NoScope

Penetration Test Assessment Report

Example Helpdesk | Sample report

Created by NoScope
November 13, 2026

CLIENT CONFIDENTIAL

© NoScope

Document control

Record of scans included in this report

Scan	Date
Scan 1.1	October 12, 2026

Intended parties

Recipients authorised to receive and review this report

Client contact	Client company
Jane Doe	Example Corp

Table of contents

1 Executive summary 03
1.1 Overview 03
1.2 Purpose of the assessment 03
1.3 Impact summary 03
1.4 Improvement points 04
1.5 Conclusion 04
2 Scope 05
3 Severity definitions 06
4 Findings overview 07
4.1 Vulnerability distribution 07
4.2 Vulnerabilities table 08
4.3 Detailed vulnerability descriptions 15
5 Appendix A: About NoScope 40
6 Appendix B: Methodology 41
6.1 Vulnerability classes in scope of the methodology 42
7 Appendix C: Confidentiality statement and disclaimer 43

1. Executive summary

1.1 Overview

Example Corp engaged NoScope to run a continuous penetration assessment of the Example Helpdesk web application and API in the **Development** environment, combining **black-box** and **white-box** testing.

This report reflects the current state of validated findings for the primary target, **helpdesk.example.com**, as of November 13, 2026, including each finding's current status.

The assessment shows **57** validated findings: **1** critical, **6** high, **48** medium, **2** low and **0** info, each confirmed through proof-of-concept exploitation. This sample lists all 57 in section 4.2 and describes 6 of them in full in section 4.3.

1.2 Purpose of the assessment

This report documents the security findings identified through NoScope's continuous assessment of the **helpdesk.example.com** web application and its API. It gives the team a current view of validated vulnerabilities, their severity and their remediation status, so issues can be prioritized and tracked to closure. Each finding included has been confirmed through proof-of-concept exploitation.

1.3 Impact summary

The assessment identified multiple security findings across critical, high, medium, and low severity levels. Critical and high-risk vulnerabilities allow unauthorized privilege escalation, full tenant data access, account takeovers via guessable OTPs or session flaws, and exposure of sensitive service credentials and integration secrets. Other issues permit unauthorized cross-tenant data harvesting, ticket manipulation, internal note exposure, and server-side request forgery leading to potential denial of service.

1.4 Improvement points

Fix the critical email login code brute force first: the 4-digit code has no attempt limit, so any account whose email or username is known can be taken over. Next, address the high severity authentication and authorization flaws, including platform owner API keys that reach every company, API key impersonation, token retention after demotion, missing membership checks, raw secret exposure in API responses, and guest link leakage into the external database. Then remediate the medium issues involving tenant data isolation, ticket and metadata scoping, customer access to internal notes and other users' records, platform-level exposure of integration secrets, profile and password security gaps, server-side request forgery risks, bulk data handling limits, and permission bypasses on auxiliary endpoints.

1.5 Conclusion

The assessment surfaced vulnerabilities spanning critical, high, medium, and low severity levels, with critical and high findings presenting realistic pathways to unauthorised access and sensitive data exposure. Full remediation details and recommended mitigations are outlined in the findings section of this report.

It is recommended to prioritise remediation in order of severity and validate all fixes through structured retesting to confirm findings are fully resolved and no residual risk remains.

2. Scope

The following parameters defined the boundaries of this engagement

Assessment type	Web application and API security assessment (continuous)
Perspective	External, unauthenticated and authenticated
Environment	Development
First scan	October 12, 2026, 8:04 PM UTC
Latest scan	October 12, 2026, 8:04 PM UTC
Total scans	1
Primary	https://helpdesk.example.com
In-scope	https://helpdesk.example.com, https://api.helpdesk.example.com
Testing approach	Black-box and white-box (live testing plus source code review)
Credentials	Credentials were provided by the client (username/password login).
Exclusions	• Physical security testing • Social engineering and phishing campaigns • Denial-of-service attacks against production infrastructure • Testing of third-party services or external dependencies

3. Severity definitions

Vulnerabilities identified in this report are classified according to the following severity levels, based on the potential impact and exploitability of each issue

Critical | CVSS 9.0 – 10.0

Vulnerabilities that allow an attacker to fully compromise the application, gain unauthorized administrative access, or breach security boundaries with minimal complexity. Exploitation requires no specialized conditions and presents an immediate threat. Remediation is required urgently.

High | CVSS 7.0 – 8.9

Vulnerabilities that pose a significant and direct risk of unauthorized data access, privilege escalation, or service disruption. Exploitation is practical under normal conditions and does not require chaining with other vulnerabilities. Remediation should be a high priority.

Medium | CVSS 4.0 – 6.9

Vulnerabilities that present a moderate risk, typically requiring specific preconditions, configuration states, or chained exploitation to achieve significant impact. These should be addressed in the near term as part of ongoing security improvement.

