guard in 2024:

One line of difference in the source, and the entire security outcome flips.

What This Does Not Do

Two limits, both measured, both of which bound the finding honestly.

Ids are not enumerable, and the attacker needs two of them, not one. Both lookups key on the pair: `GetTaskAttachmentCmd.java:44` calls `findAttachmentByTaskIdAndAttachmentId(taskId, attachmentId)`, and `GetTaskCommentCmd.java:48` is the equivalent for comments. A task id alone is useless. Every listing endpoint we tried returns an empty array for a zero-grant caller:

```
task             []
history/task     []
history/detail   []
```

The default id generator is `StrongUuidGenerator`. We make no brute-force claim and did not attempt one.

Deletion is logged. Our report claimed none of these operations is logged distinctly, and that is wrong for the destructive one. `DeleteAttachmentCmd.java:75-88` writes a `UserOperationLogEntry` of type `OPERATION_TYPE_DELETE_ATTACHMENT` against the task. The three read paths produce nothing, but a destroyed attachment does leave a trail naming the caller.

Severity, And a Scoring Mistake Worth Describing

CVSS 4.0	`AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N`	**8.6 High**

Our first pass scored this 5.9 and 6.0, both Medium. That was a double count, and since the inflated-severity direction gets all the scrutiny it seems worth showing the other kind.

The id requirement from section 5 was priced twice. Once as `AC:H`, on the grounds that the attacker must obtain an identifier not available through the API. Then again as `I:L`, on the grounds that the amount of data an attacker can affect is limited to objects whose ids they happen to know. One obstacle, deducted from two independent metrics. CVSS metrics are scored independently by construction, and charging a barrier twice is not conservatism, it is arithmetic error.

Integrity is `I:H`. The spec's test is whether the attacker can modify any or all files protected by the impacted component. Attachments are literally files, the attacker can reach any of them engine-wide, and the delete is not recoverable:

```
// engine/.../impl/cmd/DeleteAttachmentCmd.ja
commandContext.getDbEntityManager().delete(at

if (attachment.getContentId() != null) {
  commandContext.getByteArrayManager()
      .deleteByteArrayById(attachment.getCont
}
```

The row and the content blob. No soft delete, no tombstone, nothing to restore from.

Attack complexity is `AC:L`, which is the metric most open to challenge. `AC:H` in CVSS 3.1 requires the attacker to invest measurable effort in preparation or execution. The primary realistic attacker invests none, and this is the path our own report already identifies as the realistic one: ids are long-lived, authorization is never re-checked, so a user who legitimately held access to a task keeps both ids and keeps the ability to read and destroy its

In CVSS 4.0 the question does not even arise. `AC` was redefined to mean evasion of security-hardening techniques such as ASLR or a cryptographic weakness, and `AT` covers deployment conditions such as a race or network-path injection. Needing a non-public identifier is neither, so `AC:L/AT:N` is the only defensible 4.0 reading and our `AC:H` there was a 3.1 metric copied across mechanically.

For a reviewer who wants the conservative floor: keeping `AC:H` while fixing only the double count gives 6.8 and **7.6 High**. Both readings land High in 4.0 once `I:H` is granted, so the disagreement moves the number and not the headline. We did not take `A:L` for the data destruction, which would give 8.3, because 3.1 availability concerns the component's availability rather than individual objects.

Fix And Detection

Add the line the guarded siblings already have. Four insertions, no new logic:

```
public AttachmentDto getAttachment(String att
  ensureHistoryEnabled(Status.NOT_FOUND);
  ensureTaskExists(Status.NOT_FOUND);
  ...
}
```

`getAttachment()` at `:75`, `getAttachmentData()` at `:88`, `deleteAttachment()` at `:102`, and `getComment()` at `:70`.

- **Rename the guard.** `ensureTaskExists` reads as a parameter sanity check and is an authorization boundary. Something like `ensureTaskVisibleToCaller` makes the next reviewer able to tell from a call site whether authorization was applied, which is the thing nobody could tell here.
- **Extend the tests to reads.** `TaskCommentAuthorizationTest` covers twelve mutation methods and no read. The attachment resource has no authorization test at all.
- **Detect the destructive case now.** Operation log entries of type `DeleteAttachment` whose user id holds no grant on the owning task are a clean signal, and they are already being written.

Finding	CodeAnt AI Security Research Team. The call-site audit, the lab reproduction and the engine-layer confirmation that nothing sits behind the REST check are ours.
Not claimed	No id enumeration. The attacker needs a task id and an object id that pair correctly, and we neither achieved nor attempted brute force.
Corrected here	Our report said none of these operations is logged distinctly. Attachment deletion is logged. Our first scoring pass also deducted the id requirement from two metrics at once, which is corrected in section 6.
Coordination	Reported through VulnCheck as CNA. CVE-2026-109431.
Testing	Locally-provisioned containers only. The only object destroyed was an attachment we created ourselves in the same lab run.