Low | CVSS 0.1 – 3.9

Vulnerabilities with limited direct impact or low likelihood of exploitation. These represent security hardening opportunities and should be resolved as part of routine maintenance once higher-priority issues have been addressed.

Info | N/A

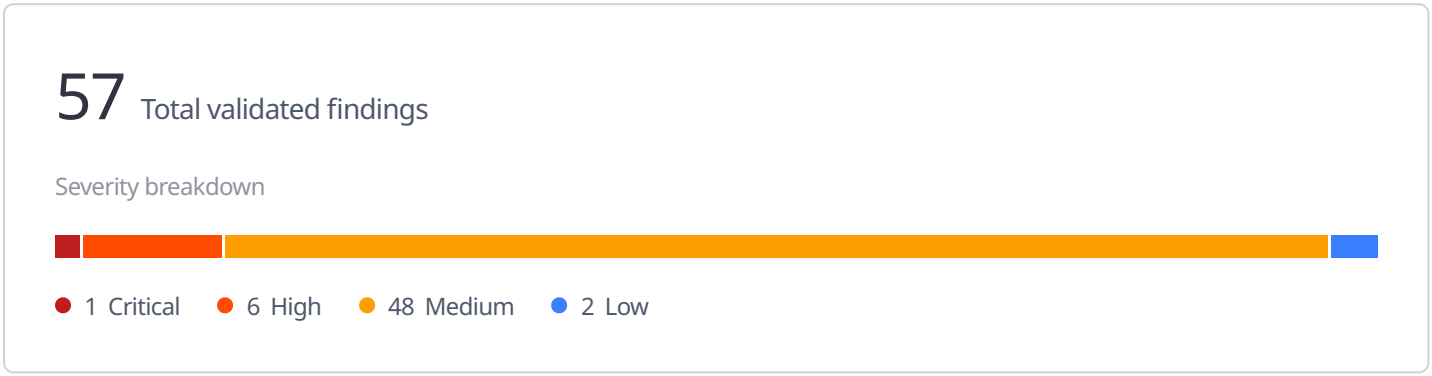
Observations that do not represent directly exploitable vulnerabilities but deviate from security best practices or may facilitate future attacks if left unaddressed. Resolution is at the organization's discretion.

4. Findings overview

This section reflects the current state of validated vulnerabilities across the assessment, ordered by severity. Each entry links to its detailed description with reproduction evidence and remediation guidance.

4.1 Vulnerability distribution

Distribution of validated findings by severity level



4.2 Vulnerabilities table

The table below lists every confirmed vulnerability identified during the assessment, ordered by severity

#	Finding	Severity (CVSS)	Status	Detected
1	4-digit email login code can be brute-forced into a full session	Critical 9.1	Open	Scan 1.1 October 12, 2026
2	Ticket sync copies secret guest links into the external database	High 8.5	Open	Scan 1.1 October 12, 2026
3	Platform owner API keys grant admin access to every company	High 8.1	Open	Scan 1.1 October 12, 2026
4	Company API keys can take over the account of the person who created them	High 7.6	Open	Scan 1.1 October 12, 2026
5	Settings API returns integration secrets in plain text	High 7.6	Open	Scan 1.1 October 12, 2026
6	API keys keep working after their creator loses admin rights	High 7.1	Open	Scan 1.1 October 12, 2026
7	API keys keep company access after the creator's membership ends	High 7.1	Open	Scan 1.1 October 12, 2026
8	Database admin key can be sent over unencrypted HTTP	Medium 6.9	Open	Scan 1.1 October 12, 2026
9	Platform company list exposes every customer's integration secrets	Medium 6.8	Open	Scan 1.1 October 12, 2026

#	Finding	Severity (CVSS)	Status	Detected
10	Settings page exposes integration secrets in the browser	Medium 6.8	Open	Scan 1.1 October 12, 2026
11	Customers can read staff internal notes through the external ticket API	Medium 6.5	Open	Scan 1.1 October 12, 2026
12	Customers can read every ticket in their company	Medium 6.5	Open	Scan 1.1 October 12, 2026
13	Inbound email can attach a ticket to any user on the platform	Medium 6.5	Open	Scan 1.1 October 12, 2026
14	External ticket API returns internal replies and skips visibility rules	Medium 6.5	Open	Scan 1.1 October 12, 2026
15	External ticket API skips customer visibility rules	Medium 6.5	Open	Scan 1.1 October 12, 2026
16	Email and password can be changed without the current password	Medium 6.5	Open	Scan 1.1 October 12, 2026
17	Unauthenticated inbound email triggers staff notifications and outbound mail	Medium 6.5	Open	Scan 1.1 October 12, 2026
18	External ticket list returns every company ticket to customers	Medium 6.5	Open	Scan 1.1 October 12, 2026