[FAQ]

Frequently Asked Questions

What is CVE-2026-109431? +

Does this affect every Camunda 7 installation? +

Does an attacker need an account? +

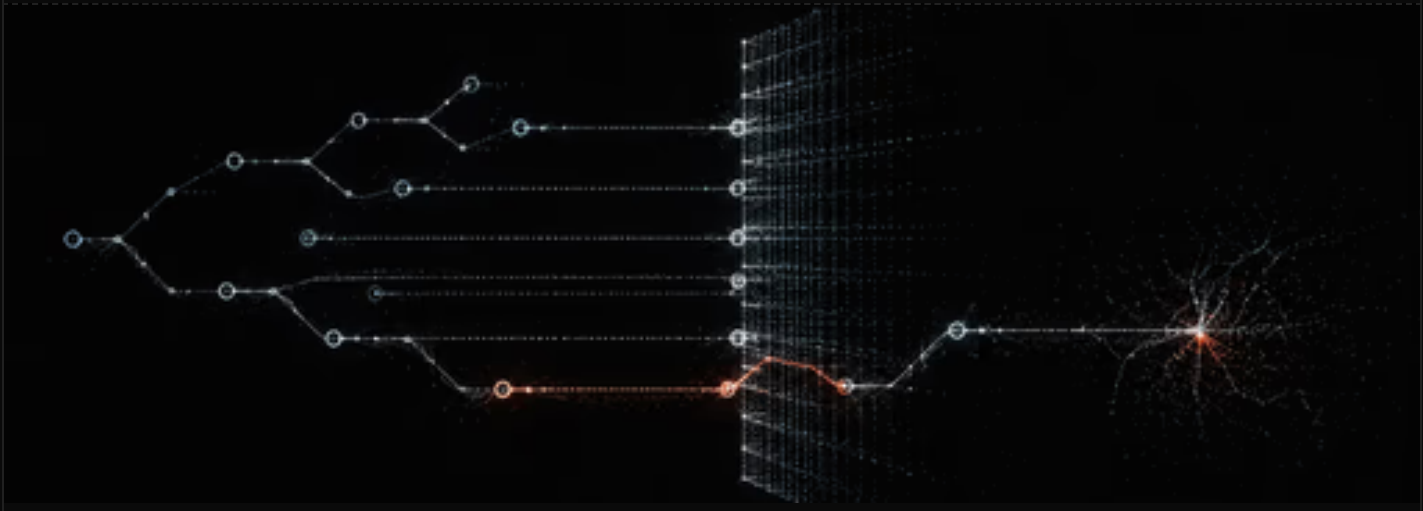
What can an attacker do? +

Can a deleted attachment be recovered? +

How do I fix CVE-2026-109431? +

More from CodeAnt

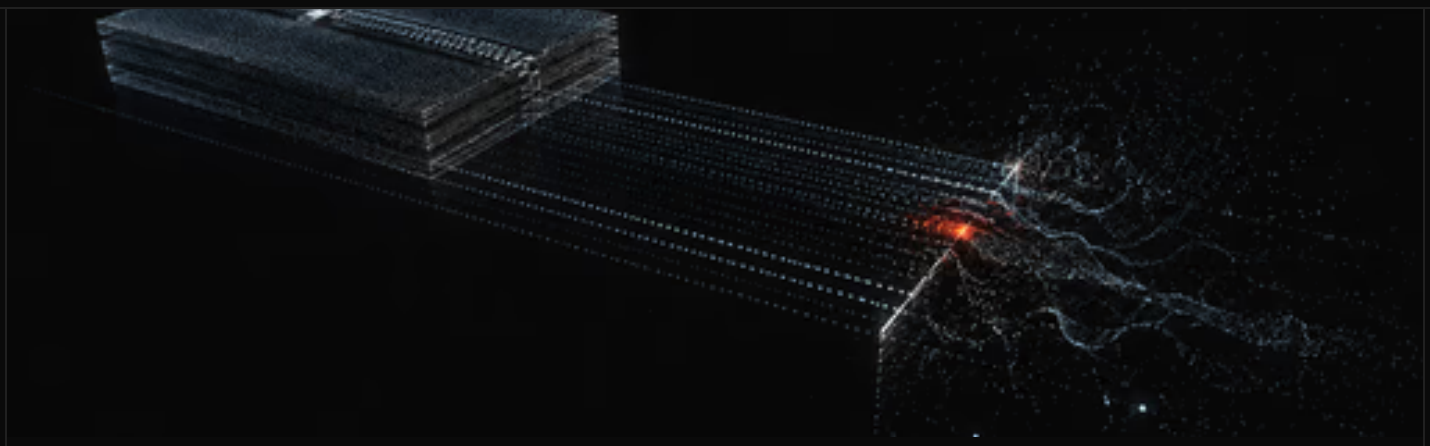
VIEW ALL



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

[GET STARTED]

Find out what's already exploitable in your codebase.