#	Finding	Severity (CVSS)	Status	Detected
19	Ticket import downloads remote files with no size limit	Medium 6.5	Open	Scan 1.1 October 12, 2026
20	Webhook URLs that contain secrets are written to application logs	Medium 6.4	Open	Scan 1.1 October 12, 2026
21	Ticket import lets staff make the server fetch internal URLs	Medium 6.4	Open	Scan 1.1 October 12, 2026
22	CSV export allows spreadsheet formula injection	Medium 6.1	Open	Scan 1.1 October 12, 2026
23	Inbound ticket subjects can trigger Slack mentions	Medium 5.8	Open	Scan 1.1 October 12, 2026
24	Platform owner role can read and change any company's data	Medium 5.5	Open	Scan 1.1 October 12, 2026
25	Database connection test fetches any stored URL and returns the response	Medium 5.5	Open	Scan 1.1 October 12, 2026
26	Webhook, Slack and database URLs are fetched with no address restrictions	Medium 5.5	Open	Scan 1.1 October 12, 2026
27	Teams can include users who aren't company members	Medium 5.5	Open	Scan 1.1 October 12, 2026

#	Finding	Severity (CVSS)	Status	Detected
28	Ticket merge can pull in tickets the user isn't allowed to see	Medium 5.4	Open	Scan 1.1 October 12, 2026
29	External ticket API accepts any user as requester or assignee	Medium 5.4	Open	Scan 1.1 October 12, 2026
30	Ticket update accepts tags that belong to other companies	Medium 5.4	Open	Scan 1.1 October 12, 2026
31	Ticket create accepts a requester from another company	Medium 5.4	Open	Scan 1.1 October 12, 2026
32	Login can resolve to the wrong account when an email matches another username	Medium 5.4	Open	Scan 1.1 October 12, 2026
33	Bulk assign accepts users and teams from other companies	Medium 5.4	Open	Scan 1.1 October 12, 2026
34	Staff can edit or delete other users' private canned replies	Medium 5.4	Open	Scan 1.1 October 12, 2026
35	Tags from one company can be attached to another company's tickets	Medium 5.4	Open	Scan 1.1 October 12, 2026
36	Guest links keep working for suspended companies	Medium 5.4	Open	Scan 1.1 October 12, 2026

#	Finding	Severity (CVSS)	Status	Detected
37	Password change doesn't sign out existing sessions	Medium 5.4	Open	Scan 1.1 October 12, 2026
38	Guest replies still trigger integrations for suspended companies	Medium 5.4	Open	Scan 1.1 October 12, 2026
39	Agents can edit another user's private canned reply	Medium 5.4	Open	Scan 1.1 October 12, 2026
40	Guest links still open tickets that were merged away	Medium 5.3	Open	Scan 1.1 October 12, 2026
41	Login code request reveals whether an account exists	Medium 5.3	Open	Scan 1.1 October 12, 2026
42	Admin and agent dashboards load unbounded ticket sets	Medium 5.0	Open	Scan 1.1 October 12, 2026
43	Database sync loads unlimited response bodies into memory	Medium 4.9	Open	Scan 1.1 October 12, 2026
44	Customers can set staff-only ticket fields and change the requester	Medium 4.3	Open	Scan 1.1 October 12, 2026
45	Ticket CSV export ignores the page limit and loads every row	Medium 4.3	Open	Scan 1.1 October 12, 2026

#	Finding	Severity (CVSS)	Status	Detected
46	Ticket workflow fields can be changed without a staff role	Medium 4.3	Open	Scan 1.1 October 12, 2026
47	CSV export leaks other users' emails and tag names	Medium 4.3	Open	Scan 1.1 October 12, 2026
48	Any company member can list every user and membership	Medium 4.3	Open	Scan 1.1 October 12, 2026
49	API keys can read unpublished knowledge base articles	Medium 4.3	Open	Scan 1.1 October 12, 2026
50	Customers can detect draft knowledge base articles	Medium 4.3	Open	Scan 1.1 October 12, 2026
51	Customers can read shared canned replies meant for staff	Medium 4.3	Open	Scan 1.1 October 12, 2026
52	Any ticket viewer can submit or overwrite satisfaction ratings	Medium 4.3	Open	Scan 1.1 October 12, 2026
53	Any company member can list every team's member names and emails	Medium 4.3	Open	Scan 1.1 October 12, 2026
54	External ticket view returns internal replies	Medium 4.3	Open	Scan 1.1 October 12, 2026

#	Finding	Severity (CVSS)	Status	Detected
55	External ticket view returns any company ticket to customers	Medium 4.3	Open	Scan 1.1 October 12, 2026
56	Pending invites aren't counted against the plan's staff seat limit	Low 3.8	Open	Scan 1.1 October 12, 2026
57	Leaving out the invite role bypasses the plan's staff seat limit	Low 2.7	Open	Scan 1.1 October 12, 2026

4.3 Detailed vulnerability descriptions

To keep this sample short, it includes full write-ups for 6 of the 57 findings, chosen to cover a range of severities and attack types. The other 51 are listed in section 4.2 and are documented to the same standard in client reports.

4.3.1 4-digit email login code can be brute-forced into a full session

Authentication bypass | CWE-307 | A07:2021 - Identification and Authentication Failures | Likelihood: High

The email login code is only 4 digits, and the server never slows down, locks out, or expires the code after a wrong guess. Any party that knows an account's email or username can try every code over the network and receive a full session, with no password and no action from the account holder.

Attack type	Authentication bypass
Severity	Critical | 9.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`
Status	Open
Detected	Scan 1.1 (October 12, 2026)
Endpoint	helpdesk.example.com/api/v1/auth/otp/verify (auth_controller.rb:30)

Impact

An attacker can take over any account whose email or username they know, including staff and administrators, and then read and change that account's tickets and workspace data for up to 30 days. That exposes customer conversations and lets the attacker act as the real user.

Technical analysis

Expected behaviour. Email login codes are a complete login. `POST /api/v1/auth/otp/verify` is a public route that takes only a login and a code, so the code is the only secret protecting the account and needs to resist guessing.

What happens. `generate_otp!` stores `format("%04d", SecureRandom.random_number(10_000))`, so there are only 10,000 possible codes, each valid for 10 minutes (`user.rb:50-53`). `verify_otp!` rejects a wrong code but doesn't count failures, lock the account, add a delay or clear the code (`user.rb:56-62`). A new `otp_request` installs a fresh code and a fresh 10-minute window with no backoff. No rate limiting middleware such as `Rack::Attack` is configured anywhere in the codebase.

Why it matters. On average about 5,000 guesses find the code, which fits inside one 10-minute window at a modest request rate. A correct guess calls `issue_session` (`auth_controller.rb:117-125`), which returns a JWT, the user record and every membership, exactly like a password login. The victim doesn't need to open the email.

Controls in place. Codes are compared with `secure_compare`, which correctly prevents timing attacks but doesn't limit the number of attempts.

Reproduction & evidence

Step 1

Request a login code for a known user. No password or session is required. An unknown login returns 404, which also confirms the account exists.

```
$ POST /api/v1/auth/otp/request
Content-Type: application/json

{"login": "admin@acme.test"}
```

```
200 OK
{"message": "OTP sent"}
```

Step 2

Brute-force every 4-digit code as a blind attacker. The server never throttles, locks out, or invalidates the code on a wrong guess.

```
$ for n in 0000..9999:
    POST /api/v1/auth/otp/verify
    {"login": "admin@acme.test", "otp": n, "remember_me": true}
```

```
wrong guesses rejected (HTTP 401):            7043
requests rate-limited / locked out (HTTP 429):  0
correct code accepted after 7044 attempts  -> 200 OK
session token issued (30-day, remember_me):    [REDACTED]

No lockout at any point. Average case is ~5,000 attempts,
well inside the 10-minute code window.
```

Remediation

1. Count failed attempts per code in `User#verify_otp!` and clear `otp_code` and `otp_sent_at` after a small limit (for example 5), so further guesses fail until a new code is issued.
2. Add rate limiting to `otp_request` and `otp_verify`, keyed by client IP and by normalised login, with a backoff that also limits how often new codes can be requested.
3. Lengthen the code to at least 6 to 8 digits, and return the same response for known and unknown logins on `otp_request`.
4. Keep `issue_session` behind these checks.

References

- https://cwe.mitre.org/data/definitions/307.html
- https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

4.3.2 Platform owner API keys grant admin access to every company

Access control | CWE-863 | A01:2021 - Broken Access Control | Likelihood: Medium

An API key created by a platform owner account is not confined to the company it was issued for. Any party that later holds the key can list every tenant, read each tenant's stored integration secrets, and create, re-plan, or suspend any company. Keys created by ordinary company admins carry none of this reach.

Attack type	Access control
Severity	High 8.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N`
Status	Open
Detected	Scan 1.1 (October 12, 2026)
Endpoint	helpdesk.example.com/api/v1/platform/companies (authenticatable.rb:57)

Impact

A leaked or third-party integration token issued by the platform owner becomes a master key for every tenant: their stored integration secrets and webhook URLs can be read, and any workspace can be suspended or reconfigured. That is a platform-wide confidentiality and integrity failure, not access to one company.

Technical analysis

Expected behaviour. API keys are company-scoped bearer credentials. The API documentation limits them to tenant routes and `/external/tickets`, where the key selects the company. Listing, creating, suspending and re-planning companies is a platform owner power that belongs to the owner's signed-in session, not to a company key.