START PENTEST BOOK A DEMO

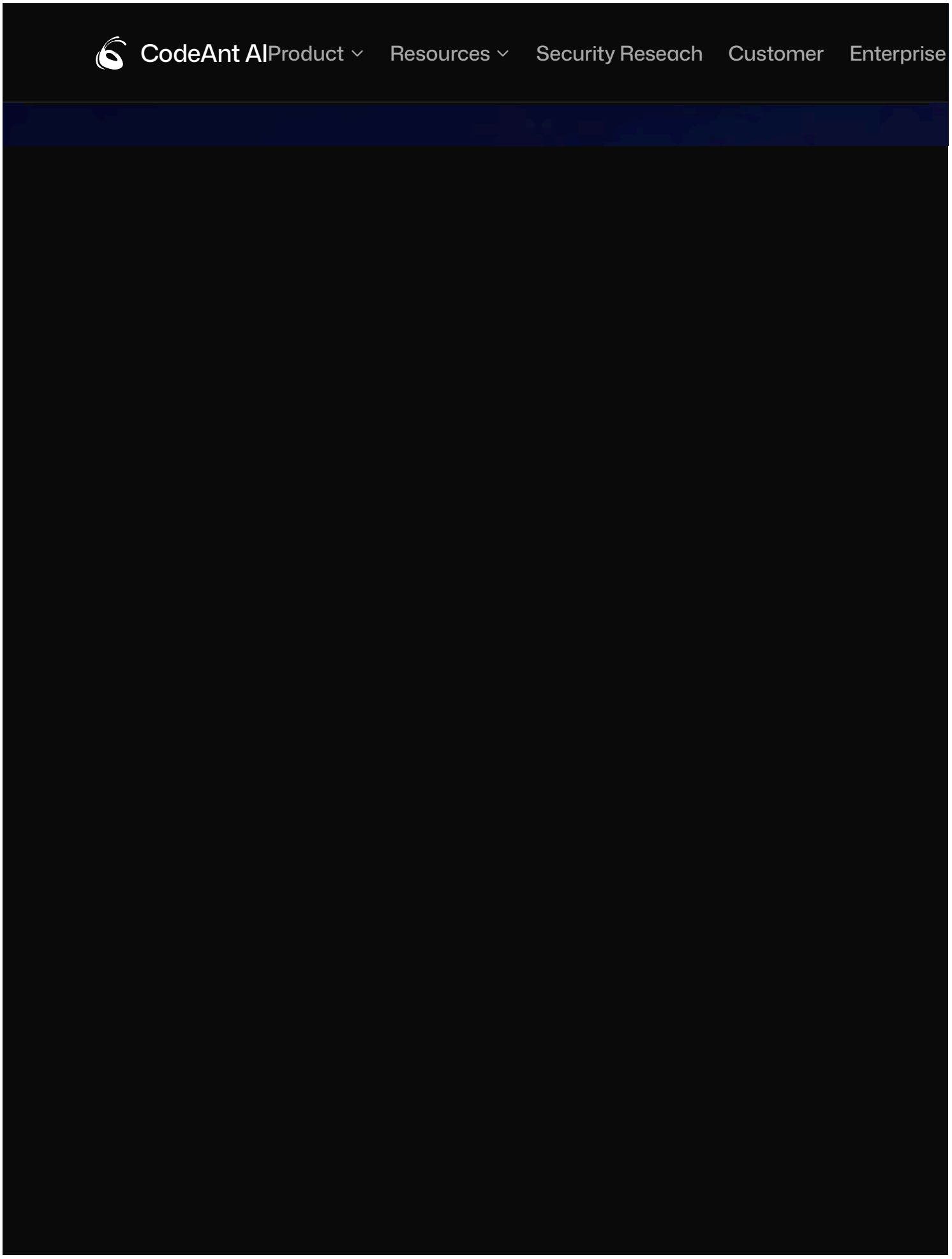
CodeAnt AI

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


CodeAnt AI
Product
Resources
Security Reseach
Customer
Enterprise