What happens. When a request carries a `td_` token, `authenticate_api_key!` sets the current user to the key's creator and the current company to the key's company (`authenticatable.rb:29-37`). `PlatformController` skips company scoping and relies only on `require_platform_owner!` (`authenticatable.rb:57-60`). That check asks whether the user is a platform owner and never checks `api_key_auth?`, so the key's company binding is ignored.

Result. `companies_index` loads every company and renders `CompanySerializer.full`, including raw `settings`, `slack_webhook_url` and `outbound_webhook_url` (`platform_controller.rb:8-10`). `companies_create` and `companies_update` write plan, status and an arbitrary settings hash for any company ID (`platform_controller.rb:13-27`). Integration keys for Linear and Supabase are stored inside `settings`, so the list response discloses every tenant's integration secrets.

Limits. The key must have been created by a platform owner through the normal API key action. Keys created by company admins and normal user sessions are correctly refused by the same check. Because exploitation depends on holding a key that a platform owner created, this is rated High rather than Critical, and the key is scored as the low-privilege, company-scoped integration credential it is meant to be (PR:L).

Reproduction & evidence

Step 1

As the platform owner, mint a company API key scoped to Acme. The key records the owner as its creator.

```
$ POST /api/v1/api_keys
Authorization: Bearer <platform-owner JWT>
X-Company-Id: 7

{"api_key": {"name": "integration"}}
```

```
201 Created  (response trimmed)
{"api_key": {"name": "integration",
             "token": "[REDACTED]"}}
```

Step 2

Using only that td_ key, with no owner JWT and no company header, list every company. Each tenant's integration secrets are returned.

```
$ GET /api/v1/platform/companies
Authorization: Bearer [REDACTED]
```

```
200 OK  (3 companies; stored secret fields listed below)
Acme    supabase.service_role_key  [REDACTED]
Acme    linear.api_key             [REDACTED]
Acme    slack_webhook_url          [REDACTED]
Globex  supabase.service_role_key  [REDACTED]
```

Step 3

With the same key, suspend a different tenant (Globex, id 8) and downgrade its plan. The key's own company is never checked.

```
$ PATCH /api/v1/platform/companies/8
Authorization: Bearer [REDACTED]

{"company": {"plan": "free", "status": "suspended"}}
```

```
200 OK
{"company": {"id": 8, "slug": "globex",
             "plan": "free", "status": "suspended"}}
```

Remediation

1. In require_platform_owner!, return 403 when the request is authenticated by an API key, before the platform_owner check. Platform routes should accept only a platform owner's user session.
2. Stop treating the key creator's platform_owner flag as authority outside the key's own company.
3. Rotate any API keys that platform owner accounts created, and review platform audit logs for unexpected company changes.

References

- https://cwe.mitre.org/data/definitions/863.html
- https://owasp.org/Top10/A01_2021-Broken_Access_Control/

4.3.3 Platform company list exposes every customer's integration secrets

Information disclosure | CWE-200 | A01:2021 - Broken Access Control | Likelihood: Medium

The platform screen used to manage customer companies also returns each company's stored third-party secrets: database service-role keys, Linear API keys, and Slack webhook URLs. These values are masked everywhere else in the product, so one platform owner session exposes every tenant's external credentials in a single response.

Attack type	Information disclosure
Severity	Medium | 6.8
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:N/A:N`
Status	Open
Detected	Scan 1.1 (October 12, 2026)
Endpoint	helpdesk.example.com/api/v1/platform/companies (platform_controller.rb:13)

Impact

A platform-owner session, or a stolen platform-owner token, can take every customer's Supabase admin key, Linear API key, and Slack webhook link. Those credentials reach outside this helpdesk into the customers' own databases and chat tools, so one compromised operator login exposes all tenants' connected systems.

Technical analysis

Expected behaviour. Integration secrets are stored for server-side sync and aren't meant to reach the browser. The settings API masks Supabase and Linear keys to the last 4 characters (`settings_controller.rb:165-181`), `CompanySerializer.full` returns the outbound webhook secret only as a boolean, and the setup guide says the service-role key must stay server-side.

What happens. `companies_index`, `companies_create` and `companies_update` all render `CompanySerializer.full`, which copies `company.settings`, `slack_webhook_url` and `outbound_webhook_url` with no redaction (`company_serializer.rb:13-18`). The sync services read the same nested values as live credentials (`supabase_sync_service.rb:16-20`, `linear_sync_service.rb:23-26`), so the response contains working keys.

Result. One request with a platform owner token returns every tenant's Supabase service-role key, Linear API key and Slack webhook URL. A PATCH that only changes the plan still returns the stored secrets, because the serializer runs after the update. Existing sessions also survive a password change (finding 37), which extends the window for a stolen token.

Limits. The caller needs a platform owner session or token. Impact applies to every company that has saved an integration, which is the documented settings feature rather than a misconfiguration. The score counts the disclosure itself (C:H, I:N); scope is changed because the exposed keys reach the customers' own Supabase, Linear and Slack systems.

Reproduction & evidence

Step 1

Sign in as a platform owner. Platform routes need no company header.

```
$ POST /api/v1/auth/login

{"login": "owner@example.com", "password": "..."}
```

200 OK with a bearer token for a platform_owner account

Step 2

List every company. This is the response the platform screen loads. Each tenant's stored integration secrets come back in clear text.

```
$ GET /api/v1/platform/companies
Authorization: Bearer <platform-owner JWT>
```

```
200 OK  (3 companies; stored secret fields listed below)
Acme    supabase.service_role_key  [REDACTED]
Acme    linear.api_key             [REDACTED]
Acme    slack_webhook_url          [REDACTED]
Globex  supabase.service_role_key  [REDACTED]

Only has_outbound_webhook_secret is masked (boolean).
```

Remediation

1. Build the settings returned to the browser from an allowlist of non-secret fields. Replace service-role keys and API keys with a last-4 mask or a boolean.
2. Drop slack_webhook_url from the platform company list, or mask it the same way.
3. Ask affected tenants to rotate any integration keys that may have been exposed.

References

- https://cwe.mitre.org/data/definitions/200.html
- https://owasp.org/Top10/A01_2021-Broken_Access_Control/

4.3.4 Customers can read staff internal notes through the external ticket API

CWE-639 | A01:2021 - Broken Access Control | Likelihood: Medium | IDOR

Staff replies marked internal are hidden on the normal ticket API and returned in full on the external ticket API. A signed-in customer who opens their own ticket through the external route receives every internal note that agents wrote on it. The same route also returns other customers' tickets, so their internal notes are exposed the same way.

Attack type	IDOR
Severity	Medium 6.5
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`
Likelihood	Medium
CWE	CWE-639
OWASP Top 10	A01:2021 - Broken Access Control
Status	Open
Detected	Scan 1.1 (October 12, 2026)
Endpoint	helpdesk.example.com/api/v1/external/tickets/134

Impact

Customers can read notes that agents wrote only for each other, such as troubleshooting steps, escalation comments or remarks about the customer. The external serializer also returns other customers' tickets (findings 18 and 55), so internal notes on those tickets are exposed in the same response. The caller needs only a normal customer login. No note was created, edited or deleted during testing.

Technical analysis

Confirmed. NoScope confirmed this using a customer account that is the requester of ticket 134. The customer route `GET /api/v1/tickets/134` returns HTTP 200 with 2 replies, both public, so access to the ticket itself is correct.

What happens. Both routes load the ticket and embed a `replies` array. The customer handler drops replies where `is_internal` is true before serializing. The external handler serializes every reply, so the same token on `GET /api/v1/external/tickets/134` returns a third reply with `is_internal: true` and the body of a staff note.

Root cause. The failure is object-level filtering of child records. The external serializer doesn't apply the customer visibility rule to replies, and reply IDs and bodies are included in the payload, so the customer learns that the note exists and reads its text in a single request.

Attack path

1

Entry

Read own ticket on the customer route

```
GET /api/v1/tickets/134
```

Impact: HTTP 200, replies 342 and 343 only, both is_internal false

2

Impact

Read the same ticket on the external route

```
GET /api/v1/external/tickets/134
```

Impact: HTTP 200, extra reply 344 is_internal true, body Internal note: check logs for error code.

Reproduction & evidence

Step 1

As the customer who owns ticket 134, read it on the normal ticket API. Internal replies are filtered out.

```
$ GET /api/v1/tickets/134
Authorization: Bearer <customer JWT>
X-Company-Id: 7
```

```
200 OK, 2 replies, both public:
{"id": 342, "is_internal": false, "body": "Here is more context..."}
{"id": 343, "is_internal": false, "body": "Thanks, we're looking..."}
```

Step 2

Repeat on the external ticket API with the same customer token. The staff-only note is returned.

```
$ GET /api/v1/external/tickets/134
Authorization: Bearer <customer JWT>
X-Company-Id: 7
```

```
200 OK, 3 replies, including:
{"id": 344, "is_internal": true, "author_id": 31,
 "body": "Internal note: check logs for error code."}
```

Short-term mitigation

Filter internal replies out of `GET /api/v1/external/tickets/:id` for customer callers, using the same rule as `GET /api/v1/tickets/:id`. Don't rely on the client to hide `is_internal`.

Long-term fix

1. Apply one reply visibility rule in the external serializer: when the caller is a customer, drop every reply with `is_internal: true` before serialization.
2. Apply the same rule to the external list endpoint if it embeds replies, and to any future route that returns ticket replies.
3. Add a regression test asserting that a customer token never receives internal replies from any ticket endpoint.

References

- https://cwe.mitre.org/data/definitions/639.html
- https://owasp.org/Top10/A01_2021-Broken_Access_Control/

4.3.5 Ticket import lets staff make the server fetch internal URLs

SSRF | CWE-918 | A10:2021 - Server-Side Request Forgery | Likelihood: Medium

Ticket import fetches any URL a staff member supplies, including addresses only the server can reach, such as loopback, internal services, and the cloud metadata endpoint. When the response looks like a ticket file, its contents are stored as tickets the same account can open, turning the server into a proxy for internal systems.

Attack type	SSRF
Severity	Medium 6.4
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N`
Status	Open
Detected	Scan 1.1 (October 12, 2026)
Endpoint	helpdesk.example.com/api/v1/tickets/import (tickets_controller.rb:145)

Impact

An agent or admin can pull data from internal systems into the helpdesk and read it back as tickets, and can also make the server contact those systems even when the reply is not saved. That exposes internal services and any secrets they return, and it lets a normal staff account act from the server's network position.

Technical analysis

Expected behaviour. Ticket import is a staff action that loads a ticket file from a URL. Agents aren't meant to gain the application server's network position, such as loopback, private ranges or the cloud metadata service.

What happens. After the staff role check, `import` passes `params[:url]` straight to `URI.open(url, read_timeout: 15)` (`tickets_controller.rb:149-153`). There's no scheme allowlist, no host allowlist and no blocking of loopback, private or link-local addresses, and redirects are followed by default.

Read-back path. If the response starts with `[` or `{` it's parsed as JSON, otherwise as CSV, and each row is stored through `create_from_import` (`tickets_controller.rb:155-166`). The title and description then appear in the agent's ticket list. Responses that don't parse are still requested, so any internal endpoint can be called blind.

Limits. The attacker needs an agent or admin session. Customers and anonymous users are correctly rejected by `require_staff!`. Each call makes one request with a 15-second timeout. Testing demonstrated the loopback case; the same code path applies no restriction to private ranges or the cloud metadata address (169.254.169.254), which was established from the source rather than requested during testing.

Reproduction & evidence

Step 1

From an agent session, import from an internal-only URL. The server, not the browser, makes the request.

```
$ POST /api/v1/tickets/import
Authorization: Bearer <agent JWT>
X-Company-Id: 7

{"url": "http://127.0.0.1:8099/internal.json"}
```

```
200 OK
{"imported": 1}

127.0.0.1:8099 is reachable only from the application host.
```

Step 2

List tickets with the same session. The internal service's response was stored as a ticket the agent can read.

```
$ GET /api/v1/tickets
Authorization: Bearer <agent JWT>
X-Company-Id: 7
```

```
200 OK  (response trimmed)
{"title": "ssrf-proof",
 "description": "<content served by 127.0.0.1:8099>"}
```

Remediation

1. Parse the supplied URL, allow only http and https, resolve the host, and reject loopback, private, link-local (including 169.254.169.254), and unspecified addresses.
2. Disable redirects, or re-validate every redirect target with the same rules before connecting.
3. Prefer an explicit allowlist of import hosts, and cap the size of the fetched response.

References

- https://cwe.mitre.org/data/definitions/918.html
- https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/

4.3.6 Leaving out the invite role bypasses the plan's staff seat limit

Business logic | CWE-840 | A04:2021 - Insecure Design | Likelihood: Low

A company admin can add staff after the workspace has reached its plan's seat limit. Sending an invite with the role left off skips the seat check, and the invite is still saved as an agent invite. Accepting it adds another active staff member, so the plan limit and the revenue tied to it stop being enforced.

Attack type	Business logic
Severity	Low | 2.7
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:N/I:L/A:N`
Status	Open
Detected	Scan 1.1 (October 12, 2026)
Endpoint	helpdesk.example.com/api/v1/invites (invites_controller.rb:14)

Impact

A customer on a capped plan can keep adding unpaid support staff. Each extra person can handle tickets in that workspace, so plan limits and the revenue tied to them are not enforced.

Technical analysis

Expected behaviour. A company admin can invite staff only while the workspace is under its plan's seat limit (3, 15 or 100 seats depending on the plan). Explicit staff invites over the limit are rejected with `Agent limit reached for plan`.

What happens. The seat check only runs when the raw parameter is exactly `admin` or `agent` (`invites_controller.rb:18-21`). An omitted or null role skips the check, and the insert then stores `role || "agent"` (`invites_controller.rb:24-26`), so the invite is saved as an agent invite anyway.

No second check. `Invite#accept!` creates an active membership with the stored role and doesn't compare against the plan limit (`invite.rb:21-26`). The invite token is returned in the create response, so the admin can accept it without email delivery and add as many unpaid seats as they like.

Controls in place. Invites that explicitly set the role to `agent` or `admin` are correctly rejected when the plan is full.

Reproduction & evidence

Step 1

Initech is on the free plan (3 staff seats) and already at the cap. An explicit agent invite is correctly rejected.

```
$ POST /api/v1/invites
Authorization: Bearer <admin JWT>
X-Company-Id: 9

{"invite": {"email": "blocked@initech.test", "role": "agent"}}
```

```
422 Unprocessable Entity
{"error": "Agent limit reached for plan"}
```

Step 2

Send the same invite with the role omitted. The seat check is skipped and the invite is stored as an agent invite.

```
$ POST /api/v1/invites
Authorization: Bearer <admin JWT>
X-Company-Id: 9

{"invite": {"email": "extra-agent@initech.test"}}
```

```
201 Created
{"invite": {"role": "agent", "token": "[REDACTED]"}}
```

Step 3

Accept it as a new user. The workspace now has 4 active staff on a 3-seat plan, confirmed from the platform view.

```
$ POST /api/v1/auth/invites/accept

{"token": "[REDACTED]", "username": "extraagent",
 "password": "..."}
```

```
200 OK
Initech: plan=free  agent_limit=3  agent_count=4  (4/3 seats)
```

Remediation

1. Resolve one effective role before the seat check, defaulting to agent, reject any value outside the allowed roles, and run the check whenever that role is admin or agent.
2. Repeat the seat check inside the invite-accept step before creating an admin or agent membership, and count pending invites against the limit so they cannot be used to exceed it.

References

- https://cwe.mitre.org/data/definitions/840.html
- https://owasp.org/Top10/A04_2021-Insecure_Design/

5. Appendix A: About NoScope

NoScope is a continuous penetration testing platform that identifies exploitable vulnerabilities across web applications and APIs. Testing is performed through automated reconnaissance and targeted exploitation, aligned with industry-recognized standards including the OWASP Testing Guide and the Penetration Testing Execution Standard (PTES).

Each assessment follows a structured methodology covering reconnaissance, vulnerability discovery, exploitation, and reporting. Findings are validated through proof-of-concept exploitation to confirm impact before inclusion, so the report reflects confirmed issues rather than unverified alerts.

6. Appendix B: Methodology

The assessment followed a structured methodology covering reconnaissance, vulnerability discovery, exploitation, and reporting.

1. Reconnaissance & surface mapping

Headless-browser crawling, JavaScript bundle analysis, fuzz-driven endpoint discovery, and parsing of any available API specs to build a working map of routes, parameters, and trust boundaries.

2. Authentication & session management

Evaluation of login, registration, and OAuth flows, session token lifecycle and rotation, logout behavior, cookie security attributes, and resistance to credential-based and fixation attacks.

3. Authorization & access control

IDOR and BOLA probing across object identifiers, horizontal and vertical privilege escalation checks, and verification of tenant isolation on every authenticated endpoint reached during recon.

4. Injection & server-side flaws

Targeted payloads for SQL, command, template, and XML injection, SSRF and XXE on request-driven sinks, server-side request forgery against internal metadata endpoints, and unsafe deserialization where reachable.

5. Client-side & file handling

Reflected, stored, and DOM-based XSS across user-controlled sinks, CSP and framing controls, prompt injection on AI features, and content-type or magic-byte validation on upload paths.

6. Business logic & race conditions

Workflow abuse cases such as state manipulation, replay, and rate-limit bypass; race conditions on critical state changes; and rule violations specific to the application's intended use.

6.1 Vulnerability classes in scope of the methodology

NoScope's AI Agent is designed to test for the vulnerability classes below, mapped to the OWASP Top 10. Coverage on any individual scan depends on the application surface reached and the behavior exercised during that scan.

Category	Vulnerability classes
Injection and execution	SQL, NoSQL, command, template (SSTI), and XML injection; SSRF; insecure deserialization
Authentication and session	Broken authentication, session and token lifecycle flaws, fixation, logout and cookie security
Authorization and access control	IDOR and BOLA, horizontal and vertical privilege escalation, tenant isolation, CSRF
Client-side	Reflected, stored, and DOM-based XSS; CSP and framing controls; prompt injection on AI features; upload validation
Data exposure and logic	Sensitive data exposure, information disclosure via errors, business logic flaws
Configuration and crypto	Security misconfiguration, weak cryptographic handling, credential exposure

7. Appendix C: Confidentiality statement and disclaimer

This document and the information it contains are confidential and intended exclusively for the named recipient. Disclosure, copying, or distribution to any other party without prior written consent from NoScope is strictly prohibited. Findings reflect the security posture of the target as observed across the assessment and current as of the latest scan shown in this report and should be handled as restricted technical material.

This assessment does not constitute certification or accreditation against ISO 27001, ISO 27002:2022, NIST CSF, or any other security standard or framework. The output of this assessment may serve as a basis for a future certification or accreditation effort but should not be treated as such.